//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// ARM-specific macro definitions

#pragma once

#ifndef \_M\_AMD64

#error Include amd64.h in builds of AMD64 targets only.

#endif

extern "C" VOID amd64\_SAVE\_REGISTERS(void\*);

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// ARM-specific macro definitions

#pragma once

#ifndef \_M\_ARM

#error Include arm.h in builds of ARM targets only.

#endif

extern "C" LPVOID arm\_GET\_CURRENT\_FRAME(void);

extern "C" VOID arm\_SAVE\_REGISTERS(void\*);

/\*

\* The relevant part of the frame looks like this (high addresses at the top, low ones at the bottom):

\*

\* ----------------------

\* r3 <=== Homed input parameters

\* r2 <

\* r1 <

\* r0 <===

\* lr <=== return address

\* r11 <=== current r11 (frame pointer)

\* ...

\*/

const DWORD ReturnAddrOffsetFromFramePtr = 1;

const DWORD ArgOffsetFromFramePtr = 2;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// ARM64-specific macro definitions

#pragma once

#ifndef \_M\_ARM64

#error Include arm64.h in builds of ARM64 targets only.

#endif

extern "C" LPVOID arm64\_GET\_CURRENT\_FRAME(void);

extern "C" VOID arm64\_SAVE\_REGISTERS(void\*);

/\*

\* The relevant part of the frame looks like this (high addresses at the top, low ones at the bottom):

\*

\* ----------------------

\* r3 <=== Homed input parameters

\* r2 <

\* r1 <

\* r0 <===

\* lr <=== return address

\* r11 <=== current r11 (frame pointer)

\* ...

\*/

const DWORD ReturnAddrOffsetFromFramePtr = 1;

const DWORD ArgOffsetFromFramePtr = 2;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#if DYNAMIC\_INTERPRETER\_THUNK

#define DefaultEntryThunk Js::InterpreterStackFrame::DelayDynamicInterpreterThunk

#if \_M\_X64

#define AsmJsDefaultEntryThunk Js::InterpreterStackFrame::AsmJsDelayDynamicInterpreterThunk

#elif \_M\_IX86

#define AsmJsDefaultEntryThunk Js::InterpreterStackFrame::DelayDynamicInterpreterThunk

#endif

#else

#define DefaultEntryThunk Js::InterpreterStackFrame::InterpreterThunk

#endif

#define ProfileEntryThunk Js::ScriptContext::DebugProfileProbeThunk

#define DefaultDeferredParsingThunk Js::JavascriptFunction::DeferredParsingThunk

#define ProfileDeferredParsingThunk Js::ScriptContext::ProfileModeDeferredParsingThunk

#define DefaultDeferredDeserializeThunk Js::JavascriptFunction::DeferredDeserializeThunk

#define ProfileDeferredDeserializeThunk Js::ScriptContext::ProfileModeDeferredDeserializeThunk

#if ENABLE\_NATIVE\_CODEGEN

class NativeCodeGenerator;

class ThreadContext;

struct CodeGenWorkItem;

class NativeCodeData;

class StackSym;

class Func;

struct InlinedFrameLayout;

typedef intptr IntConstType;

typedef uintptr UIntConstType;

typedef IntMath<intptr>::Type IntConstMath;

typedef double FloatConstType;

#include "..\Backend\EmitBuffer.h"

#include "..\Backend\InterpreterThunkEmitter.h"

#include "..\Runtime\Bytecode\BackEndOpCodeAttr.h"

#include "..\Backend\BackEndOpCodeAttrAsmJs.h"

#include "..\Backend\CodeGenNumberAllocator.h"

#include "..\Backend\NativeCodeData.h"

#include "..\Backend\JnHelperMethod.h"

#include "..\Backend\IRType.h"

#include "..\Backend\InlineeFrameInfo.h"

NativeCodeGenerator \* NewNativeCodeGenerator(Js::ScriptContext \* nativeCodeGen);

void DeleteNativeCodeGenerator(NativeCodeGenerator \* nativeCodeGen);

void CloseNativeCodeGenerator(NativeCodeGenerator\* nativeCodeGen);

bool IsClosedNativeCodeGenerator(NativeCodeGenerator\* nativeCodeGen);

void SetProfileModeNativeCodeGen(NativeCodeGenerator \*pNativeCodeGen, BOOL fSet);

void UpdateNativeCodeGeneratorForDebugMode(NativeCodeGenerator\* nativeCodeGen);

CriticalSection \*GetNativeCodeGenCriticalSection(NativeCodeGenerator \*pNativeCodeGen);

bool TryReleaseNonHiPriWorkItem(Js::ScriptContext\* scriptContext, CodeGenWorkItem\* workItem);

void NativeCodeGenEnterScriptStart(NativeCodeGenerator \* nativeCodeGen);

bool IsNativeFunctionAddr(Js::ScriptContext \*scriptContext, void \* address);

void FreeNativeCodeGenAllocation(Js::ScriptContext\* scriptContext, void\* address);

CodeGenAllocators\* GetForegroundAllocator(NativeCodeGenerator \* nativeCodeGen, PageAllocator\* pageallocator);

void GenerateFunction(NativeCodeGenerator \* nativeCodeGen, Js::FunctionBody \* functionBody, Js::ScriptFunction \* function = NULL);

void GenerateLoopBody(NativeCodeGenerator \* nativeCodeGen, Js::FunctionBody \* functionBody, Js::LoopHeader \* loopHeader, Js::EntryPointInfo\* entryPointInfo, uint localCount, Js::Var localSlots[]);

#ifdef ENABLE\_PREJIT

void GenerateAllFunctions(NativeCodeGenerator \* nativeCodeGen, Js::FunctionBody \* fn);

#endif

#ifdef IR\_VIEWER

Js::Var RejitIRViewerFunction(NativeCodeGenerator \*nativeCodeGen, Js::FunctionBody \*fn, Js::ScriptContext \*scriptContext);

#endif

BOOL IsIntermediateCodeGenThunk(Js::JavascriptMethod codeAddress);

BOOL IsAsmJsCodeGenThunk(Js::JavascriptMethod codeAddress);

typedef Js::JavascriptMethod (\*CheckCodeGenFunction)(Js::ScriptFunction \* function);

CheckCodeGenFunction GetCheckCodeGenFunction(Js::JavascriptMethod codeAddress);

uint GetBailOutRegisterSaveSlotCount();

uint GetBailOutReserveSlotCount();

#if DBG

void CheckIsExecutable(Js::RecyclableObject \* function, Js::JavascriptMethod entryPoint);

#endif

#ifdef PROFILE\_EXEC

namespace Js

{

class ScriptContextProfiler;

};

void CreateProfilerNativeCodeGen(NativeCodeGenerator \* nativeCodeGen, Js::ScriptContextProfiler \* profiler);

void ProfilePrintNativeCodeGen(NativeCodeGenerator \* nativeCodeGen);

void SetProfilerFromNativeCodeGen(NativeCodeGenerator \* toNativeCodeGen, NativeCodeGenerator \* fromNativeCodeGen);

#endif

void DeleteNativeCodeData(NativeCodeData \* data);

#else

inline BOOL IsIntermediateCodeGenThunk(Js::JavascriptMethod codeAddress) { return false; }

inline BOOL IsAsmJsCodeGenThunk(Js::JavascriptMethod codeAddress) { return false; }

#endif

#if \_M\_X64

extern "C" void \* amd64\_ReturnFromCallWithFakeFrame();

#endif

struct InlinedFrameLayout

{

Js::InlineeCallInfo callInfo;

Js::JavascriptFunction \*function;

Js::Var arguments; // The arguments object.

//Js::Var argv[0]; // Here it would be embedded arguments array (callInfo.count elements)

// but can't have 0-size arr in base class, so we define it in derived class.

Js::Var\* GetArguments()

{

return (Js::Var\*)(this + 1);

}

template<class Fn>

void MapArgs(Fn callback)

{

Js::Var\* arguments = this->GetArguments();

for (uint i = 0; i < callInfo.Count; i++)

{

callback(i, (Js::Var\*)((uintptr\_t\*)arguments + i));

}

}

InlinedFrameLayout\* Next()

{

InlinedFrameLayout \*next = (InlinedFrameLayout \*)(this->GetArguments() + this->callInfo.Count);

return next;

}

};

class BailOutRecord;

struct LazyBailOutRecord

{

uint32 offset;

BYTE\* instructionPointer; // Instruction pointer of the bailout code

BailOutRecord\* bailoutRecord;

LazyBailOutRecord() : offset(0), instructionPointer(nullptr), bailoutRecord(nullptr) {}

LazyBailOutRecord(uint32 offset, BYTE\* address, BailOutRecord\* record) :

offset(offset), instructionPointer(address),

bailoutRecord(record)

{}

void SetBailOutKind();

#if DBG

void Dump(Js::FunctionBody\* functionBody);

#endif

};

struct StackFrameConstants

{

#if defined(\_M\_IX86)

static const size\_t StackCheckCodeHeightThreadBound = 35;

static const size\_t StackCheckCodeHeightNotThreadBound = 47;

static const size\_t StackCheckCodeHeightWithInterruptProbe = 53;

#elif defined(\_M\_X64)

static const size\_t StackCheckCodeHeightThreadBound = 57;

static const size\_t StackCheckCodeHeightNotThreadBound = 62;

static const size\_t StackCheckCodeHeightWithInterruptProbe = 68;

#elif defined(\_M\_ARM)

static const size\_t StackCheckCodeHeight = 64;

static const size\_t StackCheckCodeHeightThreadBound = StackFrameConstants::StackCheckCodeHeight;

static const size\_t StackCheckCodeHeightNotThreadBound = StackFrameConstants::StackCheckCodeHeight;

static const size\_t StackCheckCodeHeightWithInterruptProbe = StackFrameConstants::StackCheckCodeHeight;

#elif defined(\_M\_ARM64)

static const size\_t StackCheckCodeHeight = 58\*2;

static const size\_t StackCheckCodeHeightThreadBound = StackFrameConstants::StackCheckCodeHeight;

static const size\_t StackCheckCodeHeightNotThreadBound = StackFrameConstants::StackCheckCodeHeight;

static const size\_t StackCheckCodeHeightWithInterruptProbe = StackFrameConstants::StackCheckCodeHeight;

#endif

};

struct NativeResourceIds

{

static const short SourceCodeResourceNameId = 0x64;

static const short ByteCodeResourceNameId = 0x65;

static const short NativeMapResourceNameId = 0x66;

static const short NativeThrowMapResourceNameId = 0x67;

};

#if defined(\_M\_IX86)

struct ThunkConstants

{

static const size\_t ThunkInstructionSize = 2;

static const size\_t ThunkSize = 6;

};

#endif

enum LibraryValue {

ValueInvalid,

ValueUndeclBlockVar,

ValueEmptyString,

ValueUndefined,

ValueNull,

ValueTrue,

ValueFalse,

ValueNegativeZero,

ValueNumberTypeStatic,

ValueStringTypeStatic,

ValueObjectType,

ValueObjectHeaderInlinedType,

ValueRegexType,

ValueArrayConstructor,

ValuePositiveInfinity,

ValueNaN,

ValueJavascriptArrayType,

ValueNativeIntArrayType,

ValueNativeFloatArrayType,

ValueConstructorCacheDefaultInstance,

ValueAbsDoubleCst,

ValueUintConvertConst,

ValueBuiltinFunctions,

ValueJnHelperMethods,

ValueCharStringCache

};

enum VTableValue {

#if !\_M\_X64

VtableJavascriptNumber,

#endif

VtableDynamicObject,

VtableInvalid,

VtablePropertyString,

VtableJavascriptBoolean,

VtableSmallDynamicObjectSnapshotEnumeratorWPCache,

VtableJavascriptArray,

VtableInt8Array,

VtableUint8Array,

VtableUint8ClampedArray,

VtableInt16Array,

VtableUint16Array,

VtableInt32Array,

VtableUint32Array,

VtableFloat32Array,

VtableFloat64Array,

VtableJavascriptPixelArray,

VtableInt64Array,

VtableUint64Array,

VtableBoolArray,

VtableCharArray,

VtableInt8VirtualArray,

VtableUint8VirtualArray,

VtableUint8ClampedVirtualArray,

VtableInt16VirtualArray,

VtableUint16VirtualArray,

VtableInt32VirtualArray,

VtableUint32VirtualArray,

VtableFloat32VirtualArray,

VtableFloat64VirtualArray,

VtableNativeIntArray,

#if ENABLE\_COPYONACCESS\_ARRAY

VtableCopyOnAccessNativeIntArray,

#endif

VtableNativeFloatArray,

VtableJavascriptNativeIntArray,

VtableJavascriptRegExp,

VtableStackScriptFunction,

VtableConcatStringMulti,

VtableCompoundString,

// SIMD\_JS

VtableSimd128F4,

VtableSimd128I4,

Count

};

#if DBG\_DUMP || defined(ENABLE\_IR\_VIEWER)

const wchar\_t \*GetVtableName(VTableValue value);

#endif

enum AuxArrayValue {

AuxPropertyIdArray,

AuxIntArray,

AuxFloatArray,

AuxVarsArray,

AuxVarArrayVarCount,

AuxFuncInfoArray

};

enum OptimizationOverridesValue {

OptimizationOverridesArraySetElementFastPathVtable,

OptimizationOverridesIntArraySetElementFastPathVtable,

OptimizationOverridesFloatArraySetElementFastPathVtable,

OptimizationOverridesSideEffects

};

enum FunctionBodyValue {

FunctionBodyConstantVar,

FunctionBodyNestedFuncReference,

FunctionBodyReferencedPropertyId,

FunctionBodyPropertyIdFromCacheId,

FunctionBodyLiteralRegex,

FunctionBodyStringTemplateCallsiteRef

};

enum ScriptContextValue {

ScriptContextNumberAllocator,

ScriptContextCharStringCache,

ScriptContextRecycler,

ScriptContextOptimizationOverrides

};

enum NumberAllocatorValue {

NumberAllocatorEndAddress,

NumberAllocatorFreeObjectList

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

/\*\*\*

\* banned.h - list of Microsoft Security Development Lifecycle banned APIs

\*

\* Purpose:

\* This include file contains a list of banned API which should not be used in new code and

\* removed from legacy code over time

\* History

\* 01-Jan-2006 - mikehow - Initial Version

\* 22-Apr-2008 - mikehow - Updated to SDL 4.1, commented out recommendations and added memcpy

\* 26-Jan-2009 - mikehow - Updated to SDL 5.0, made the list sane, added compliance levels

\* 10-Feb-2009 - mikehow - Updated based on feedback from MS Office

\* 12-May-2009 - jpardue - Updated based on feedback from mikehow (added wmemcpy)

\*

\*\*\*/

// IMPORTANT:

// Some of these functions are Windows specific, so you may want to add \*nix specific banned function calls

#ifndef \_INC\_BANNED

# define \_INC\_BANNED

#endif

#ifdef \_MSC\_VER

# pragma once

# pragma deprecated (strcpy, strcpyA, strcpyW, wcscpy, \_tcscpy, \_mbscpy, StrCpy, StrCpyA, StrCpyW, lstrcpy, lstrcpyA, lstrcpyW, \_tccpy, \_mbccpy, \_ftcscpy)

# pragma deprecated (strcat, strcatA, strcatW, wcscat, \_tcscat, \_mbscat, StrCat, StrCatA, StrCatW, lstrcat, lstrcatA, lstrcatW, StrCatBuff, StrCatBuffA, StrCatBuffW, StrCatChainW, \_tccat, \_mbccat, \_ftcscat)

# pragma deprecated (wvsprintf, wvsprintfA, wvsprintfW, vsprintf, \_vstprintf, vswprintf)

# pragma deprecated (strncpy, wcsncpy, \_tcsncpy, \_mbsncpy, \_mbsnbcpy, StrCpyN, StrCpyNA, StrCpyNW, StrNCpy, strcpynA, StrNCpyA, StrNCpyW, lstrcpyn, lstrcpynA, lstrcpynW)

# pragma deprecated (strncat, wcsncat, \_tcsncat, \_mbsncat, \_mbsnbcat, StrCatN, StrCatNA, StrCatNW, StrNCat, StrNCatA, StrNCatW, lstrncat, lstrcatnA, lstrcatnW, lstrcatn)

# pragma deprecated (IsBadWritePtr, IsBadHugeWritePtr, IsBadReadPtr, IsBadHugeReadPtr, IsBadCodePtr, IsBadStringPtr)

# pragma deprecated (gets, \_getts, \_gettws)

# pragma deprecated (RtlCopyMemory, CopyMemory)

# pragma deprecated (wnsprintf, wnsprintfA, wnsprintfW, sprintfW, sprintfA, wsprintf, wsprintfW, wsprintfA, sprintf, swprintf, \_stprintf, \_snwprintf, \_snprintf, \_sntprintf)

# pragma deprecated (\_vsnprintf, vsnprintf, \_vsnwprintf, \_vsntprintf, wvnsprintf, wvnsprintfA, wvnsprintfW)

# pragma deprecated (strtok, \_tcstok, wcstok, \_mbstok)

# pragma deprecated (makepath, \_tmakepath, \_makepath, \_wmakepath)

# pragma deprecated (\_splitpath, \_tsplitpath, \_wsplitpath)

# pragma deprecated (scanf, wscanf, \_tscanf, sscanf, swscanf, \_stscanf, snscanf, snwscanf, \_sntscanf)

# pragma deprecated (\_itoa, \_itow, \_i64toa, \_i64tow, \_ui64toa, \_ui64tot, \_ui64tow, \_ultoa, \_ultot, \_ultow)

#if (\_SDL\_BANNED\_LEVEL3)

# pragma deprecated (CharToOem, CharToOemA, CharToOemW, OemToChar, OemToCharA, OemToCharW, CharToOemBuffA, CharToOemBuffW)

# pragma deprecated (alloca, \_alloca)

# pragma deprecated (strlen, wcslen, \_mbslen, \_mbstrlen, StrLen, lstrlen)

# pragma deprecated (ChangeWindowMessageFilter)

#endif

#ifndef PATHCCH\_NO\_DEPRECATE

// Path APIs which assume MAX\_PATH instead of requiring the caller to specify

// the buffer size have been deprecated. Include <PathCch.h> and use the PathCch

// equivalents instead.

# pragma deprecated (PathAddBackslash, PathAddBackslashA, PathAddBackslashW)

# pragma deprecated (PathAddExtension, PathAddExtensionA, PathAddExtensionW)

# pragma deprecated (PathAppend, PathAppendA, PathAppendW)

# pragma deprecated (PathCanonicalize, PathCanonicalizeA, PathCanonicalizeW)

# pragma deprecated (PathCombine, PathCombineA, PathCombineW)

# pragma deprecated (PathRenameExtension, PathRenameExtensionA, PathRenameExtensionW)

#endif // PATHCCH\_NO\_DEPRECATE

#else // \_MSC\_VER

#endif /\* \_INC\_BANNED \*/

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonMinMemory.h"

typedef \_Return\_type\_success\_(return >= 0) LONG NTSTATUS;

#define NT\_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)

#include <wchar.h>

// === C Runtime Header Files ===

#include <stdarg.h>

#include <float.h>

#include <limits.h>

#if defined(\_UCRT)

#include <cmath>

#else

#include <math.h>

#endif

#include <time.h>

#include <io.h>

#include <malloc.h>

extern "C" void \* \_AddressOfReturnAddress(void);

#include "Common\GetCurrentFrameID.h"

namespace Js

{

typedef int32 PropertyId;

typedef unsigned long ModuleID;

}

#define IsTrueOrFalse(value) ((value) ? L"True" : L"False")

// Header files

#include "core\BinaryFeatureControl.h"

#include "TemplateParameter.h"

#include "Common\vtinfo.h"

#include "EnumClassHelp.h"

#include "Common\Tick.h"

#include "Common\Int16Math.h"

#include "Common\Int32Math.h"

#include "Common\UInt16Math.h"

#include "Common\UInt32Math.h"

#include "common\Int64Math.h"

template<typename T> struct IntMath { using Type = void; };

template<> struct IntMath<int16> { using Type = Int16Math; };

template<> struct IntMath<int32> { using Type = Int32Math; };

template<> struct IntMath<uint16> { using Type = UInt16Math; };

template<> struct IntMath<uint32> { using Type = UInt32Math; };

template<> struct IntMath<int64> { using Type = Int64Math; };

#include "Common\DaylightTimeHelper.h"

#include "Common\DateUtilities.h"

#include "Common\NumberUtilitiesBase.h"

#include "Common\NumberUtilities.h"

#include <codex\Utf8Codex.h>

#include "Common\unicode.h"

#include "core\DelayLoadLibrary.h"

#include "core\EtwTraceCore.h"

#include "Common\RejitReason.h"

#include "Common\ThreadService.h"

// Exceptions

#include "Exceptions\Exceptionbase.h"

#include "Exceptions\InternalErrorException.h"

#include "Exceptions\OutOfMemoryException.h"

#include "Exceptions\OperationAbortedException.h"

#include "Exceptions\RejitException.h"

#include "Exceptions\ScriptAbortException.h"

#include "Exceptions\StackOverflowException.h"

#include "Exceptions\NotImplementedException.h"

#include "Exceptions\AsmJsParseException.h"

#include "Memory\AutoPtr.h"

#include "Memory\AutoAllocatorObjectPtr.h"

#include "Memory\leakreport.h"

#include "DataStructures\DoublyLinkedListElement.h"

#include "DataStructures\DoublyLinkedList.h"

#include "DataStructures\SimpleHashTable.h"

#include "Memory\XDataAllocator.h"

#include "Memory\CustomHeap.h"

#include "Core\FinalizableObject.h"

#include "Memory\RecyclerRootPtr.h"

#include "Memory\RecyclerFastAllocator.h"

#include "Memory\RecyclerPointers.h"

#include "util\pinned.h"

// Data Structures 2

#include "DataStructures\StringBuilder.h"

#include "DataStructures\WeakReferenceDictionary.h"

#include "DataStructures\LeafValueDictionary.h"

#include "DataStructures\Dictionary.h"

#include "DataStructures\List.h"

#include "DataStructures\Stack.h"

#include "DataStructures\Queue.h"

#include "DataStructures\CharacterBuffer.h"

#include "DataStructures\InternalString.h"

#include "DataStructures\Interval.h"

#include "DataStructures\InternalStringNoCaseComparer.h"

#include "DataStructures\SparseArray.h"

#include "DataStructures\growingArray.h"

#include "DataStructures\EvalMapString.h"

#include "DataStructures\RegexKey.h"

#include "DataStructures\LineOffsetCache.h"

#include "core\ICustomConfigFlags.h"

#include "core\CmdParser.h"

#include "core\ProfileInstrument.h"

#include "core\ProfileMemory.h"

#include "core\StackBackTrace.h"

#include "Common\Event.h"

#include "Common\Jobs.h"

#include "common\vtregistry.h" // Depends on SimpleHashTable.h

#include "DataStructures\Cache.h" // Depends on config flags

#include "DataStructures\MruDictionary.h" // Depends on DoublyLinkedListElement

#include "common\SmartFPUControl.h"

// This class is only used by AutoExp.dat

class AutoExpDummyClass

{

};

#pragma warning(push)

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(HEAP\_TRACK\_ALLOC) || defined(ENABLE\_DEBUG\_CONFIG\_OPTIONS)

#include <typeinfo.h>

#endif

#pragma warning(pop)

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "Banned.h"

#include "CommonDefines.h"

#define \_CRT\_RAND\_S // Enable rand\_s in the CRT

#pragma warning(push)

#pragma warning(disable: 4995) /\* 'function': name was marked as #pragma deprecated \*/

// === Windows Header Files ===

#define INC\_OLE2 /\* for windows.h \*/

#define CONST\_VTABLE /\* for objbase.h \*/

#include <windows.h>

/\* Don't want GetObject and GetClassName to be defined to GetObjectW and GetClassNameW \*/

#undef GetObject

#undef GetClassName

#undef Yield /\* winbase.h defines this but we want to use it for Js::OpCode::Yield; it is Win16 legacy, no harm undef'ing it \*/

#pragma warning(pop)

// === Core Header Files ===

#include "Core\api.h"

#include "Core\CommonTypedefs.h"

#include "core\CriticalSection.h"

#include "core\Assertions.h"

// === Exceptions Header Files ===

#include "Exceptions\Throw.h"

#include "Exceptions\ExceptionCheck.h"

#include "Exceptions\reporterror.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "TargetVer.h"

#include "Warnings.h"

//----------------------------------------------------------------------------------------------------

// Chakra Core version

//----------------------------------------------------------------------------------------------------

#define CHAKRA\_CORE\_MAJOR\_VERSION 1

#define CHAKRA\_CORE\_MINOR\_VERSION 1

#define CHAKRA\_CORE\_VERSION\_RELEASE 0

#define CHAKRA\_CORE\_VERSION\_PRERELEASE 1

#define CHAKRA\_CORE\_VERSION\_RELEASE\_QFE 0

#define CHAKRA\_VERSION\_RELEASE 0

#define CHAKRA\_VERSION\_PRERELEASE 1

// NOTE: need to update the GUID in ByteCodeCacheReleaseFileVersion.h as well

//----------------------------------------------------------------------------------------------------

// Default debug/fretest/release flags values

// - Set the default values of debug/fretest/release flags if it is not set by the command line

//----------------------------------------------------------------------------------------------------

#ifndef DBG\_DUMP

#define DBG\_DUMP 0

#endif

#ifdef \_DEBUG

#ifdef DEBUG

#undef DEBUG

#endif

#define DEBUG 1

#endif

// if test hook is enabled, debug config options are enabled too

#ifdef ENABLE\_TEST\_HOOKS

#ifndef ENABLE\_DEBUG\_CONFIG\_OPTIONS

#define ENABLE\_DEBUG\_CONFIG\_OPTIONS 1

#endif

#endif

// ENABLE\_DEBUG\_CONFIG\_OPTIONS is enabled in debug build when DBG or DBG\_DUMP is defined

// It is enabled in fretest build (jscript9test.dll and jc.exe) in the build script

#if DBG || DBG\_DUMP

#ifndef ENABLE\_DEBUG\_CONFIG\_OPTIONS

#define ENABLE\_DEBUG\_CONFIG\_OPTIONS 1

#endif

// Flag to control availability of other flags to control regex debugging, tracing, profiling, etc. This is separate from

// ENABLE\_DEBUG\_CONFIG\_OPTIONS because enabling this flag may affect performance significantly, even with default values for

// the regex flags this flag would make available.

#ifndef ENABLE\_REGEX\_CONFIG\_OPTIONS

#define ENABLE\_REGEX\_CONFIG\_OPTIONS 1

#endif

#endif

// TODO: consider removing before RTM: keep for CHK/FRETEST but remove from FRE.

// This will cause terminate process on AV/Assert rather that letting PDM (F12/debugger scenarios) eat exceptions.

// At least for now, enable this even in FRE builds. See ReportError.h.

#define ENABLE\_DEBUG\_API\_WRAPPER 1

//----------------------------------------------------------------------------------------------------

// Define Architectures' aliases for Simplicity

//----------------------------------------------------------------------------------------------------

#if defined(\_M\_ARM) || defined(\_M\_ARM64)

#define \_M\_ARM32\_OR\_ARM64 1

#endif

#if defined(\_M\_IX86) || defined(\_M\_ARM)

#define \_M\_IX86\_OR\_ARM32 1

#define TARGET\_32 1

#endif

#if defined(\_M\_X64) || defined(\_M\_ARM64)

#define \_M\_X64\_OR\_ARM64 1

#define TARGET\_64 1

#endif

//----------------------------------------------------------------------------------------------------

// Enabled features

//----------------------------------------------------------------------------------------------------

// NOTE: Disabling these might not work and are not fully supported and maintained

// Even if it builds, it may not work properly. Disable at your own risk

// ByteCode

#define VARIABLE\_INT\_ENCODING 1 // Byte code serialization variable size int field encoding

#define BYTECODE\_BRANCH\_ISLAND // Byte code short branch and branch island

// Language features

#define ENABLE\_INTL\_OBJECT // Intl support

#define ENABLE\_ES6\_CHAR\_CLASSIFIER // ES6 Unicode character classifier support

// Type system features

#define PERSISTENT\_INLINE\_CACHES // \*\*\* TODO: Won't build if disabled currently

#define SUPPORT\_FIXED\_FIELDS\_ON\_PATH\_TYPES // \*\*\* TODO: Won't build if disabled currently

// GC features

#define CONCURRENT\_GC\_ENABLED 1 // \*\*\* TODO: Won't build if disabled currently

#define PARTIAL\_GC\_ENABLED 1 // \*\*\* TODO: Won't build if disabled currently

#define BUCKETIZE\_MEDIUM\_ALLOCATIONS 1 // \*\*\* TODO: Won't build if disabled currently

#define SMALLBLOCK\_MEDIUM\_ALLOC 1 // \*\*\* TODO: Won't build if disabled currently

#define LARGEHEAPBLOCK\_ENCODING 1 // Large heap block metadata encoding

#define RECYCLER\_WRITE\_BARRIER // Write Barrier support

#define IDLE\_DECOMMIT\_ENABLED 1 // Idle Decommit

#define RECYCLER\_PAGE\_HEAP // PageHeap support

// JIT features

#if DISABLE\_JIT

#define ENABLE\_NATIVE\_CODEGEN 0

#define ENABLE\_PROFILE\_INFO 0

#define ENABLE\_BACKGROUND\_PARSING 0 // Disable background parsing in this mode

// We need to decouple the Jobs infrastructure out of

// Backend to make background parsing work with JIT disabled

#define DYNAMIC\_INTERPRETER\_THUNK 0

#define DISABLE\_DYNAMIC\_PROFILE\_DEFER\_PARSE

#define ENABLE\_COPYONACCESS\_ARRAY 0

// Used to temporarily disable ASMjs related code to get nonative compiling

#define TEMP\_DISABLE\_ASMJS

#else

// By default, enable the JIT

#define ENABLE\_NATIVE\_CODEGEN 1

#define ENABLE\_PROFILE\_INFO 1

#define ENABLE\_BACKGROUND\_PARSING 1

#define ENABLE\_COPYONACCESS\_ARRAY 1

#ifndef DYNAMIC\_INTERPRETER\_THUNK

#if defined(\_M\_IX86\_OR\_ARM32) || defined(\_M\_X64\_OR\_ARM64)

#define DYNAMIC\_INTERPRETER\_THUNK 1

#else

#define DYNAMIC\_INTERPRETER\_THUNK 0

#endif

#endif

#endif

// Other features

// #define CHAKRA\_CORE\_DOWN\_COMPAT 1

#if defined(ENABLE\_DEBUG\_CONFIG\_OPTIONS) || defined(CHAKRA\_CORE\_DOWN\_COMPAT)

#define DELAYLOAD\_SET\_CFG\_TARGET 1

#endif

#ifdef NTBUILD

#define ENABLE\_PROJECTION

#define ENABLE\_FOUNDATION\_OBJECT

#define ENABLE\_EXPERIMENTAL\_FLAGS

#define ENABLE\_WININET\_PROFILE\_DATA\_CACHE

#define ENABLE\_BASIC\_TELEMETRY

#define ENABLE\_DOM\_FAST\_PATH

#define ENABLE\_JS\_ETW // ETW support

#define EDIT\_AND\_CONTINUE

#endif

// Telemetry flags

#ifdef ENABLE\_BASIC\_TELEMETRY

#define ENABLE\_DIRECTCALL\_TELEMETRY

#endif

// Telemetry features (non-DEBUG related)

#ifdef ENABLE\_BASIC\_TELEMETRY

// These defines can be "overridden" in other headers (e.g. ESBuiltInsTelemetryProvider.h) in case a specific telemetry provider wants to change an option for performance.

#define TELEMETRY\_OPCODE\_OFFSET\_ENABLED true // If the BytecodeOffset and FunctionId are logged.

#define TELEMETRY\_PROPERTY\_OPCODE\_FILTER(propertyId) true // Any filter to apply on a per propertyId basis in the opcode handler for GetProperty/TypeofProperty/GetMethodProperty/etc.

#define TELEMETRY\_OPCODE\_GET\_PROPERTY\_VALUES true // If no telemetry providers need the values of properties then this option skips getting the value in the TypeofProperty opcode handler.

// #define TELEMETRY\_PROFILED // If telemetry should capture "Profiled\*" operations

// #define TELEMETRY\_CACHEHIT // If telemetry should capture data that was gotten with a Cache Hit

// #define TELEMETRY\_JSO // If telemetry should capture JavascriptOperators (expensive, as it happens during JITed code too, not just interpreted mode)

#define TELEMETRY\_AddToCache // If telemetry should capture property-gets only when the propertyId is added to the cache (generally this means only the first usage of any feature is logged)

// #define TELEMETRY\_INTERPRETER // If telemetry should capture more interpreter events compared to just TELEMETRY\_AddToCache

#define TELEMETRY\_TRACELOGGING // Telemetry output using TraceLogging

// #define TELEMETRY\_OUTPUTPRINT // Telemetry output using Output::Print

// Enable/disable specific telemetry providers:

#define TELEMETRY\_ESB // Telemetry of ECMAScript Built-Ins usage or detection.

// #define TELEMETRY\_ARRAY\_USAGE // Telemetry of Array usage statistics

#define TELEMETRY\_DateParse // Telemetry of `Date.parse`

#ifdef TELEMETRY\_ESB

// Because ESB telemetry is in-production and has major performance implications, this redefines some of the #defines above to disable non-critical functionality to get more performance.

#undef TELEMETRY\_OPCODE\_OFFSET\_ENABLED // Disable the FunctionId+Offset tracker.

#define TELEMETRY\_OPCODE\_OFFSET\_ENABLED false

#undef TELEMETRY\_PROPERTY\_OPCODE\_FILTER // Redefine the Property Opcode filter to ignore non-built-in properties.

#define TELEMETRY\_PROPERTY\_OPCODE\_FILTER(propertyId) (propertyId < Js::PropertyIds::\_countJSOnlyProperty)

#undef TELEMETRY\_OPCODE\_GET\_PROPERTY\_VALUES

#define TELEMETRY\_OPCODE\_GET\_PROPERTY\_VALUES false

//#define TELEMETRY\_ESB\_STRINGS // Telemetry that uses strings (slow), used for constructor detection for ECMAScript Built-Ins polyfills and Constructor-properties of Chakra-built-ins.

//#define TELEMETRY\_ESB\_GetConstructorPropertyPolyfillDetection // Whether telemetry will inspect the `.constructor` property of every Object instance to determine if it's a polyfill of a known ES built-in.

#endif

#else

#define TELEMETRY\_OPCODE\_OFFSET\_ENABLED false

#define TELEMETRY\_OPCODE\_FILTER(propertyId) false

#endif

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

#define ENABLE\_DIRECTCALL\_TELEMETRY\_STATS

#endif

//----------------------------------------------------------------------------------------------------

// Debug and fretest features

//----------------------------------------------------------------------------------------------------

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

#define BAILOUT\_INJECTION

#if ENABLE\_PROFILE\_INFO

#define DYNAMIC\_PROFILE\_STORAGE

#define DYNAMIC\_PROFILE\_MUTATOR

#endif

#define RUNTIME\_DATA\_COLLECTION

#define SECURITY\_TESTING

#define PROFILE\_EXEC

#define BGJIT\_STATS

#define REJIT\_STATS

#define PERF\_HINT

#define POLY\_INLINE\_CACHE\_SIZE\_STATS

#define JS\_PROFILE\_DATA\_INTERFACE 1

#define EXCEPTION\_RECOVERY 1

#define RECYCLER\_TEST\_SUPPORT

#define ARENA\_ALLOCATOR\_FREE\_LIST\_SIZE

// TODO (t-doilij) combine IR\_VIEWER and ENABLE\_IR\_VIEWER

#ifdef \_M\_IX86

#if ENABLE\_NATIVE\_CODEGEN

#define IR\_VIEWER

#define ENABLE\_IR\_VIEWER

#define ENABLE\_IR\_VIEWER\_DBG\_DUMP // TODO (t-doilij) disable this before check-in

#endif

#endif

#ifdef ENABLE\_JS\_ETW

#define TEST\_ETW\_EVENTS

// VTUNE profiling requires ETW trace

#if defined(\_M\_IX86) || defined(\_M\_X64)

#define VTUNE\_PROFILING

#endif

#endif

#ifdef NTBUILD

#define PERF\_COUNTERS

#define ENABLE\_MUTATION\_BREAKPOINT

#endif

#ifdef \_CONTROL\_FLOW\_GUARD

#define CONTROL\_FLOW\_GUARD\_LOGGER

#endif

#ifndef ENABLE\_TEST\_HOOKS

#define ENABLE\_TEST\_HOOKS

#endif

#endif // ENABLE\_DEBUG\_CONFIG\_OPTIONS

//----------------------------------------------------------------------------------------------------

// Debug only features

//----------------------------------------------------------------------------------------------------

#ifdef DEBUG

#define BYTECODE\_TESTING

#define FAULT\_INJECTION

#define RECYCLER\_NO\_PAGE\_REUSE

#ifdef NTBUILD

#define INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

#define INTERNAL\_MEM\_PROTECT\_HEAP\_CMDLINE

#endif

#endif

#ifdef DBG

#define VALIDATE\_ARRAY

#define GENERATE\_DUMP

#endif

#if DBG\_DUMP

#undef DBG\_EXTRAFIELD // make sure we don't extra fields in free build.

#define TRACK\_DISPATCH

#define BGJIT\_STATS

#define REJIT\_STATS

#define POLY\_INLINE\_CACHE\_SIZE\_STATS

#define INLINE\_CACHE\_STATS

#define FIELD\_ACCESS\_STATS

#define MISSING\_PROPERTY\_STATS

#define EXCEPTION\_RECOVERY 1

#define EXCEPTION\_CHECK // Check exception handling.

#define PROFILE\_EXEC

#define PROFILE\_MEM

#define PROFILE\_TYPES

#define PROFILE\_EVALMAP

#define PROFILE\_OBJECT\_LITERALS

#define PROFILE\_BAILOUT\_RECORD\_MEMORY

#define MEMSPECT\_TRACKING

#define PROFILE\_RECYCLER\_ALLOC

#define PROFILE\_STRINGS

#define PROFILE\_DICTIONARY 1

#define RECYCLER\_SLOW\_CHECK\_ENABLED // This can be disabled to speed up the debug build's GC

#define RECYCLER\_STRESS

#define RECYCLER\_STATS

#define RECYCLER\_FINALIZE\_CHECK

#define RECYCLER\_FREE\_MEM\_FILL

#define RECYCLER\_DUMP\_OBJECT\_GRAPH

#define RECYCLER\_MEMORY\_VERIFY

#define RECYCLER\_ZERO\_MEM\_CHECK

#define RECYCLER\_TRACE

#define RECYCLER\_VERIFY\_MARK

#ifdef PERF\_COUNTERS

#define RECYCLER\_PERF\_COUNTERS

#define HEAP\_PERF\_COUNTERS

#endif // PERF\_COUNTERS

#define PAGEALLOCATOR\_PROTECT\_FREEPAGE

#define ARENA\_MEMORY\_VERIFY

#define SEPARATE\_ARENA

#define HEAP\_TRACK\_ALLOC

#define CHECK\_MEMORY\_LEAK

#define LEAK\_REPORT

#define PROJECTION\_METADATA\_TRACE

#define ERROR\_TRACE

#define DEBUGGER\_TRACE

#define PROPERTY\_RECORD\_TRACE

#define ARENA\_ALLOCATOR\_FREE\_LIST\_SIZE

#ifdef DBG\_EXTRAFIELD

#define HEAP\_ENUMERATION\_VALIDATION

#endif

#endif // DBG\_DUMP

//----------------------------------------------------------------------------------------------------

// Special build features

// - features that can be enabled on private builds for debugging

//----------------------------------------------------------------------------------------------------

#ifdef ENABLE\_JS\_ETW

// #define ETW\_MEMORY\_TRACKING // ETW events for internal allocations

#endif

// #define OLD\_ITRACKER // Switch to the old IE8 ITracker GUID

// #define LOG\_BYTECODE\_AST\_RATIO // log the ratio between AST size and bytecode generated.

// #define DUMP\_FRAGMENTATION\_STATS // Display HeapBucket fragmentation stats after sweep

// ----- Fretest or free build special build features (already enabled in debug builds) -----

// #define TRACK\_DISPATCH

// #define BGJIT\_STATS

// Profile defines that can be enabled in release build

// #define PROFILE\_EXEC

// #define PROFILE\_MEM

// #define PROFILE\_STRINGS

// #define PROFILE\_TYPES

// #define PROFILE\_OBJECT\_LITERALS

// #define PROFILE\_RECYCLER\_ALLOC

// #define MEMSPECT\_TRACKING

// #define HEAP\_TRACK\_ALLOC

// Recycler defines that can be enabled in release build

// #define RECYCLER\_STRESS

// #define RECYCLER\_STATS

// #define RECYCLER\_FINALIZE\_CHECK

// #define RECYCLER\_FREE\_MEM\_FILL

// #define RECYCLER\_DUMP\_OBJECT\_GRAPH

// #define RECYCLER\_MEMORY\_VERIFY

// #define RECYCLER\_TRACE

// #define RECYCLER\_VERIFY\_MARK

// #ifdef PERF\_COUNTERS

// #define RECYCLER\_PERF\_COUNTERS

// #define HEAP\_PERF\_COUNTERS

// #endif //PERF\_COUNTERS

// Other defines that can be enabled in release build

// #define PAGEALLOCATOR\_PROTECT\_FREEPAGE

// #define ARENA\_MEMORY\_VERIFY

// #define SEPARATE\_ARENA

// #define LEAK\_REPORT

// #define CHECK\_MEMORY\_LEAK

// #define RECYCLER\_MARK\_TRACK

// #define INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

//----------------------------------------------------------------------------------------------------

// Disabled features

//----------------------------------------------------------------------------------------------------

//Enable/disable dom properties

#define DOMEnabled 0

//----------------------------------------------------------------------------------------------------

// Platform dependent flags

//----------------------------------------------------------------------------------------------------

#ifndef INT32VAR

#if defined(\_M\_X64\_OR\_ARM64)

#define INT32VAR 1

#else

#define INT32VAR 0

#endif

#endif

#ifndef FLOATVAR

#if defined(\_M\_X64)

#define FLOATVAR 1

#else

#define FLOATVAR 0

#endif

#endif

#if defined(\_M\_IX86) || defined(\_M\_X64)

#ifndef TEMP\_DISABLE\_ASMJS

#define ASMJS\_PLAT

#endif

#endif

#if \_WIN32 || \_WIN64

#if \_M\_IX86

#define I386\_ASM 1

#endif //\_M\_IX86

#endif // \_WIN32 || \_WIN64

#ifndef PDATA\_ENABLED

#if defined(\_M\_ARM32\_OR\_ARM64) || defined(\_M\_X64)

#define PDATA\_ENABLED 1

#else

#define PDATA\_ENABLED 0

#endif

#endif

//----------------------------------------------------------------------------------------------------

// Dependent flags

// - flags values that are dependent on other flags

//----------------------------------------------------------------------------------------------------

#ifndef CONCURRENT\_GC\_ENABLED

#undef IDLE\_DECOMMIT\_ENABLED // Currently idle decommit can only be enabled if concurrent gc is enabled

#endif

#ifdef BAILOUT\_INJECTION

#define ENABLE\_PREJIT

#endif

#if defined(ENABLE\_DEBUG\_CONFIG\_OPTIONS)

// Enable Output::Trace

#define ENABLE\_TRACE

#endif

#if DBG || defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT) || defined(TRACK\_DISPATCH) || defined(ENABLE\_TRACE) || defined(RECYCLER\_PAGE\_HEAP)

#define STACK\_BACK\_TRACE

#endif

#if defined(STACK\_BACK\_TRACE) || defined(CONTROL\_FLOW\_GUARD\_LOGGER)

#define DBGHELP\_SYMBOL\_MANAGER

#endif

#if defined(TRACK\_DISPATCH) || defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

#define TRACK\_JS\_DISPATCH

#endif

// LEAK\_REPORT and CHECK\_MEMORY\_LEAK requires RECYCLER\_DUMP\_OBJECT\_GRAPH

// HEAP\_TRACK\_ALLOC and RECYCLER\_STATS

#if defined(LEAK\_REPORT) || defined(CHECK\_MEMORY\_LEAK)

#define RECYCLER\_DUMP\_OBJECT\_GRAPH

#define HEAP\_TRACK\_ALLOC

#define RECYCLER\_STATS

#endif

// PROFILE\_RECYCLER\_ALLOC requires PROFILE\_MEM

#if defined(PROFILE\_RECYCLER\_ALLOC) && !defined(PROFILE\_MEM)

#define PROFILE\_MEM

#endif

// RECYCLER\_DUMP\_OBJECT\_GRAPH is needed when using PROFILE\_RECYCLER\_ALLOC

#if defined(PROFILE\_RECYCLER\_ALLOC) && !defined(RECYCLER\_DUMP\_OBJECT\_GRAPH)

#define RECYCLER\_DUMP\_OBJECT\_GRAPH

#endif

#if defined(HEAP\_TRACK\_ALLOC) || defined(PROFILE\_RECYCLER\_ALLOC)

#define TRACK\_ALLOC

#define TRACE\_OBJECT\_LIFETIME // track a particular object's lifetime

#endif

#if defined(USED\_IN\_STATIC\_LIB)

#undef FAULT\_INJECTION

#undef RECYCLER\_DUMP\_OBJECT\_GRAPH

#undef HEAP\_TRACK\_ALLOC

#undef RECYCLER\_STATS

#undef PERF\_COUNTERS

#endif

// Not having the config options enabled trumps all the above logic for these switches

#ifndef ENABLE\_DEBUG\_CONFIG\_OPTIONS

#undef ARENA\_MEMORY\_VERIFY

#undef RECYCLER\_MEMORY\_VERIFY

#undef PROFILE\_MEM

#undef PROFILE\_DICTIONARY

#undef PROFILE\_RECYCLER\_ALLOC

#undef PROFILE\_EXEC

#undef PROFILE\_EVALMAP

#undef FAULT\_INJECTION

#undef RECYCLER\_STRESS

#undef RECYCLER\_SLOW\_VERIFY

#undef RECYCLER\_VERIFY\_MARK

#undef RECYCLER\_STATS

#undef RECYCLER\_FINALIZE\_CHECK

#undef RECYCLER\_DUMP\_OBJECT\_GRAPH

#undef DBG\_DUMP

#undef BGJIT\_STATS

#undef EXCEPTION\_RECOVERY

#undef PROFILE\_STRINGS

#undef PROFILE\_TYPES

#undef PROFILE\_OBJECT\_LITERALS

#undef SECURITY\_TESTING

#undef LEAK\_REPORT

#endif

//----------------------------------------------------------------------------------------------------

// Default flags values

// - Set the default values of flags if it is not set by the command line or above

//----------------------------------------------------------------------------------------------------

#ifndef JS\_PROFILE\_DATA\_INTERFACE

#define JS\_PROFILE\_DATA\_INTERFACE 0

#endif

#ifndef PROFILE\_DICTIONARY

#define PROFILE\_DICTIONARY 0

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// INL files

#include "Common\vtinfo.inl"

#include "DataStructures\DoublyLinkedListElement.inl"

#include "DataStructures\DoublyLinkedList.inl"

#include "Memory\Recycler.inl"

#include "Memory\MarkContext.inl"

#include "Memory\HeapBucket.inl"

#include "Memory\LargeHeapBucket.inl"

#include "Memory\HeapBlock.inl"

#include "Memory\HeapBlockMap.inl"

#include "Common\Jobs.inl"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonBasic.h"

// === C Runtime Header Files ===

#pragma warning(push)

#pragma warning(disable: 4995) /\* 'function': name was marked as #pragma deprecated \*/

#include <stdio.h>

#pragma warning(pop)

#include <intrin.h>

// === Core Header Files ===

#include "Core\CommonMinMax.h"

#include "EnumHelp.h"

#include "core\Assertions.h"

#include "core\SysInfo.h"

#include "core\PerfCounter.h"

#include "core\PerfCounterSet.h"

#include "Common\MathUtil.h"

#include "Core\AllocSizeMath.h"

#include "core\FaultInjection.h"

#include "core\BasePtr.h"

#include "core\AutoFILE.h"

#include "core\Output.h"

// === Basic Memory Header Files ===

namespace Memory {}

using namespace Memory;

#include "Memory\Allocator.h"

#include "Memory\HeapAllocator.h"

// === Data structures Header Files ===

#include "DataStructures\Comparer.h"

#include "DataStructures\SizePolicy.h"

#include "DataStructures\BitVector.h"

#include "DataStructures\SList.h"

#include "DataStructures\DList.h"

#include "DataStructures\KeyValuePair.h"

#include "DataStructures\BaseDictionary.h"

#include "DataStructures\DictionaryEntry.h"

// === Configurations Header ===

#include "core\ConfigFlagsTable.h"

// === Page/Arena Memory Header Files ===

#include "Memory\VirtualAllocWrapper.h"

#include "Memory\MemoryTracking.h"

#include "Memory\AllocationPolicyManager.h"

#include "Memory\PageAllocator.h"

#include "Memory\ArenaAllocator.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonMin.h"

// === Recycler Memory Header Files ===

class FinalizableObject;

namespace Js

{

class DefaultListLockPolicy;

}

#ifdef ENABLE\_BASIC\_TELEMETRY

#include "GCTelemetry.h"

#endif

#include "Memory\IdleDecommitPageAllocator.h"

#include "Memory\RecyclerPageAllocator.h"

#include "Memory\FreeObject.h"

#include "Memory\PagePool.h"

#include "DataStructures\SimpleHashTable.h"

#include "DataStructures\PageStack.h"

#include "DataStructures\ContinuousPageStack.h"

#include "Memory\RecyclerWriteBarrierManager.h"

#include "Memory\HeapConstants.h"

#include "Memory\HeapBlock.h"

#include "Memory\SmallHeapBlockAllocator.h"

#include "Memory\SmallNormalHeapBlock.h"

#include "Memory\SmallLeafHeapBlock.h"

#include "Memory\SmallFinalizableHeapBlock.h"

#include "Memory\LargeHeapBlock.h"

#include "Memory\HeapBucket.h"

#include "Memory\SmallLeafHeapBucket.h"

#include "Memory\SmallNormalHeapBucket.h"

#include "Memory\SmallFinalizableHeapBucket.h"

#include "Memory\LargeHeapBucket.h"

#include "Memory\HeapInfo.h"

#include "Memory\HeapBlockMap.h"

#include "Memory\RecyclerObjectDumper.h"

#include "Memory\RecyclerWeakReference.h"

#include "Memory\RecyclerSweep.h"

#include "Memory\RecyclerHeuristic.h"

#include "Memory\MarkContext.h"

#include "Memory\RecyclerWatsonTelemtry.h"

#include "Memory\Recycler.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#ifdef PHASE

PHASE(All)

PHASE(BGJit)

PHASE(LibInit)

PHASE(JsLibInit)

PHASE(Parse)

PHASE(RegexCompile)

PHASE(DeferParse)

PHASE(ParserBind)

PHASE(FunctionSourceInfoParse)

PHASE(StringTemplateParse)

PHASE(SkipNestedDeferred)

PHASE(CacheScopeInfoNames)

PHASE(ScanAhead)

PHASE(ParallelParse)

PHASE(EarlyReferenceErrors)

PHASE(ByteCode)

PHASE(CachedScope)

PHASE(StackFunc)

PHASE(StackClosure)

PHASE(DelayCapture)

PHASE(DebuggerScope)

PHASE(ByteCodeSerialization)

PHASE(VariableIntEncoding)

PHASE(NativeCodeSerialization)

PHASE(Delay)

PHASE(Speculation)

PHASE(GatherCodeGenData)

PHASE(Asmjs)

PHASE(AsmjsTmpRegisterAllocation)

PHASE(AsmjsEncoder)

PHASE(AsmjsInterpreter)

PHASE(AsmJsJITTemplate)

PHASE(AsmjsFunctionEntry)

PHASE(AsmjsInterpreterStack)

PHASE(AsmjsEntryPointInfo)

PHASE(BackEnd)

PHASE(IRBuilder)

PHASE(SwitchOpt)

PHASE(BailOnNoProfile)

PHASE(BackendConcatExprOpt)

PHASE(ClosureRangeCheck)

PHASE(ClosureRegCheck)

PHASE(Inline)

PHASE(InlineRecursive)

PHASE(InlineAtEveryCaller) //Inlines a function, say, foo at every caller of foo. Doesn't guarantee all the calls within foo are inlined too.

PHASE(InlineTree) //Inlines every function within a top function, say, foo (which needs to be top function) Note: -force:inline achieves the effect of both -force:InlineTree & -force:InlineAtEveryCaller

PHASE(TryAggressiveInlining)

PHASE(InlineConstructors)

PHASE(InlineBuiltIn)

PHASE(InlineInJitLoopBody)

PHASE(InlineAccessors)

PHASE(InlineGetters)

PHASE(InlineSetters)

PHASE(InlineApply)

PHASE(InlineApplyTarget)

PHASE(BailOutOnNotStackArgs)

PHASE(InlineCall)

PHASE(InlineCallTarget)

PHASE(PartialPolymorphicInline)

PHASE(PolymorphicInline)

PHASE(PolymorphicInlineFixedMethods)

PHASE(InlineOutsideLoops)

PHASE(InlineFunctionsWithLoops)

PHASE(EliminateArgoutForInlinee)

PHASE(InlineBuiltInCaller)

PHASE(InlineArgsOpt)

PHASE(RemoveInlineFrame)

PHASE(InlinerConstFold)

PHASE(ExecBOIFastPath)

PHASE(FGBuild)

PHASE(RemoveBreakBlock)

PHASE(TailDup)

PHASE(FGPeeps)

PHASE(GlobOpt)

PHASE(OptimizeTryCatch)

PHASE(Backward)

PHASE(TrackIntUsage)

PHASE(TrackNegativeZero)

PHASE(TypedArrayVirtual)

PHASE(TrackIntOverflow)

PHASE(TrackCompoundedIntOverflow)

PHASE(Forward)

PHASE(ValueTable)

PHASE(PathDependentValues)

PHASE(TrackRelativeIntBounds)

PHASE(BoundCheckElimination)

PHASE(BoundCheckHoist)

PHASE(LoopCountBasedBoundCheckHoist)

PHASE(CopyProp)

PHASE(ObjPtrCopyProp)

PHASE(ConstProp)

PHASE(ConstFold)

PHASE(CSE)

PHASE(HoistConstInt)

PHASE(TypeSpec)

PHASE(AggressiveIntTypeSpec)

PHASE(AggressiveMulIntTypeSpec)

PHASE(LossyIntTypeSpec)

PHASE(FloatTypeSpec)

PHASE(StringTypeSpec)

PHASE(InductionVars)

PHASE(Invariants)

PHASE(FieldCopyProp)

PHASE(FieldHoist)

PHASE(FieldPRE)

PHASE(HostOpt)

PHASE(ObjTypeSpec)

PHASE(ObjTypeSpecNewObj)

PHASE(ObjTypeSpecIsolatedFldOps)

PHASE(ObjTypeSpecIsolatedFldOpsWithBailOut)

PHASE(ObjTypeSpecStore)

PHASE(EquivObjTypeSpec)

PHASE(EquivObjTypeSpecByDefault)

PHASE(TraceObjTypeSpecTypeGuards)

PHASE(TraceObjTypeSpecWriteGuards)

PHASE(LiveOutFields)

PHASE(DisabledObjTypeSpec)

#if DBG

PHASE(SimulatePolyCacheWithOneTypeForFunction)

#endif

PHASE(CheckThis)

PHASE(StackArgOpt)

PHASE(IndirCopyProp)

PHASE(ArrayCheckHoist)

PHASE(ArrayMissingValueCheckHoist)

PHASE(ArraySegmentHoist)

PHASE(JsArraySegmentHoist)

PHASE(ArrayLengthHoist)

PHASE(EliminateArrayAccessHelperCall)

PHASE(NativeArray)

PHASE(NativeNewScArray)

PHASE(NativeArrayConversion)

PHASE(CopyOnAccessArray)

PHASE(NativeArrayLeafSegment)

PHASE(TypedArrayTypeSpec)

PHASE(LdLenIntSpec)

PHASE(FixDataProps)

PHASE(FixMethodProps)

PHASE(FixAccessorProps)

PHASE(FixDataVarProps)

PHASE(UseFixedDataProps)

PHASE(UseFixedDataPropsInInliner)

PHASE(LazyBailout)

PHASE(LazyFixedDataBailout)

PHASE(LazyFixedTypeBailout)

PHASE(FixedMethods)

PHASE(FEFixedMethods)

PHASE(FixedFieldGuardCheck)

PHASE(FixedNewObj)

PHASE(JitAllocNewObj)

PHASE(FixedCtorInlining)

PHASE(FixedCtorCalls)

PHASE(FixedScriptMethodInlining)

PHASE(FixedScriptMethodCalls)

PHASE(FixedBuiltInMethodInlining)

PHASE(FixedBuiltInMethodCalls)

PHASE(SplitNewScObject)

PHASE(OptTagChecks)

PHASE(MemOp)

PHASE(MemSet)

PHASE(MemCopy)

PHASE(DeadStore)

PHASE(ReverseCopyProp)

PHASE(MarkTemp)

PHASE(MarkTempNumber)

PHASE(MarkTempObject)

PHASE(MarkTempNumberOnTempObject)

PHASE(Lowerer)

PHASE(FastPath)

PHASE(LoopFastPath)

PHASE(LeafFastPath)

PHASE(MathFastPath)

PHASE(Atom)

PHASE(MulStrengthReduction)

PHASE(AgenPeeps)

PHASE(BranchFastPath)

PHASE(CallFastPath)

PHASE(BitopsFastPath)

PHASE(OtherFastPath)

PHASE(ObjectFastPath)

PHASE(ProfileBasedFldFastPath)

PHASE(AddFldFastPath)

PHASE(RootObjectFldFastPath)

PHASE(ArrayLiteralFastPath)

PHASE(ArrayCtorFastPath)

PHASE(NewScopeSlotFastPath)

PHASE(FrameDisplayFastPath)

PHASE(HoistMarkTempInit)

PHASE(HoistConstAddr)

PHASE(PreLowererPeeps)

PHASE(CFGInJit)

PHASE(TypedArray)

PHASE(TracePinnedTypes)

PHASE(InterruptProbe)

PHASE(EncodeConstants)

PHASE(RegAlloc)

PHASE(Liveness)

PHASE(LinearScan)

PHASE(OpHelperRegOpt)

PHASE(StackPack)

PHASE(SecondChance)

PHASE(RegionUseCount)

PHASE(RegHoistLoads)

PHASE(ClearRegLoopExit)

PHASE(Peeps)

PHASE(Layout)

PHASE(EHBailoutPatchUp)

PHASE(FinalLower)

PHASE(PrologEpilog)

PHASE(InsertNOPs)

PHASE(Encoder)

PHASE(Emitter)

#if defined(\_M\_IX86) || defined(\_M\_X64)

PHASE(BrShorten)

PHASE(LoopAlign)

#endif

#ifdef RECYCLER\_WRITE\_BARRIER

#if DBG\_DUMP

PHASE(SWB)

#endif

#endif

PHASE(Run)

PHASE(Interpreter)

PHASE(EvalCompile)

PHASE(FastIndirectEval)

PHASE(IdleDecommit)

PHASE(IdleCollect)

PHASE(MemoryAllocation)

#ifdef RECYCLER\_PAGE\_HEAP

PHASE(PageHeap)

#endif

PHASE(LargeMemoryAllocation)

PHASE(PageAllocatorAlloc)

PHASE(Recycler)

PHASE(ThreadCollect)

PHASE(ExplicitFree)

PHASE(ExpirableCollect)

PHASE(GarbageCollect)

PHASE(ConcurrentCollect)

PHASE(BackgroundResetMarks)

PHASE(BackgroundFindRoots)

PHASE(BackgroundRescan)

PHASE(BackgroundRepeatMark)

PHASE(BackgroundFinishMark)

PHASE(ConcurrentPartialCollect)

PHASE(ParallelMark)

PHASE(PartialCollect)

PHASE(ResetMarks)

PHASE(ResetWriteWatch)

PHASE(FindRoot)

PHASE(FindRootArena)

PHASE(FindImplicitRoot)

PHASE(FindRootExt)

PHASE(ScanStack)

PHASE(ConcurrentMark)

PHASE(ConcurrentWait)

PHASE(Rescan)

PHASE(Mark)

PHASE(Sweep)

PHASE(SweepWeak)

PHASE(SweepSmall)

PHASE(SweepLarge)

PHASE(SweepPartialReuse)

PHASE(ConcurrentSweep)

PHASE(Finalize)

PHASE(Dispose)

PHASE(FinishPartial)

PHASE(Host)

PHASE(BailOut)

PHASE(RegexQc)

PHASE(InlineCache)

PHASE(PolymorphicInlineCache)

PHASE(MissingPropertyCache)

PHASE(CloneCacheInCollision)

PHASE(ConstructorCache)

PHASE(InlineCandidate)

PHASE(InlineHostCandidate)

PHASE(ScriptFunctionWithInlineCache)

PHASE(Arena)

PHASE(ApplyUsage)

PHASE(ObjectHeaderInlining)

PHASE(ObjectHeaderInliningForConstructors)

PHASE(ObjectHeaderInliningForObjectLiterals)

PHASE(ObjectHeaderInliningForEmptyObjects)

#if DBG\_DUMP

PHASE(TypePropertyCache)

PHASE(InlineSlots)

#endif

PHASE(DynamicProfile)

#ifdef DYNAMIC\_PROFILE\_STORAGE

PHASE(DynamicProfileStorage)

#endif

PHASE(JITLoopBody)

PHASE(JITLoopBodyInTryCatch)

PHASE(ReJIT)

PHASE(ExecutionMode)

PHASE(SimpleJitDynamicProfile)

PHASE(SimpleJit)

PHASE(FullJit)

PHASE(FailNativeCodeInstall)

PHASE(PixelArray)

PHASE(Etw)

PHASE(Profiler)

PHASE(CustomHeap)

PHASE(XDataAllocator)

PHASE(PageAllocator)

PHASE(StringConcat)

#if DBG\_DUMP

PHASE(PRNG)

#endif

PHASE(PreReservedHeapAlloc)

PHASE(CFG)

PHASE(ExceptionStackTrace)

PHASE(ExtendedExceptionInfoStackTrace)

PHASE(ProjectionMetadata)

PHASE(TypeHandlerTransition)

PHASE(Debugger)

PHASE(ENC)

PHASE(ConsoleScope)

PHASE(ScriptProfiler)

PHASE(JSON)

PHASE(RegexResultNotUsed)

PHASE(Error)

PHASE(PropertyRecord)

PHASE(TypePathDynamicSize)

PHASE(ConditionalCompilation)

PHASE(InterpreterProfile)

PHASE(InterpreterAutoProfile)

PHASE(ByteCodeConcatExprOpt)

PHASE(TraceInlineCacheInvalidation)

PHASE(TracePropertyGuards)

#ifdef ENABLE\_JS\_ETW

PHASE(StackFramesEvent)

#endif

PHASE(PerfHint)

PHASE(DeferSourceLoad)

PHASE(ObjectMutationBreakpoint)

#undef PHASE

#endif

#ifndef DEFAULT\_CONFIG\_BgJitDelay

#if \_M\_ARM

#define DEFAULT\_CONFIG\_BgJitDelay (70)

#else

#define DEFAULT\_CONFIG\_BgJitDelay (30)

#endif

#define DEFAULT\_CONFIG\_ASMJS (true)

#define DEFAULT\_CONFIG\_AsmJsEdge (false)

#define DEFAULT\_CONFIG\_AsmJsStopOnError (false)

#define DEFAULT\_CONFIG\_SIMDJS (false)

#define DEFAULT\_CONFIG\_BgJitDelayFgBuffer (0)

#define DEFAULT\_CONFIG\_BgJitPendingFuncCap (31)

#define DEFAULT\_CONFIG\_CurrentSourceInfo (true)

#define DEFAULT\_CONFIG\_CreateFunctionProxy (true)

#define DEFAULT\_CONFIG\_HybridFgJit (false)

#define DEFAULT\_CONFIG\_HybridFgJitBgQueueLengthThreshold (32)

#define DEFAULT\_CONFIG\_Prejit (false)

#define DEFAULT\_CONFIG\_DeferNested (true)

#define DEFAULT\_CONFIG\_DeferTopLevelTillFirstCall (true)

#define DEFAULT\_CONFIG\_DefineGetterSetter (true)

#define DEFAULT\_CONFIG\_DirectCallTelemetryStats (false)

#define DEFAULT\_CONFIG\_errorStackTrace (true)

#define DEFAULT\_CONFIG\_FastPathCap (-1) // By default, we do not have any fast path cap

#define DEFAULT\_CONFIG\_FastLineColumnCalculation (true)

#define DEFAULT\_CONFIG\_PrintLineColumnInfo (false)

#define DEFAULT\_CONFIG\_ForceDecommitOnCollect (false)

#define DEFAULT\_CONFIG\_ForceDeferParse (false)

#define DEFAULT\_CONFIG\_NoDeferParse (false)

#define DEFAULT\_CONFIG\_ForceDynamicProfile (false)

#define DEFAULT\_CONFIG\_ForceExpireOnNonCacheCollect (false)

#define DEFAULT\_CONFIG\_ForceFastPath (false)

#define DEFAULT\_CONFIG\_ForceJITLoopBody (false)

#define DEFAULT\_CONFIG\_ForceCleanPropertyOnCollect (false)

#define DEFAULT\_CONFIG\_ForceCleanCacheOnCollect (false)

#define DEFAULT\_CONFIG\_ForceGCAfterJSONParse (false)

#define DEFAULT\_CONFIG\_ForceSerialized (false)

#define DEFAULT\_CONFIG\_ForceES5Array (false)

#define DEFAULT\_CONFIG\_ForceAsmJsLinkFail (false)

#define DEFAULT\_CONFIG\_DumpCommentsFromReferencedFiles (false)

#define DEFAULT\_CONFIG\_ExtendedErrorStackForTestHost (false)

//Following determines inline thresholds

#define DEFAULT\_CONFIG\_InlineThreshold (35) //Default start

#define DEFAULT\_CONFIG\_AggressiveInlineThreshold (80) //Limit for aggressive inlining.

#define DEFAULT\_CONFIG\_InlineThresholdAdjustCountInLargeFunction (20)

#define DEFAULT\_CONFIG\_InlineThresholdAdjustCountInMediumSizedFunction (6)

#define DEFAULT\_CONFIG\_InlineThresholdAdjustCountInSmallFunction (10)

#define DEFAULT\_CONFIG\_ConstructorInlineThreshold (21) //Monomorphic constructor threshold

#define DEFAULT\_CONFIG\_ConstructorCallsRequiredToFinalizeCachedType (2)

#define DEFAULT\_CONFIG\_OutsideLoopInlineThreshold (16) //Threshold to inline outside loops

#define DEFAULT\_CONFIG\_LeafInlineThreshold (60) //Inlinee threshold for function which is leaf (irrespective of it has loops or not)

#define DEFAULT\_CONFIG\_LoopInlineThreshold (25) //Inlinee threshold for function with loops

#define DEFAULT\_CONFIG\_PolymorphicInlineThreshold (35) //Polymorphic inline threshold

#define DEFAULT\_CONFIG\_InlineCountMax (1200) //Max sum of bytecodes of inlinees inlined into a function (excluding built-ins)

#define DEFAULT\_CONFIG\_AggressiveInlineCountMax (8000) //Max sum of bytecodes of inlinees inlined into a function (excluding built-ins) when inlined aggressively

#define DEFAULT\_CONFIG\_MaxFuncInlineDepth (2) //Maximum number of times a function can be inlined within a top function

#define DEFAULT\_CONFIG\_MaxNumberOfInlineesWithLoop (40) //Inlinee with a loop is controlled by LoopInlineThreshold, though we don't want to inline lot of inlinees with loop, this ensures a limit.

#define DEFAULT\_CONFIG\_ConstantArgumentInlineThreshold (157) // Bytecode threshold for functions with constant arguments which are used for branching

#define DEFAULT\_CONFIG\_RecursiveInlineThreshold (2000) // Bytecode threshold recursive call at a call site

#define DEFAULT\_CONFIG\_RecursiveInlineDepthMax (8) // Maximum inline depth for recursive calls

#define DEFAULT\_CONFIG\_RecursiveInlineDepthMin (2) // Minimum inline depth for recursive calls

#define DEFAULT\_CONFIG\_CloneInlinedPolymorphicCaches (true)

#define DEFAULT\_CONFIG\_HighPrecisionDate (false)

#define DEFAULT\_CONFIG\_TDZ (true)

#define DEFAULT\_CONFIG\_ForceOldDateAPI (false)

#define DEFAULT\_CONFIG\_Loop (1)

#define DEFAULT\_CONFIG\_ForceDiagnosticsMode (false)

#define DEFAULT\_CONFIG\_EnableJitInDiagMode (true)

#define DEFAULT\_CONFIG\_EnableJitInHybridDebugging (true)

#define DEFAULT\_CONFIG\_UseFullName (true)

#define DEFAULT\_CONFIG\_EnableContinueAfterExceptionWrappersForHelpers (true)

#define DEFAULT\_CONFIG\_EnableContinueAfterExceptionWrappersForBuiltIns (true)

#define DEFAULT\_CONFIG\_EnableFunctionSourceReportForHeapEnum (true)

#define DEFAULT\_CONFIG\_LoopInterpretCount (150)

#define DEFAULT\_CONFIG\_LoopProfileIterations (25)

#define DEFAULT\_CONFIG\_JitLoopBodyHotLoopThreshold (20000)

#define DEFAULT\_CONFIG\_LoopBodySizeThresholdToDisableOpts (255)

#define DEFAULT\_CONFIG\_MaxJitThreadCount (2)

#define DEFAULT\_CONFIG\_ForceMaxJitThreadCount (false)

#ifdef RECYCLER\_PAGE\_HEAP

#define DEFAULT\_CONFIG\_PageHeap ((Js::Number) PageHeapMode::PageHeapModeOff)

#define DEFAULT\_CONFIG\_PageHeapAllocStack (false)

#define DEFAULT\_CONFIG\_PageHeapFreeStack (false)

#define DEFAULT\_CONFIG\_PageHeapBlockType ((Js::Number) PageHeapBlockTypeFilter::PageHeapBlockTypeFilterAll)

#endif

#define DEFAULT\_CONFIG\_LowMemoryCap (0xB900000) // 185 MB - based on memory cap for process on low-capacity device

#define DEFAULT\_CONFIG\_MaxCodeFill (500)

#define DEFAULT\_CONFIG\_MaxLoopsPerFunction (10)

#define DEFAULT\_CONFIG\_NopFrequency (8)

#define DEFAULT\_CONFIG\_SpeculationCap (1) // Needs to be 1 and not 0 since the compiler complains about a condition being always false

#define DEFAULT\_CONFIG\_ProfileBasedSpeculationCap (1600)

#define DEFAULT\_CONFIG\_Verbose (false)

#define DEFAULT\_CONFIG\_ForceStrictMode (false)

#define DEFAULT\_CONFIG\_EnableEvalMapCleanup (true)

#define DEFAULT\_CONFIG\_ExpirableCollectionGCCount (5) // Number of GCs during which entry point profiling occurs

#define DEFAULT\_CONFIG\_ExpirableCollectionTriggerThreshold (50) // Threshold at which Entry Point Collection is triggered

#define DEFAULT\_CONFIG\_RegexTracing (false)

#define DEFAULT\_CONFIG\_RegexProfile (false)

#define DEFAULT\_CONFIG\_RegexDebug (false)

#define DEFAULT\_CONFIG\_RegexOptimize (true)

#define DEFAULT\_CONFIG\_DynamicRegexMruListSize (16)

#define DEFAULT\_CONFIG\_GoptCleanupThreshold (25)

#define DEFAULT\_CONFIG\_AsmGoptCleanupThreshold (500)

#define DEFAULT\_CONFIG\_OptimizeForManyInstances (false)

#define DEFAULT\_CONFIG\_DeferParseThreshold (4 \* 1024) // Unit is number of characters

#define DEFAULT\_CONFIG\_ProfileBasedDeferParseThreshold (100) // Unit is number of characters

#define DEFAULT\_CONFIG\_ProfileBasedSpeculativeJit (true)

#define DEFAULT\_CONFIG\_WininetProfileCache (true)

#define DEFAULT\_CONFIG\_MinProfileCacheSize (5) // Minimum number of functions before profile is saved.

#define DEFAULT\_CONFIG\_ProfileDifferencePercent (15) // If 15% of the functions have different profile we will trigger a save.

// ES6 - BLUE

#define DEFAULT\_CONFIG\_\_\_proto\_\_ (true)

#define DEFAULT\_CONFIG\_LetConst (true)

#define DEFAULT\_CONFIG\_WeakMap (true)

#define DEFAULT\_CONFIG\_Map (true)

#define DEFAULT\_CONFIG\_Set (true)

#define DEFAULT\_CONFIG\_Intl (true)

#define DEFAULT\_CONFIG\_IntlBuiltIns (true)

// ES6 DEFAULT BEHAVIOR

#define DEFAULT\_CONFIG\_ES6 (true) // master flag to gate all P0-spec-test compliant ES6 features

//CollectGarbage is legacy IE specific global function disabled in Microsoft Edge.

#define DEFAULT\_CONFIG\_CollectGarbage (false)

// ES6 sub-feature gate - to enable-disable ES6 sub-feature when ES6 flag is enabled

#define DEFAULT\_CONFIG\_ES6Species (true)

#define DEFAULT\_CONFIG\_ES6AsyncAwait (false)

#define DEFAULT\_CONFIG\_ES6Classes (true)

#define DEFAULT\_CONFIG\_ES6DateParseFix (true)

#define DEFAULT\_CONFIG\_ES6DefaultArgs (false)

#define DEFAULT\_CONFIG\_ES6Destructuring (true)

#define DEFAULT\_CONFIG\_ES6ForLoopSemantics (true)

#define DEFAULT\_CONFIG\_ES6FunctionName (true)

#define DEFAULT\_CONFIG\_ES6FunctionNameFull (false)

#define DEFAULT\_CONFIG\_ES6Generators (true)

#define DEFAULT\_CONFIG\_ES6Iterators (true)

#define DEFAULT\_CONFIG\_ES6IsConcatSpreadable (false)

#define DEFAULT\_CONFIG\_ES6Lambda (true)

#define DEFAULT\_CONFIG\_ES6Math (true)

#define DEFAULT\_CONFIG\_ES6Object (true)

#define DEFAULT\_CONFIG\_ES6Number (true)

#define DEFAULT\_CONFIG\_ES6NumericLiterals (true)

#define DEFAULT\_CONFIG\_ES6ObjectLiterals (true)

#define DEFAULT\_CONFIG\_ES6Promise (true)

#define DEFAULT\_CONFIG\_ES6Proxy (true)

#define DEFAULT\_CONFIG\_ES6Rest (true)

#define DEFAULT\_CONFIG\_ES6Spread (true)

#define DEFAULT\_CONFIG\_ES6String (true)

#define DEFAULT\_CONFIG\_ES6StringPrototypeFixes (true)

#define DEFAULT\_CONFIG\_ES6StringTemplate (true)

#define DEFAULT\_CONFIG\_ES6PrototypeChain (false)

#define DEFAULT\_CONFIG\_ES6Symbol (true)

#define DEFAULT\_CONFIG\_ES6ToPrimitive (false)

#define DEFAULT\_CONFIG\_ES6ToLength (false)

#define DEFAULT\_CONFIG\_ES6ToStringTag (false)

#define DEFAULT\_CONFIG\_ES6TypedArrayExtensions (true)

#define DEFAULT\_CONFIG\_ES6Unicode (true)

#define DEFAULT\_CONFIG\_ES6UnicodeVerbose (true)

#define DEFAULT\_CONFIG\_ES6Unscopables (true)

#define DEFAULT\_CONFIG\_ES6WeakSet (true)

#define DEFAULT\_CONFIG\_ES6RegExSticky (true)

#define DEFAULT\_CONFIG\_ES6RegExPrototypeProperties (false)

#define DEFAULT\_CONFIG\_ES6HasInstanceOf (false)

#define DEFAULT\_CONFIG\_ArrayBufferTransfer (false)

#define DEFAULT\_CONFIG\_ES7ExponentionOperator (false)

#define DEFAULT\_CONFIG\_ES7Builtins (false)

#define DEFAULT\_CONFIG\_ES7TrailingComma (true)

#define DEFAULT\_CONFIG\_ES6Verbose (false)

#define DEFAULT\_CONFIG\_ES6All (false)

// ES6 DEFAULT BEHAVIOR

#define DEFAULT\_CONFIG\_AsyncDebugging (true)

#define DEFAULT\_CONFIG\_TraceAsyncDebugCalls (false)

#define DEFAULT\_CONFIG\_ForcePostLowerGlobOptInstrString (false)

#define DEFAULT\_CONFIG\_EnumerateSpecialPropertiesInDebugger (true)

#endif

#define DEFAULT\_CONFIG\_MaxJITFunctionBytecodeSize (120000)

#define DEFAULT\_CONFIG\_JitQueueThreshold (6)

#define DEFAULT\_CONFIG\_FullJitRequeueThreshold (25) // Minimum number of times a function needs to be executed before it is re-added to the jit queue

#define DEFAULT\_CONFIG\_MinTemplatizedJitRunCount (100) // Minimum number of times a function needs to be interpreted before it is jitted

#define DEFAULT\_CONFIG\_MinAsmJsInterpreterRunCount (10) // Minimum number of times a function needs to be Asm interpreted before it is jitted

#define DEFAULT\_CONFIG\_MinTemplatizedJitLoopRunCount (500) // Minimum number of times a function needs to be interpreted before it is jitted

#define DEFAULT\_CONFIG\_MaxTemplatizedJitRunCount (-1) // Maximum number of times a function can be TJ before it is jitted

#define DEFAULT\_CONFIG\_MaxAsmJsInterpreterRunCount (-1) // Maximum number of times a function can be Asm interpreted before it is jitted

// Note: The following defaults only apply when the NewSimpleJit is on. The defaults for when it's off are computed in

// ConfigFlagsTable::TranslateFlagConfiguration.

#define DEFAULT\_CONFIG\_AutoProfilingInterpreter0Limit (12)

#define DEFAULT\_CONFIG\_ProfilingInterpreter0Limit (4)

#define DEFAULT\_CONFIG\_AutoProfilingInterpreter1Limit (0)

#define DEFAULT\_CONFIG\_SimpleJitLimit (132)

#define DEFAULT\_CONFIG\_ProfilingInterpreter1Limit (12)

// These are used to compute the above defaults for when NewSimpleJit is off

#define DEFAULT\_CONFIG\_AutoProfilingInterpreterLimit\_OldSimpleJit (80)

#define DEFAULT\_CONFIG\_SimpleJitLimit\_OldSimpleJit (25)

#define DEFAULT\_CONFIG\_MinProfileIterations (16)

#define DEFAULT\_CONFIG\_MinProfileIterations\_OldSimpleJit (25)

#define DEFAULT\_CONFIG\_MinSimpleJitIterations (16)

#define DEFAULT\_CONFIG\_NewSimpleJit (false)

#define DEFAULT\_CONFIG\_MaxLinearIntCaseCount (3) // Maximum number of cases (in switch statement) for which instructions can be generated linearly.

#define DEFAULT\_CONFIG\_MaxSingleCharStrJumpTableRatio (2) // Maximum single char string jump table size as multiples of the actual case arm

#define DEFAULT\_CONFIG\_MaxSingleCharStrJumpTableSize (128) // Maximum single char string jump table size

#define DEFAULT\_CONFIG\_MinSwitchJumpTableSize (9) // Minimum number of case target entries in the jump table(this may also include values that are missing in the consecutive set of integer case arms)

#define DEFAULT\_CONFIG\_SwitchOptHolesThreshold (50) // Maximum percentage of holes (missing case values in a switch statement) with which a jump table can be created

#define DEFAULT\_CONFIG\_MaxLinearStringCaseCount (4) // Maximum number of String cases (in switch statement) for which instructions can be generated linearly.

#define DEFAULT\_CONFIG\_MinDeferredFuncTokenCount (20) // Minimum size in tokens of a defer-parsed function

#if DBG

#define DEFAULT\_CONFIG\_SkipFuncCountForBailOnNoProfile (0) //Initial Number of functions in a func body to be skipped from forcibly inserting BailOnNoProfile.

#endif

#define DEFAULT\_CONFIG\_BailOnNoProfileLimit 200 // The limit of bailout on no profile info before triggering a rejit

#define DEFAULT\_CONFIG\_BailOnNoProfileRejitLimit (-1) // The limit of bailout on no profile info before disable all the no profile bailouts

#define DEFAULT\_CONFIG\_RejitRatioLimit 5 // Ratio of function calls to bailouts on a single bailout record

// above which a rejit is considered

#define DEFAULT\_CONFIG\_MinBailOutsBeforeRejit 2 // Minimum number of bailouts for a single bailout record after which a rejit is considered

#define DEFAULT\_CONFIG\_RejitMaxBailOutCount 500 // Maximum number of bailouts for a single bailout record after which rejit is forced.

#define DEFAULT\_CONFIG\_Sse (-1)

#define DEFAULT\_CONFIG\_DeletedPropertyReuseThreshold (32)

#define DEFAULT\_CONFIG\_BigDictionaryTypeHandlerThreshold (0xffff)

#define DEFAULT\_CONFIG\_ForceStringKeyedSimpleDictionaryTypeHandler (false)

#define DEFAULT\_CONFIG\_TypeSnapshotEnumeration (true)

#define DEFAULT\_CONFIG\_EnumerationCompat (false)

#define DEFAULT\_CONFIG\_ConcurrentRuntime (false)

#define DEFAULT\_CONFIG\_PrimeRecycler (false)

#define DEFAULT\_CONFIG\_DisableRentalThreading (false)

#define DEFAULT\_CONFIG\_DisableDebugObject (false)

#define DEFAULT\_CONFIG\_DumpHeap (false)

#define DEFAULT\_CONFIG\_PerfHintLevel (1)

#define DEFAULT\_CONFIG\_FailFastIfDisconnectedDelegate (false)

#define DEFAULT\_CONFIG\_IsolatePrototypes (true)

#define DEFAULT\_CONFIG\_ChangeTypeOnProto (true)

#define DEFAULT\_CONFIG\_FixPropsOnPathTypes (true)

#define DEFAULT\_CONFIG\_BailoutTraceFilter (-1)

#define DEFAULT\_CONFIG\_TempMin (0)

#define DEFAULT\_CONFIG\_TempMax (MAXINT)

#define DEFAULT\_CONFIG\_LibraryStackFrame (true)

#define DEFAULT\_CONFIG\_LibraryStackFrameDebugger (false)

#define DEFAULT\_CONFIG\_FuncObjectInlineCacheThreshold (2) // Maximum number of inline caches a function body may have to allow for inline caches to be allocated on the function object.

#define DEFAULT\_CONFIG\_ShareInlineCaches (true)

#define DEFAULT\_CONFIG\_InlineCacheInvalidationListCompactionThreshold (4)

#define DEFAULT\_CONFIG\_InMemoryTrace (false)

#define DEFAULT\_CONFIG\_InMemoryTraceBufferSize (1024)

#define DEFAULT\_CONFIG\_RichTraceFormat (false)

#define DEFAULT\_CONFIG\_TraceWithStack (false)

#define DEFAULT\_CONFIG\_InjectPartiallyInitializedInterpreterFrameError (0)

#define DEFAULT\_CONFIG\_InjectPartiallyInitializedInterpreterFrameErrorType (0)

#define DEFAULT\_CONFIG\_InvalidateSolutionContextsForGetStructure (true)

#define DEFAULT\_CONFIG\_DeferLoadingAvailableSource (false)

#ifdef ENABLE\_PROJECTION

#define DEFAULT\_CONFIG\_NoWinRTFastSig (false)

#define DEFAULT\_CONFIG\_TargetWinRTVersion (1)

#define DEFAULT\_CONFIG\_WinRTDelegateInterfaces (false)

#define DEFAULT\_CONFIG\_WinRTAdaptiveApps (true)

#endif

#define DEFAULT\_CONFIG\_RecyclerForceMarkInterior (false)

#define DEFAULT\_CONFIG\_MemProtectHeap (false)

#define DEFAULT\_CONFIG\_InduceCodeGenFailure (30) // When -InduceCodeGenFailure is passed in, 30% of JIT allocations will fail

#define DEFAULT\_CONFIG\_SkipSplitWhenResultIgnored (false)

#define DEFAULT\_CONFIG\_MinMemOpCount (16U)

#if ENABLE\_COPYONACCESS\_ARRAY

#define DEFAULT\_CONFIG\_MaxCopyOnAccessArrayLength (32U)

#define DEFAULT\_CONFIG\_MinCopyOnAccessArrayLength (5U)

#define DEFAULT\_CONFIG\_CopyOnAccessArraySegmentCacheSize (16U)

#endif

#if defined(\_M\_IX86) || defined(\_M\_X64)

#define DEFAULT\_CONFIG\_LoopAlignNopLimit (6)

#endif

#define TraceLevel\_Error (1)

#define TraceLevel\_Warning (2)

#define TraceLevel\_Info (3)

#define TEMP\_ENABLE\_FLAG\_FOR\_APPX\_BETA\_ONLY 1

#define INMEMORY\_CACHE\_MAX\_URL (5) // This is the max number of URLs that the in-memory profile cache can hold.

#define INMEMORY\_CACHE\_MAX\_PROFILE\_MANAGER (50) // This is the max number of dynamic scripts that the in-memory profile cache can have

#ifdef SUPPORT\_INTRUSIVE\_TESTTRACES

#define INTRUSIVE\_TESTTRACE\_PolymorphicInlineCache (1)

#endif

//

//FLAG(type, name, description, defaultValue)

//

// types:

// String

// Phases

// Number

// Boolean

//

// If the default value is not required it should be left empty

// For Phases, there is no default value. it should always be empty

// Default values for stings must be prefixed with 'L'. See AsmDumpMode

// Scroll till the extreme right to see the default values

#if defined(FLAG) || defined(FLAG\_REGOVR\_EXP)

#ifndef FLAG

#define FLAG(...)

#endif

#ifndef FLAG\_REGOVR\_ASMJS

#define FLAG\_REGOVR\_ASMJS FLAG

#endif

#ifndef FLAG\_REGOVR\_EXP

#define FLAG\_REGOVR\_EXP FLAG

#endif

// NON-RELEASE FLAGS

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

// Regular FLAG

#define FLAGNR(Type, Name, String, Default) FLAG(Type, Name, String, Default, NoParent, FALSE)

// Regular flag with acronym

#ifndef FLAGNRA

#define FLAGNRA(Type, Name, Acronym, String, Default) \

FLAGNR(Type, Name, String, Default) \

FLAGNR(Type, Acronym, String, Default)

#endif

// Child FLAG with PARENT FLAG

#define FLAGPNR(Type, ParentName, Name, String, Default) FLAG(Type, Name, String, Default, ParentName, FALSE)

// Regular FLAG with callback function

#define FLAGNRC(Type, Name, String, Default) FLAG(Type, Name, String, Default, NoParent, TRUE)

#else

#define FLAGNR(Type, Name, String, Default)

#ifdef FLAGNRA

#undef FLAGNRA

#endif

#define FLAGNRA(Type, Name, Acronym, String, Default)

#define FLAGPNR(Type, ParentName, Name, String, Default)

#define FLAGNRC(Type, Name, String, Default)

#endif

// RELEASE FLAGS

#define FLAGPR(Type, ParentName, Name, String, Default) FLAG(Type, Name, String, Default, ParentName, FALSE)

#define FLAGR(Type, Name, String, Default) FLAG(Type, Name, String, Default, NoParent, FALSE)

// RELEASE FLAGS WITH REGISTRY OVERRIDE

#define FLAGPR\_REGOVR\_ASMJS(Type, ParentName, Name, String, Default) FLAG\_REGOVR\_ASMJS(Type, Name, String, Default, ParentName, FALSE)

#define FLAGPR\_REGOVR\_EXP(Type, ParentName, Name, String, Default) FLAG\_REGOVR\_EXP(Type, Name, String, Default, ParentName, FALSE)

// Release flag with non-release acronym

#ifndef FLAGRA

#define FLAGRA(Type, Name, Acronym, String, Default) \

FLAGR(Type, Name, String, Default) \

FLAGNR(Type, Acronym, String, Default)

#endif

// Please keep this list alphabetically sorted

#if DBG

FLAGNR(Boolean, ArrayValidate , "Validate each array for valid elements (default: false)", false)

FLAGNR(Boolean, MemOpMissingValueValidate, "Validate Missing Value Tracking on memset/memcopy", false)

#endif

#ifdef ARENA\_MEMORY\_VERIFY

FLAGNR(Boolean, ArenaNoFreeList , "Do not free list in arena", false)

FLAGNR(Boolean, ArenaNoPageReuse , "Do not reuse page in arena", false)

FLAGNR(Boolean, ArenaUseHeapAlloc , "Arena use heap to allocate memory instead of page allocator", false)

#endif

FLAGNR(Boolean, ValidateInlineStack, "Does a stack walk on helper calls to validate inline stack is correctly restored", false)

FLAGNR(Boolean, AsmDiff , "Dump the IR without memory locations and varying parameters.", false)

FLAGNR(String, AsmDumpMode , "Dump the final assembly to a file without memory locations and varying parameters\n\t\t\t\t\tThe 'filename' is the file where the assembly will be dumped. Dump to console if no file is specified", nullptr)

FLAGR (Boolean, Asmjs , "Enable Asmjs", DEFAULT\_CONFIG\_ASMJS)

FLAGNR(Boolean, AsmJsStopOnError , "Stop execution on any AsmJs validation errors", DEFAULT\_CONFIG\_AsmJsStopOnError)

FLAGNR(Boolean, AsmJsEdge , "Enable asm.js features which may have backward incompatible changes or not validate on old demos", DEFAULT\_CONFIG\_AsmJsEdge)

FLAGPR\_REGOVR\_EXP(Boolean, ES6, Simdjs, "Enable Simdjs", DEFAULT\_CONFIG\_SIMDJS)

FLAGR(Boolean, Simd128TypeSpec, "Enable type-specialization of Simd128 symbols", false)

FLAGNR(Boolean, AssertBreak , "Debug break on assert", false)

FLAGNR(Boolean, AssertPopUp , "Pop up asserts (default: false)", false)

FLAGNR(Boolean, AssertIgnore , "Ignores asserts if set", false)

FLAGNR(Boolean, AsyncDebugging, "Enable async debugging feature (default: false)", DEFAULT\_CONFIG\_AsyncDebugging)

FLAGNR(Number, BailOnNoProfileLimit, "The limit of bailout on no profile info before triggering a rejit", DEFAULT\_CONFIG\_BailOnNoProfileLimit)

FLAGNR(Number, BailOnNoProfileRejitLimit, "The limit of bailout on no profile info before we disable the bailouts", DEFAULT\_CONFIG\_BailOnNoProfileRejitLimit)

FLAGNR(String, DumpOnCrash , "generate heap dump on asserts or unhandled exception if set", nullptr)

FLAGNR(String, FullMemoryDump , "Will perform a full memory dump when -DumpOnCrash is supplied.", nullptr)

#ifdef BAILOUT\_INJECTION

FLAGR (NumberPairSet, BailOut , "Source location to insert BailOut", )

FLAGNR(Boolean, BailOutAtEveryLine , "Inserts BailOut at every line of source (default: false)", false)

FLAGNR(Boolean, BailOutAtEveryByteCode, "Inserts BailOut at every Byte code (default: false)", false)

FLAGNR(Boolean, BailOutAtEveryImplicitCall, "Force generating implicit call bailout even when we don't need it", false)

FLAGR (NumberSet, BailOutByteCode , "Byte code location to insert BailOut. Use with -prejit only", )

#endif

FLAGNR(Boolean, Benchmark , "Disable security code which introduce variability in benchmarks", false)

FLAGR (Boolean, BgJit , "Background JIT. Disable to force heuristic-based foreground JITting. (default: true)", true)

FLAGNR(Number, BgJitDelay , "Delay to wait for speculative jitting before starting script execution", DEFAULT\_CONFIG\_BgJitDelay)

FLAGNR(Number, BgJitDelayFgBuffer , "When speculatively jitting in the foreground thread, do so for (BgJitDelay - BgJitDelayBuffer) milliseconds", DEFAULT\_CONFIG\_BgJitDelayFgBuffer)

FLAGNR(Number, BgJitPendingFuncCap , "Disable delay if pending function count larger then cap", DEFAULT\_CONFIG\_BgJitPendingFuncCap)

FLAGNR(Boolean, CreateFunctionProxy , "Create function proxies instead of full function bodies", DEFAULT\_CONFIG\_CreateFunctionProxy)

FLAGNR(Boolean, HybridFgJit , "When background JIT is enabled, enable jitting in the foreground based on heuristics. This flag is only effective when OptimizeForManyInstances is disabled (UI threads).", DEFAULT\_CONFIG\_HybridFgJit)

FLAGNR(Number, HybridFgJitBgQueueLengthThreshold, "The background job queue length must exceed this threshold to consider jitting in the foreground", DEFAULT\_CONFIG\_HybridFgJitBgQueueLengthThreshold)

FLAGNR(Boolean, BytecodeHist , "Provide a histogram of the bytecodes run by the script. (NoNative required).", false)

FLAGNR(Boolean, CurrentSourceInfo , "Enable IASD get current script source info", DEFAULT\_CONFIG\_CurrentSourceInfo)

FLAGNR(Boolean, CFGLog , "Log CFG checks", false)

FLAGNR(Boolean, CheckAlignment , "Insert checks in the native code to verify 8-byte alignment of stack", false)

FLAGNR(Boolean, CheckEmitBufferPermissions, "Check JIT code buffers at commit and decommit time to ensure no PAGE\_EXECUTE\_READWRITE pages.", false)

#ifdef CHECK\_MEMORY\_LEAK

FLAGR (Boolean, CheckMemoryLeak , "Check for heap memory leak", false)

FLAGR (String, DumpOnLeak , "Create a dump on failed memory leak check", nullptr)

#endif

FLAGNR(Boolean, CheckOpHelpers , "Verify opHelper labels in the JIT are set properly", false)

FLAGNR(Boolean, CloneInlinedPolymorphicCaches, "Clones polymorphic inline caches in inlined functions", DEFAULT\_CONFIG\_CloneInlinedPolymorphicCaches)

FLAGNR(Boolean, ConcurrentRuntime , "Enable Concurrent GC and background JIT when creating runtime", DEFAULT\_CONFIG\_ConcurrentRuntime)

FLAGNR(Boolean, Console , "Create console window in GUI app", false)

FLAGNR(Boolean, ConsoleExitPause , "Pause on exit when a console window is created in GUI app", false)

FLAGNR(Number, ConstructorInlineThreshold , "Maximum size in bytecodes of an constructor inline candidate with monomorphic field access", DEFAULT\_CONFIG\_ConstructorInlineThreshold)

FLAGNR(Number, ConstructorCallsRequiredToFinalizeCachedType, "Number of calls to a constructor required before the type cached in the constructor cache is finalized", DEFAULT\_CONFIG\_ConstructorCallsRequiredToFinalizeCachedType)

#ifdef SECURITY\_TESTING

FLAGNR(Boolean, CrashOnException , "Removes the top-level exception handler, allowing jc.exe to crash on an unhandled exception. No effect on IE. (default: false)", false)

#endif

FLAGNR(Boolean, Debug , "Disable phases (layout, security code, etc) which makes JIT output harder to debug", false)

FLAGNR(NumberSet, DebugBreak , "Index of the function where you want to break", )

FLAGNR(Boolean, DebugWindow , "Send console output to debugger window", false)

FLAGNR(Boolean, DeferNested , "Enable deferred parsing of nested function", DEFAULT\_CONFIG\_DeferNested)

FLAGNR(Boolean, DeferTopLevelTillFirstCall , "Enable tracking of deferred top level functions in a script file, until the first function of the script context is parsed.", DEFAULT\_CONFIG\_DeferTopLevelTillFirstCall)

FLAGNR(Number, DeferParse , "Minimum size of defer-parsed script (non-zero only: use /nodeferparse do disable", 0)

FLAGNR(Boolean, DefineGetterSetter , "Enable IE11 \_\_defineGetter\_\_ and \_\_defineSetter\_\_ methods", DEFAULT\_CONFIG\_DefineGetterSetter)

FLAGNR(Boolean, DirectCallTelemetryStats, "Enables logging stats for direct call telemetry", DEFAULT\_CONFIG\_DirectCallTelemetryStats)

FLAGNR(Boolean, DisableArrayBTree , "Disable creation of BTree for Arrays", false)

FLAGNR(Boolean, DisableRentalThreading, "Disable rental threading when creating runtime", DEFAULT\_CONFIG\_DisableRentalThreading)

FLAGNR(Boolean, DisableVTuneSourceLineInfo, "Disable VTune Source line info for Dynamic JITted code", false)

FLAGNR(Boolean, DisplayMemStats, "Display memory usage statistics", false)

FLAGNR(Phases, Dump , "What All to dump", )

#ifdef DUMP\_FRAGMENTATION\_STATS

FLAGR (Boolean, DumpFragmentationStats, "Dump bucket state after every GC", false)

#endif

FLAGNR(Boolean, DumpIRAddresses, "Print addresses in IR dumps", false)

FLAGNR(Boolean, DumpLineNoInColor, "Print the source code in high intensity color for better readability", false)

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

FLAGR (Boolean, DumpObjectGraphOnExit , "Dump object graph on recycler destructor", false)

FLAGR (Boolean, DumpObjectGraphOnCollect, "Dump object graph on recycler destructor", false)

#endif

FLAGNR(Boolean, DumpEvalStringOnRemoval, "Dumps an eval string when its being removed from the eval map", false)

FLAGNR(Boolean, DumpObjectGraphOnEnum, "Dump object graph on recycler heap enumeration", false)

#ifdef DYNAMIC\_PROFILE\_STORAGE

FLAGNRA(String, DynamicProfileCache , Dpc, "File to cache dynamic profile information", nullptr)

FLAGNR(String, DynamicProfileCacheDir, "Directory to cache dynamic profile information", nullptr)

FLAGNRA(String, DynamicProfileInput , Dpi, "Read only file containing dynamic profile information", nullptr)

#endif

#ifdef EDIT\_AND\_CONTINUE

FLAGNR(Boolean, EditTest , "Enable edit and continue test tools", false)

#endif

FLAGNR(Boolean, WininetProfileCache, "Use the WININET cache to save the profile information", DEFAULT\_CONFIG\_WininetProfileCache)

FLAGNR(Boolean, NoDynamicProfileInMemoryCache, "Enable in-memory cache for dynamic sources", false)

FLAGNR(Boolean, ProfileBasedSpeculativeJit, "Enable dynamic profile based speculative JIT", DEFAULT\_CONFIG\_ProfileBasedSpeculativeJit)

FLAGNR(Number, ProfileBasedSpeculationCap, "In the presence of dynamic profile speculative JIT is capped to this many bytecode instructions", DEFAULT\_CONFIG\_ProfileBasedSpeculationCap)

#ifdef DYNAMIC\_PROFILE\_MUTATOR

FLAGNR(String, DynamicProfileMutatorDll , "Path of the mutator DLL", L"DynamicProfileMutatorImpl.dll")

FLAGNR(String, DynamicProfileMutator , "Type of local, temp, return, param, loop implicit flag and implicit flag. \n\t\t\t\t\ti.e local=LikelyArray\_NoMissingValues\_NonInts\_NonFloats;temp=Int8Array;param=LikelyNumber;return=LikelyString;loopimplicitflag=ImplicitCall\_ToPrimitive;implicitflag=ImplicitCall\_None\n\t\t\t\t\tor pass DynamicProfileMutator:random\n\t\t\t\t\tSee DynamicProfileInfo.h for enum values", nullptr)

#endif

FLAGNR(Boolean, ExecuteByteCodeBufferReturnsInvalidByteCode, "Serialized byte code execution always returns SCRIPT\_E\_INVALID\_BYTECODE", false)

FLAGR(Number, ExpirableCollectionGCCount, "Number of GCs during which Expirable object profiling occurs", DEFAULT\_CONFIG\_ExpirableCollectionGCCount)

FLAGR (Number, ExpirableCollectionTriggerThreshold, "Threshold at which Expirable Object Collection is triggered (In Percentage)", DEFAULT\_CONFIG\_ExpirableCollectionTriggerThreshold)

FLAGR(Boolean, SkipSplitOnNoResult, "If the result of Regex split isn't used, skip executing the regex. (Perf optimization)", DEFAULT\_CONFIG\_SkipSplitWhenResultIgnored)

#ifdef TEST\_ETW\_EVENTS

FLAGNR(String, TestEtwDll , "Path of the TestEtwEventSink DLL", nullptr)

#endif

// ES6 (BLUE-shipped) features/flags

FLAGNR(Boolean, \_\_proto\_\_ , "\_\_proto\_\_ support", DEFAULT\_CONFIG\_\_\_proto\_\_)

FLAGNR(Boolean, LetConst , "Enable support of 'let' and 'const' keyword", DEFAULT\_CONFIG\_LetConst)

FLAGNR(Boolean, Map , "Enable ES6 Map feature", DEFAULT\_CONFIG\_Map)

FLAGNR(Boolean, Set , "Enable ES6 Set feature", DEFAULT\_CONFIG\_Set)

FLAGNR(Boolean, WeakMap , "Enable ES6 WeakMap feature", DEFAULT\_CONFIG\_WeakMap)

FLAGNR(Boolean, CollectGarbage , "Enable CollectGarbage API", DEFAULT\_CONFIG\_CollectGarbage)

FLAGR (Boolean, Intl , "Intl object support", DEFAULT\_CONFIG\_Intl)

FLAGNR(Boolean, IntlBuiltIns , "Intl built-in function support", DEFAULT\_CONFIG\_IntlBuiltIns)

// ES6 (BLUE+1) features/flags

// Master ES6 flag to enable STABLE ES6 features/flags

FLAGR(Boolean, ES6 , "Enable ES6 stable features", DEFAULT\_CONFIG\_ES6)

// Master ES6 flag to enable ALL sub ES6 features/flags

FLAGNRC(Boolean, ES6All , "Enable all ES6 features, both stable and unstable", DEFAULT\_CONFIG\_ES6All)

// Master ES6 flag to enable Threshold ES6 features/flags

FLAGNRC(Boolean, ES6Experimental , "Enable all experimental features", DEFAULT\_CONFIG\_ES6All)

// Per ES6 feature/flag

FLAGPR (Boolean, ES6, ES6Species , "Enable ES6 '@@species' properties and built-in behaviors" , DEFAULT\_CONFIG\_ES6Species)

FLAGPR\_REGOVR\_EXP(Boolean, ES6, ES7AsyncAwait , "Enable ES7 'async' and 'await' keywords" , DEFAULT\_CONFIG\_ES6AsyncAwait)

FLAGPR (Boolean, ES6, ES6Classes , "Enable ES6 'class' and 'extends' keywords" , DEFAULT\_CONFIG\_ES6Classes)

FLAGPR (Boolean, ES6, ES6DateParseFix , "Enable ES6 Date.parse fixes" , DEFAULT\_CONFIG\_ES6DateParseFix)

FLAGPR\_REGOVR\_EXP(Boolean, ES6, ES6DefaultArgs , "Enable ES6 Default Arguments" , DEFAULT\_CONFIG\_ES6DefaultArgs)

FLAGPR (Boolean, ES6, ES6Destructuring , "Enable ES6 Destructuring" , DEFAULT\_CONFIG\_ES6Destructuring)

FLAGPR (Boolean, ES6, ES6ForLoopSemantics , "Enable ES6 for loop per iteration bindings" , DEFAULT\_CONFIG\_ES6ForLoopSemantics)

FLAGPR (Boolean, ES6, ES6FunctionName , "Enable ES6 function.name" , DEFAULT\_CONFIG\_ES6FunctionName)

FLAGPR\_REGOVR\_EXP(Boolean, ES6, ES6FunctionNameFull , "Enable ES6 Full function.name" , DEFAULT\_CONFIG\_ES6FunctionNameFull)

FLAGPR (Boolean, ES6, ES6Generators , "Enable ES6 generators" , DEFAULT\_CONFIG\_ES6Generators)

FLAGPR\_REGOVR\_EXP(Boolean, ES6, ES7ExponentiationOperator, "Enable ES7 exponentiation operator (\*\*)" , DEFAULT\_CONFIG\_ES7ExponentionOperator)

FLAGPR\_REGOVR\_EXP(Boolean, ES6, ES7Builtins , "Enable ES7 built-ins" , DEFAULT\_CONFIG\_ES7Builtins)

FLAGPR (Boolean, ES6, ES7TrailingComma , "Enable ES7 trailing comma in function" , DEFAULT\_CONFIG\_ES7TrailingComma)

FLAGPR (Boolean, ES6, ES6Iterators , "Enable ES6 iterators" , DEFAULT\_CONFIG\_ES6Iterators)

FLAGPR (Boolean, ES6, ES6IsConcatSpreadable , "Enable ES6 isConcatSpreadable Symbol" , DEFAULT\_CONFIG\_ES6IsConcatSpreadable)

FLAGPR (Boolean, ES6, ES6Lambda , "Enable ES6 lambdas" , DEFAULT\_CONFIG\_ES6Lambda)

FLAGPR (Boolean, ES6, ES6Math , "Enable ES6 Math extensions" , DEFAULT\_CONFIG\_ES6Math)

FLAGPR (Boolean, ES6, ES6Object , "Enable ES6 Object extensions" , DEFAULT\_CONFIG\_ES6Object)

FLAGPR (Boolean, ES6, ES6Number , "Enable ES6 Number extensions" , DEFAULT\_CONFIG\_ES6Number)

FLAGPR (Boolean, ES6, ES6NumericLiterals , "Enable ES6 NumericLiterals" , DEFAULT\_CONFIG\_ES6NumericLiterals)

FLAGPR (Boolean, ES6, ES6ObjectLiterals , "Enable ES6 Object literal extensions" , DEFAULT\_CONFIG\_ES6ObjectLiterals)

FLAGPR (Boolean, ES6, ES6Promise , "Enable ES6 Promise feature" , DEFAULT\_CONFIG\_ES6Promise)

FLAGPR (Boolean, ES6, ES6Proxy , "Enable ES6 Proxy feature" , DEFAULT\_CONFIG\_ES6Proxy)

FLAGPR (Boolean, ES6, ES6Rest , "Enable ES6 Rest parameters" , DEFAULT\_CONFIG\_ES6Rest)

FLAGPR (Boolean, ES6, ES6Spread , "Enable ES6 Spread support" , DEFAULT\_CONFIG\_ES6Spread)

FLAGPR (Boolean, ES6, ES6String , "Enable ES6 String extensions" , DEFAULT\_CONFIG\_ES6String)

FLAGPR (Boolean, ES6, ES6StringPrototypeFixes, "Enable ES6 String.prototype fixes" , DEFAULT\_CONFIG\_ES6StringPrototypeFixes)

FLAGPR (Boolean, ES6, ES6StringTemplate , "Enable ES6 string template feature" , DEFAULT\_CONFIG\_ES6StringTemplate)

FLAGPR\_REGOVR\_EXP(Boolean, ES6, ES6PrototypeChain , "Enable ES6 prototypes (Example: Date prototype is object)", DEFAULT\_CONFIG\_ES6PrototypeChain)

FLAGPR (Boolean, ES6, ES6Symbol , "Enable ES6 Symbol feature" , DEFAULT\_CONFIG\_ES6Symbol)

FLAGPR (Boolean, ES6, ES6ToPrimitive , "Enable ES6 ToPrimitve symbol" , DEFAULT\_CONFIG\_ES6ToPrimitive)

FLAGPR (Boolean, ES6, ES6ToLength , "Enable ES6 ToLength fixes" , DEFAULT\_CONFIG\_ES6ToLength)

FLAGPR (Boolean, ES6, ES6ToStringTag , "Enable ES6 ToStringTag symbol" , DEFAULT\_CONFIG\_ES6ToStringTag)

FLAGPR (Boolean, ES6, ES6TypedArrayExtensions, "Enable ES6 TypedArray extensions" , DEFAULT\_CONFIG\_ES6TypedArrayExtensions)

FLAGPR (Boolean, ES6, ES6Unicode , "Enable ES6 Unicode 6.0 extensions" , DEFAULT\_CONFIG\_ES6Unicode)

FLAGPR (Boolean, ES6, ES6UnicodeVerbose , "Enable ES6 Unicode 6.0 verbose failure output" , DEFAULT\_CONFIG\_ES6UnicodeVerbose)

FLAGPR (Boolean, ES6, ES6Unscopables , "Enable ES6 With Statement Unscopables" , DEFAULT\_CONFIG\_ES6Unscopables)

FLAGPR (Boolean, ES6, ES6WeakSet , "Enable ES6 WeakSet" , DEFAULT\_CONFIG\_ES6WeakSet)

FLAGPR (Boolean, ES6, ES6RegExSticky , "Enable ES6 RegEx sticky flag" , DEFAULT\_CONFIG\_ES6RegExSticky)

FLAGPR\_REGOVR\_EXP(Boolean, ES6, ES6RegExPrototypeProperties, "Enable ES6 properties on the RegEx prototype" , DEFAULT\_CONFIG\_ES6RegExPrototypeProperties)

FLAGPR (Boolean, ES6, ES6HasInstance , "Enable ES6 @@hasInstance symbol" , DEFAULT\_CONFIG\_ES6HasInstanceOf)

FLAGPR (Boolean, ES6, ES6Verbose , "Enable ES6 verbose trace" , DEFAULT\_CONFIG\_ES6Verbose)

FLAGPR\_REGOVR\_EXP(Boolean, ES6, ArrayBufferTransfer , "Enable ArrayBuffer.transfer" , DEFAULT\_CONFIG\_ArrayBufferTransfer)

// /ES6 (BLUE+1) features/flags

#ifdef ENABLE\_PROJECTION

FLAGNR(Boolean, WinRTDelegateInterfaces , "Treat WinRT Delegates as Interfaces when determining their resolvability.", DEFAULT\_CONFIG\_WinRTDelegateInterfaces)

FLAGR(Boolean, WinRTAdaptiveApps , "Enable the adaptive apps feature, allowing for variable projection." , DEFAULT\_CONFIG\_WinRTAdaptiveApps)

#endif

// This flag to be removed once JITing generator functions is stable

FLAGNR(Boolean, JitES6Generators , "Enable JITing of ES6 generators", false)

FLAGNR(Number, FastPathCap , "Cap in source code size for enabling fast-paths", DEFAULT\_CONFIG\_FastPathCap)

FLAGNR(Boolean, FastLineColumnCalculation, "Enable fast calculation of line/column numbers from the source.", DEFAULT\_CONFIG\_FastLineColumnCalculation)

FLAGR (String, Filename , "Jscript source file", nullptr)

FLAGNR(Boolean, FreeRejittedCode , "Free rejitted code", true)

FLAGNR(Boolean, ForceGuardPages , "Force the addition of guard pages", false)

FLAGNR(Boolean, PrintGuardPageBounds , "Prints the bounds of a guard page", false)

FLAGNR(Boolean, ForceLegacyEngine , "Force a jscrip9 dll load", false)

FLAGNR(Phases, Force , "Force certain phase to run ignoring heuristics", )

FLAGNR(Phases, Stress , "Stress certain phases by making them kick in even if they normally would not.", )

FLAGNR(Boolean, ForceArrayBTree , "Force enable creation of BTree for Arrays", false)

FLAGNR(Boolean, ForceCleanPropertyOnCollect, "Force cleaning of property on collection", DEFAULT\_CONFIG\_ForceCleanPropertyOnCollect)

FLAGNR(Boolean, ForceCleanCacheOnCollect, "Force cleaning of dynamic caches on collection", DEFAULT\_CONFIG\_ForceCleanCacheOnCollect)

FLAGNR(Boolean, ForceGCAfterJSONParse, "Force GC to happen after JSON parsing", DEFAULT\_CONFIG\_ForceGCAfterJSONParse)

FLAGNR(Boolean, ForceDecommitOnCollect, "Force decommit collect", DEFAULT\_CONFIG\_ForceDecommitOnCollect)

FLAGNR(Boolean, ForceDeferParse , "Defer parsing of all function bodies", DEFAULT\_CONFIG\_ForceDeferParse)

FLAGNR(Boolean, ForceDiagnosticsMode , "Enable diagnostics mode and debug interpreter loop", false)

FLAGNR(Boolean, ForceGetWriteWatchOOM , "Force GetWriteWatch to go into OOM codepath in HeapBlockMap rescan", false)

FLAGNR(Boolean, DumpDbgControllerBytecode, "Dump dbgcontroller.js bytecode when -dump:bytecode is specified (default false)", false)

FLAGNR(Boolean, ForcePostLowerGlobOptInstrString, "Force tracking of globopt instr string post lower", DEFAULT\_CONFIG\_ForcePostLowerGlobOptInstrString)

FLAGNR(Boolean, EnumerateSpecialPropertiesInDebugger, "Enable enumeration of special debug properties", DEFAULT\_CONFIG\_EnumerateSpecialPropertiesInDebugger)

FLAGNR(Boolean, EnableJitInDiagMode , "Enable Fast F12 (only applicable with ForceDiagnosticsMode or while under debugger)", DEFAULT\_CONFIG\_EnableJitInDiagMode)

FLAGR (Boolean, EnableJitInHybridDebugging, "Enable Fast Debugging for Hybrid Debugging. Node: to turn this ON in full, EnableJitInDiagMode must be ON as well.", DEFAULT\_CONFIG\_EnableJitInHybridDebugging)

FLAGNR(Boolean, EnableContinueAfterExceptionWrappersForHelpers, "Enable wrapper over helper methods in debugger, Fast F12 only", DEFAULT\_CONFIG\_EnableContinueAfterExceptionWrappersForHelpers)

FLAGNR(Boolean, EnableContinueAfterExceptionWrappersForBuiltIns, "Enable wrapper over library calls in debugger, Fast F12 only", DEFAULT\_CONFIG\_EnableContinueAfterExceptionWrappersForBuiltIns)

FLAGNR(Boolean, EnableFunctionSourceReportForHeapEnum, "During HeapEnum, whether to report function source info (url/row/col)", DEFAULT\_CONFIG\_EnableFunctionSourceReportForHeapEnum)

FLAGNR(Number, ForceFragmentAddressSpace , "Fragment the address space", 128 \* 1024 \* 1024)

FLAGNR(Number, ForceOOMOnEBCommit, "Force CommitBuffer to return OOM", 0)

FLAGR (Boolean, ForceDynamicProfile , "Force to always generate profiling byte code", DEFAULT\_CONFIG\_ForceDynamicProfile)

FLAGNR(Boolean, ForceES5Array , "Force using ES5Array", DEFAULT\_CONFIG\_ForceES5Array)

FLAGNR(Boolean, ForceAsmJsLinkFail , "Force asm.js link time validation to fail", DEFAULT\_CONFIG\_ForceAsmJsLinkFail)

FLAGNR(Boolean, ForceExpireOnNonCacheCollect, "Allow expiration collect outside of cache collection cleanups", DEFAULT\_CONFIG\_ForceExpireOnNonCacheCollect)

FLAGNR(Boolean, ForceFastPath , "Force fast-paths in native codegen", DEFAULT\_CONFIG\_ForceFastPath)

FLAGNR(Boolean, ForceFloatPref , "Force float preferencing (JIT only)", false)

FLAGNR(Boolean, ForceJITLoopBody , "Force jit loop body only", DEFAULT\_CONFIG\_ForceJITLoopBody)

FLAGNR(Boolean, DumpCommentsFromReferencedFiles, "Allow printing comments of comment-table of the referenced file as well (use with -trace:CommentTable)", DEFAULT\_CONFIG\_DumpCommentsFromReferencedFiles)

#ifdef \_M\_ARM

FLAGNR(Boolean, ForceLocalsPtr , "Force use of alternative locals pointer (JIT only)", false)

#endif

FLAGNR(Boolean, DeferLoadingAvailableSource, "Treat available source code as a dummy defer-mappable object to go through that code path.", DEFAULT\_CONFIG\_DeferLoadingAvailableSource)

FLAGR (Boolean, ForceNative , "Force JIT everything that is called before running it, ignoring limits", false)

FLAGNR(Boolean, ForceSerialized , "Always serialize and deserialize byte codes before execution", DEFAULT\_CONFIG\_ForceSerialized)

FLAGNR(Number, ForceSerializedBytecodeMajorVersion, "Force the byte code serializer to write this major version number", 0)

FLAGNR(Number, ForceSerializedBytecodeVersionSchema, "Force the byte code serializer to write this kind of version. Decimal 10 is engineering, 20 is release mode, and 0 means use the default setting.", 0)

FLAGNR(Boolean, ForceStrictMode, "Force strict mode checks on all functions", false)

FLAGNR(Boolean, ForceUndoDefer , "Defer parsing of all function bodies, but undo deferral", false)

FLAGNR(Boolean, ForceBlockingConcurrentCollect, "Force doing in-thread GC on concurrent thread- this will skip doing concurrent collect", false)

FLAGNR(Boolean, FreTestDiagMode, "Enabled collection of diagnostic information on fretest builds", false)

#ifdef BYTECODE\_TESTING

FLAGNR(Number, ByteCodeBranchLimit, "Short branch limit before we use the branch island", 128)

FLAGNR(Boolean, MediumByteCodeLayout , "Always use medium layout for bytecodes", false)

FLAGNR(Boolean, LargeByteCodeLayout , "Always use large layout for bytecodes", false)

#endif

#ifdef FAULT\_INJECTION

FLAGNR(Number, FaultInjection , "FaultInjectMode - 0 (count only), 1 (count equal), 2 (count at or above), 3 (stackhashing)",-1)

FLAGNR(Number, FaultInjectionCount , "Injects an out of memory at the specified allocation", -1)

FLAGNR(String, FaultInjectionType , "FaultType (flag values) - 1 (Throw), 2 (NoThrow), 4 (MarkThrow), 8 (MarkNoThrow), FFFFFFFF (All)", nullptr)

FLAGNR(String, FaultInjectionFilter , "A string to restrict the fault injection, the string can be like ArenaAllocator name", nullptr)

FLAGNR(Number, FaultInjectionAllocSize, "Do fault injection only this size", -1)

FLAGNR(String, FaultInjectionStackFile , "Stacks to match, default: stack.txt in current directory", L"stack.txt")

FLAGNR(Number, FaultInjectionStackLineCount , "Count of lines in the stack file used for matching", -1)

FLAGNR(String, FaultInjectionStackHash, "Match stacks hash on Chakra frames to inject the fault, hex string", L"0")

FLAGNR(Number, FaultInjectionScriptContextToTerminateCount, "Script context# COUNT % (Number of script contexts) to terminate", 1)

#endif

FLAGNR(Number, InduceCodeGenFailure, "Probability of a codegen job failing.", DEFAULT\_CONFIG\_InduceCodeGenFailure)

FLAGNR(Number, InduceCodeGenFailureSeed, "Seed used while calculating codegen failure probability", 0)

FLAGNR(Number, InjectPartiallyInitializedInterpreterFrameError, "The number of interpreter stack frame (with 1 being bottom-most) to inject error before the frame is initialized.", DEFAULT\_CONFIG\_InjectPartiallyInitializedInterpreterFrameError)

FLAGNR(Number, InjectPartiallyInitializedInterpreterFrameErrorType, "Type of error to inject: 0 - debug break, 1 - exception.", DEFAULT\_CONFIG\_InjectPartiallyInitializedInterpreterFrameErrorType)

FLAGNR(Boolean, GenerateByteCodeBufferReturnsCantGenerate, "Serialized byte code generation always returns SCRIPT\_E\_CANT\_GENERATE", false)

FLAGNR(Number, GoptCleanupThreshold, "Number of instructions seen before we cleanup the value table", DEFAULT\_CONFIG\_GoptCleanupThreshold)

FLAGNR(Number, AsmGoptCleanupThreshold, "Number of instructions seen before we cleanup the value table", DEFAULT\_CONFIG\_AsmGoptCleanupThreshold)

FLAGNR(Boolean, HighPrecisionDate, "Enable sub-millisecond resolution in Javascript Date for benchmark timing", DEFAULT\_CONFIG\_HighPrecisionDate)

FLAGNR(Number, InlineCountMax , "Maximum count in bytecodes to inline in a given function", DEFAULT\_CONFIG\_InlineCountMax)

FLAGNR(Number, InlineThreshold , "Maximum size in bytecodes of an inline candidate", DEFAULT\_CONFIG\_InlineThreshold)

FLAGNR(Number, AggressiveInlineCountMax, "Maximum count in bytecodes to inline in a given function", DEFAULT\_CONFIG\_AggressiveInlineCountMax)

FLAGNR(Number, AggressiveInlineThreshold, "Maximum size in bytecodes of an inline candidate for aggressive inlining", DEFAULT\_CONFIG\_AggressiveInlineThreshold)

FLAGNR(Number, InlineThresholdAdjustCountInLargeFunction , "Adjustment in the maximum size in bytecodes of an inline candidate in a large function", DEFAULT\_CONFIG\_InlineThresholdAdjustCountInLargeFunction)

FLAGNR(Number, InlineThresholdAdjustCountInMediumSizedFunction , "Adjustment in the maximum size in bytecodes of an inline candidate in a medium sized function", DEFAULT\_CONFIG\_InlineThresholdAdjustCountInMediumSizedFunction)

FLAGNR(Number, InlineThresholdAdjustCountInSmallFunction , "Adjustment in the maximum size in bytecodes of an inline candidate in a small function", DEFAULT\_CONFIG\_InlineThresholdAdjustCountInSmallFunction)

FLAGNR(String, Interpret , "List of functions to interpret", nullptr)

FLAGNR(Phases, Instrument , "Instrument the generated code from the given phase", )

FLAGNR(Number, JitQueueThreshold , "Max number of work items/script context in the jit queue", DEFAULT\_CONFIG\_JitQueueThreshold)

#ifdef LEAK\_REPORT

FLAGNR(String, LeakReport , "File name for the leak report", nullptr)

#endif

FLAGNR(Number, LoopInlineThreshold , "Maximum size in bytecodes of an inline candidate with loops or not enough profile data", DEFAULT\_CONFIG\_LoopInlineThreshold)

FLAGNR(Number, LeafInlineThreshold , "Maximum size in bytecodes of an inline candidate with loops or not enough profile data", DEFAULT\_CONFIG\_LeafInlineThreshold)

FLAGNR(Number, ConstantArgumentInlineThreshold, "Maximum size in bytecodes of an inline candidate with constant argument and the argument being used for a branch", DEFAULT\_CONFIG\_ConstantArgumentInlineThreshold)

FLAGNR(Number, RecursiveInlineThreshold, "Maximum size in bytecodes of an inline candidate to inline recursively", DEFAULT\_CONFIG\_RecursiveInlineThreshold)

FLAGNR(Number, RecursiveInlineDepthMax, "Maximum depth of a recursive inline call", DEFAULT\_CONFIG\_RecursiveInlineDepthMax)

FLAGNR(Number, RecursiveInlineDepthMin, "Maximum depth of a recursive inline call", DEFAULT\_CONFIG\_RecursiveInlineDepthMin)

FLAGNR(Number, Loop , "Number of times to execute the script (useful for profiling short benchmarks and finding leaks)", DEFAULT\_CONFIG\_Loop)

FLAGRA(Number, LoopInterpretCount , lic, "Number of times loop has to be interpreted before JIT Loop body", DEFAULT\_CONFIG\_LoopInterpretCount)

FLAGNR(Number, LoopProfileIterations , "Number of iterations of a loop that must be profiled before jitting the loop body", DEFAULT\_CONFIG\_LoopProfileIterations)

FLAGNR(Number, OutsideLoopInlineThreshold , "Maximum size in bytecodes of an inline candidate outside a loop in inliner", DEFAULT\_CONFIG\_OutsideLoopInlineThreshold)

FLAGNR(Number, MaxFuncInlineDepth , "Number of times to allow inlining a function recursively, plus one (min: 1, max: 255)", DEFAULT\_CONFIG\_MaxFuncInlineDepth)

FLAGNR(Number, MaxNumberOfInlineesWithLoop, "Number of times to allow inlinees with a loop in a top function", DEFAULT\_CONFIG\_MaxNumberOfInlineesWithLoop)

#ifdef MEMSPECT\_TRACKING

FLAGNR(Phases, Memspect, "Enables memspect tracking to perform memory investigations.", )

#endif

FLAGNR(Number, PolymorphicInlineThreshold , "Maximum size in bytecodes of an polymorphic inline candidate", DEFAULT\_CONFIG\_PolymorphicInlineThreshold)

FLAGNR(Boolean, PrimeRecycler , "Prime the recycler first", DEFAULT\_CONFIG\_PrimeRecycler)

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

FLAGNR(Boolean, LeakStackTrace , "Include stack trace on leaked pinned object and heap objects", false)

FLAGNR(Boolean, ForceMemoryLeak , "Fake leak some memory to test leak report and check memory leak", false)

#endif

FLAGNR(Boolean, TDZ , "Emit temporal dead zone code for let/const", DEFAULT\_CONFIG\_TDZ)

FLAGNR(Boolean, ForceOldDateAPI , "Force Chakra to use old dates API regardless of availability of a new one", DEFAULT\_CONFIG\_ForceOldDateAPI)

FLAGNR(Number, JitLoopBodyHotLoopThreshold , "Number of times loop has to be iterated in jitloopbody before it is determined as hot", DEFAULT\_CONFIG\_JitLoopBodyHotLoopThreshold)

FLAGNR(Number, LoopBodySizeThresholdToDisableOpts, "Minimum bytecode size of a loop body, above which we might consider switching off optimizations in jit loop body to avoid rejits", DEFAULT\_CONFIG\_LoopBodySizeThresholdToDisableOpts)

FLAGNR(Number, MaxJitThreadCount , "Number of maximum allowed parallel jit threads (actual number is factor of number of processors and other heuristics)", DEFAULT\_CONFIG\_MaxJitThreadCount)

FLAGNR(Boolean, ForceMaxJitThreadCount, "Force the number of parallel jit threads as specified by MaxJitThreadCount flag (creation guaranteed)", DEFAULT\_CONFIG\_ForceMaxJitThreadCount)

FLAGNR(Number, MinInterpretCount , "Minimum number of times a function must be interpreted", 0)

FLAGNR(Number, MinSimpleJitRunCount , "Minimum number of times a function must be run in simple jit", 0)

FLAGNRA(Number, MaxInterpretCount , Mic, "Maximum number of times a function can be interpreted", 0)

FLAGNRA(Number, MaxSimpleJitRunCount , Msjrc, "Maximum number of times a function will be run in SimpleJitted code", 0)

FLAGNRA(Number, MinMemOpCount , Mmoc, "Minimum count of a loop to activate MemOp", DEFAULT\_CONFIG\_MinMemOpCount)

#if ENABLE\_COPYONACCESS\_ARRAY

FLAGNR(Number, MaxCopyOnAccessArrayLength, "Maximum length of copy-on-access array", DEFAULT\_CONFIG\_MaxCopyOnAccessArrayLength)

FLAGNR(Number, MinCopyOnAccessArrayLength, "Minimum length of copy-on-access array", DEFAULT\_CONFIG\_MinCopyOnAccessArrayLength)

FLAGNR(Number, CopyOnAccessArraySegmentCacheSize, "Size of copy-on-access array segment cache (1-32)", DEFAULT\_CONFIG\_CopyOnAccessArraySegmentCacheSize)

#endif

FLAGNR(Number, MinTemplatizedJitRunCount, "Minimum number of times a function must be Templatized Jitted", DEFAULT\_CONFIG\_MinTemplatizedJitRunCount)

FLAGNR(Number, MinAsmJsInterpreterRunCount, "Minimum number of times a function must be Asm Interpreted", DEFAULT\_CONFIG\_MinAsmJsInterpreterRunCount)

FLAGNR(Number, MinTemplatizedJitLoopRunCount, "Minimum LoopCount run of the Templatized Jit function to run FullJited", DEFAULT\_CONFIG\_MinTemplatizedJitLoopRunCount)

FLAGNRA(Number, MaxTemplatizedJitRunCount, Mtjrc, "Maximum number of times a function must be templatized jit", DEFAULT\_CONFIG\_MaxTemplatizedJitRunCount)

FLAGNRA(Number, MaxAsmJsInterpreterRunCount, Maic, "Maximum number of times a function must be interpreted in asmjs", DEFAULT\_CONFIG\_MaxAsmJsInterpreterRunCount)

FLAGR (Number, AutoProfilingInterpreter0Limit, "Limit after which to transition to the next execution mode", DEFAULT\_CONFIG\_AutoProfilingInterpreter0Limit)

FLAGR (Number, ProfilingInterpreter0Limit, "Limit after which to transition to the next execution mode", DEFAULT\_CONFIG\_ProfilingInterpreter0Limit)

FLAGR (Number, AutoProfilingInterpreter1Limit, "Limit after which to transition to the next execution mode", DEFAULT\_CONFIG\_AutoProfilingInterpreter1Limit)

FLAGR (Number, SimpleJitLimit, "Limit after which to transition to the next execution mode", DEFAULT\_CONFIG\_SimpleJitLimit)

FLAGR (Number, ProfilingInterpreter1Limit, "Limit after which to transition to the next execution mode", DEFAULT\_CONFIG\_ProfilingInterpreter1Limit)

FLAGNRA(String, ExecutionModeLimits, Eml, "Execution mode limits in th form: AutoProfilingInterpreter0.ProfilingInterpreter0.AutoProfilingInterpreter1.SimpleJit.ProfilingInterpreter1 - Example: -ExecutionModeLimits:12.4.0.132.12", L"")

FLAGRA(Boolean, EnforceExecutionModeLimits, Eeml, "Enforces the execution mode limits such that they are never exceeded.", false)

FLAGNRA(Number, SimpleJitAfter , Sja, "Number of calls to a function after which to simple-JIT the function", 0)

FLAGNRA(Number, FullJitAfter , Fja, "Number of calls to a function after which to full-JIT the function. The function will be profiled for every iteration.", 0)

FLAGNR(Boolean, NewSimpleJit , "Uses the new simple JIT", DEFAULT\_CONFIG\_NewSimpleJit)

FLAGNR(Number, MaxLinearIntCaseCount , "Maximum number of cases(in switch statement) for which instructions can be generated linearly",DEFAULT\_CONFIG\_MaxLinearIntCaseCount)

FLAGNR(Number, MaxSingleCharStrJumpTableSize, "Maximum single char string jump table size", DEFAULT\_CONFIG\_MaxSingleCharStrJumpTableSize)

FLAGNR(Number, MaxSingleCharStrJumpTableRatio, "Maximum single char string jump table size as multiples of the actual case arm", DEFAULT\_CONFIG\_MaxSingleCharStrJumpTableRatio)

FLAGNR(Number, MinSwitchJumpTableSize , "Minimum size of the jump table, that is created for consecutive integer case arms in a Switch Statement",DEFAULT\_CONFIG\_MinSwitchJumpTableSize)

FLAGNR(Number, MaxLinearStringCaseCount, "Maximum number of string cases(in switch statement) for which instructions can be generated linearly",DEFAULT\_CONFIG\_MaxLinearStringCaseCount)

FLAGR(Number, MinDeferredFuncTokenCount, "Minimum length in tokens of defer-parsed function", DEFAULT\_CONFIG\_MinDeferredFuncTokenCount)

#if DBG

FLAGNR(Number, SkipFuncCountForBailOnNoProfile, "Initial Number of functions in a func body to be skipped from forcibly inserting BailOnNoProfile.", DEFAULT\_CONFIG\_SkipFuncCountForBailOnNoProfile)

#endif

FLAGNR(Number, MaxJITFunctionBytecodeSize, "The biggest function we'll JIT (bytecode size)", DEFAULT\_CONFIG\_MaxJITFunctionBytecodeSize)

FLAGNR(Number, MaxLoopsPerFunction , "Maximum number of loops in any function in the script", DEFAULT\_CONFIG\_MaxLoopsPerFunction)

FLAGNR(Number, FuncObjectInlineCacheThreshold , "Maximum number of inline caches a function body may have to allow for inline caches to be allocated on the function object", DEFAULT\_CONFIG\_FuncObjectInlineCacheThreshold)

FLAGNR(Boolean, NoDeferParse , "Disable deferred parsing", false)

FLAGNR(Boolean, NoLogo , "No logo, which we don't display anyways", false)

#ifdef \_ARM64\_

FLAGR (Boolean, NoNative , "Disable native codegen", true)

#else

FLAGR (Boolean, NoNative , "Disable native codegen", false)

#endif

FLAGNR(Number, NopFrequency , "Frequency of NOPs inserted by NOP insertion phase. A NOP is guaranteed to be inserted within a range of (1<<n) instrs (default=8)", DEFAULT\_CONFIG\_NopFrequency)

FLAGNR(Boolean, NoStrictMode , "Disable strict mode checks on all functions", false)

FLAGNR(Boolean, NormalizeStats , "When dumping stats, do some normalization (used with -instrument:linearscan)", false)

#ifdef ENABLE\_PROJECTION

FLAGNR(Boolean, NoWinRTFastSig , "Disable fast call for common WinRT function signatures", false)

#endif

FLAGNR(Phases, Off , "Turn off specific phases or feature.(Might not work for all phases)", )

FLAGNR(Phases, OffProfiledByteCode , "Turn off specific byte code for phases or feature.(Might not work for all phases)", )

FLAGNR(Phases, On , "Turn on specific phases or feature.(Might not work for all phases)", )

FLAGNR(String, OutputFile , "Log the output to a specified file. Default: output.log in the working directory.", L"output.log")

FLAGNR(String, OutputFileOpenMode , "File open mode for OutputFile. Default: wt, specify 'at' for append", L"wt")

#ifdef ENABLE\_TRACE

FLAGNR(Boolean, InMemoryTrace , "Enable in-memory trace (investigate crash using trace in dump file). Use !jd.dumptrace to print it.", DEFAULT\_CONFIG\_InMemoryTrace)

FLAGNR(Number, InMemoryTraceBufferSize, "The size of circular buffer for in-memory trace (the units used is: number of trace calls). ", DEFAULT\_CONFIG\_InMemoryTraceBufferSize)

FLAGNR(Boolean, RichTraceFormat, "Whether to use extra data in Output/Trace header.", DEFAULT\_CONFIG\_RichTraceFormat)

#ifdef STACK\_BACK\_TRACE

FLAGNR(Boolean, TraceWithStack, "Whether the trace need to include stack trace (for each trace entry).", DEFAULT\_CONFIG\_TraceWithStack)

#endif STACK\_BACK\_TRACE

#endif ENABLE\_TRACE

FLAGNR(Boolean, PrintRunTimeDataCollectionTrace, "Print traces needed for runtime data collection", false)

#ifdef ENABLE\_PREJIT

FLAGR (Boolean, Prejit , "Prejit everything, including things that are not called, ignoring limits (default: false)", DEFAULT\_CONFIG\_Prejit)

#endif

FLAGNR(Boolean, PrintSrcInDump , "Print the lineno and the source code in the intermediate dumps", true)

#if PROFILE\_DICTIONARY

FLAGNR(Number, ProfileDictionary , "Profile dictionary usage. Only dictionaries with max depth of <number> or above are displayed (0=no filter).", -1)

#endif

#ifdef PROFILE\_EXEC

FLAGNR(Phases, Profile , "Profile the given phase", )

FLAGNR(Number, ProfileThreshold , "A phase is displayed in the profiler report only if its contribution is more than this threshold", 0)

#endif

#ifdef PROFILE\_OBJECT\_LITERALS

FLAGNR(Boolean, ProfileObjectLiteral , "Profile Object literal usage", false)

#endif

#ifdef PROFILE\_MEM

FLAGNR(String, ProfileMemory , "Profile memory usage", )

#endif

#ifdef PROFILE\_STRINGS

FLAGNR(Boolean, ProfileStrings , "Profile string statistics", false)

#endif

#ifdef PROFILE\_TYPES

FLAGNR(Boolean, ProfileTypes , "Profile type statistics", false)

#endif

#ifdef PROFILE\_EVALMAP

FLAGNR(Boolean, ProfileEvalMap , "Profile eval map statistics", false)

#endif

#ifdef PROFILE\_BAILOUT\_RECORD\_MEMORY

FLAGNR(Boolean, ProfileBailOutRecordMemory, "Profile bailout record memory statistics", false)

#endif

FLAGNR(Number, RejitMaxBailOutCount, "Maximum number of bailouts for a bailout record after which rejit is forced", DEFAULT\_CONFIG\_RejitMaxBailOutCount)

FLAGNR(Number, RejitRatioLimit, "Rejit ratio (Percentage of bailouts per function after which rejit is queued)", DEFAULT\_CONFIG\_RejitRatioLimit)

FLAGNR(Number, MinBailOutsBeforeRejit, "Minimum number of bailouts for a single bailout record after which a rejit is considered", DEFAULT\_CONFIG\_MinBailOutsBeforeRejit)

FLAGNR(Boolean, LibraryStackFrame , "Display library stack frame", DEFAULT\_CONFIG\_LibraryStackFrame)

FLAGNR(Boolean, LibraryStackFrameDebugger , "Assume debugger support for library stack frame", DEFAULT\_CONFIG\_LibraryStackFrameDebugger)

#ifdef RECYCLER\_STRESS

FLAGNR(Boolean, RecyclerStress , "Stress the recycler by collect on every allocation call", false)

#ifdef CONCURRENT\_GC\_ENABLED

FLAGNR(Boolean, RecyclerBackgroundStress , "Stress the recycler by collect in the background thread on every allocation call", false)

FLAGNR(Boolean, RecyclerConcurrentStress , "Stress the concurrent recycler by concurrent collect on every allocation call", false)

FLAGNR(Boolean, RecyclerConcurrentRepeatStress , "Stress the concurrent recycler by concurrent collect on every allocation call and repeat mark and rescan in the background thread", false)

#endif

#ifdef PARTIAL\_GC\_ENABLED

FLAGNR(Boolean, RecyclerPartialStress , "Stress the partial recycler by partial collect on every allocation call", false)

#endif

FLAGNR(Boolean, RecyclerTrackStress, "Stress tracked object handling by simulating tracked objects for regular allocations", false)

FLAGNR(Boolean, RecyclerInduceFalsePositives, "Stress recycler by forcing false positive object marks", false)

#endif // RECYCLER\_STRESS

FLAGNR(Boolean, RecyclerForceMarkInterior, "Force all the mark as interior", DEFAULT\_CONFIG\_RecyclerForceMarkInterior)

#ifdef CONCURRENT\_GC\_ENABLED

FLAGNR(Number, RecyclerPriorityBoostTimeout, "Adjust priority boost timeout", 5000)

FLAGNR(Number, RecyclerThreadCollectTimeout, "Adjust thread collect timeout", 1000)

#endif

#ifdef RECYCLER\_PAGE\_HEAP

FLAGNR(Number, PageHeap, "Use full page for heap allocations", DEFAULT\_CONFIG\_PageHeap)

FLAGNR(Boolean, PageHeapAllocStack, "Capture alloc stack under page heap mode", DEFAULT\_CONFIG\_PageHeapAllocStack)

FLAGNR(Boolean, PageHeapFreeStack, "Capture free stack under page heap mode", DEFAULT\_CONFIG\_PageHeapFreeStack)

FLAGNR(NumberRange, PageHeapBucketNumber, "Bucket numbers to be used for page heap allocations", )

FLAGNR(Number, PageHeapBlockType, "Type of blocks to use page heap for", DEFAULT\_CONFIG\_PageHeapBlockType)

#endif

#ifdef RECYCLER\_NO\_PAGE\_REUSE

FLAGNR(Boolean, RecyclerNoPageReuse, "Do not reuse page in recycler", false)

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

FLAGNR(Phases, RecyclerVerify , "Verify recycler memory", )

FLAGNR(Number, RecyclerVerifyPadSize , "Padding size to verify recycler memory", 12)

#endif

FLAGNR(Boolean, RecyclerTest , "Run recycler tests instead of executing script", false)

FLAGNR(Boolean, RecyclerProtectPagesOnRescan, "Temporarily switch all pages to read only during rescan", false)

#ifdef RECYCLER\_VERIFY\_MARK

FLAGNR(Boolean, RecyclerVerifyMark , "verify concurrent gc", false)

#endif

FLAGR (Number, LowMemoryCap , "Memory cap indicating a low-memory process", DEFAULT\_CONFIG\_LowMemoryCap)

#ifdef RUNTIME\_DATA\_COLLECTION

FLAGNR(String, RuntimeDataOutputFile, "Filename to write the dynamic profile info", nullptr)

#endif

FLAGNR(Boolean, ReportErrors , "Enable reporting of syntax errors", false)

FLAGR (Number, SpeculationCap , "How much bytecode we'll speculatively JIT", DEFAULT\_CONFIG\_SpeculationCap)

#if DBG\_DUMP || defined(BGJIT\_STATS) || defined(RECYCLER\_STATS)

FLAGNR(Phases, Stats , "Stats the given phase", )

#endif

#if EXCEPTION\_RECOVERY

FLAGNR(Boolean, SwallowExceptions , "Force a try/catch around every statement", false)

#endif

FLAGNR(Boolean, PrintSystemException , "Always print a message when there's OOM or OOS", false)

FLAGNR(Number, SwitchOptHolesThreshold, "Maximum percentage of holes (missing case values in a switch statement) with which a jump table can be created",DEFAULT\_CONFIG\_SwitchOptHolesThreshold)

FLAGR (Number, TempMin , "Temp number switch which code can temporarily use for debugging", DEFAULT\_CONFIG\_TempMin)

FLAGR (Number, TempMax , "Temp number switch which code can temporarily use for debugging", DEFAULT\_CONFIG\_TempMax)

FLAGNR(Phases, Trace , "Trace the given phase", )

#if defined(\_M\_IX86) || defined(\_M\_X64)

FLAGR(Number, LoopAlignNopLimit , "Max number of nops for loop alignment", DEFAULT\_CONFIG\_LoopAlignNopLimit)

#endif

#ifdef PROFILE\_MEM

FLAGNR(Phases, TraceMemory , "Trace memory usage", )

#endif

#if DBG\_DUMP || defined(RECYCLER\_TRACE)

//TraceMetaDataParsing flag with optional levels:

// Level 1 = interfaces only

// Level 2 = interfaces and methods

// Level 3 = interfaces, methods and parameters

// Level 4 = interfaces and properties

// Level 5 (default) = ALL

FLAGNR(Number, TraceMetaDataParsing , "Trace metadata parsing for generating JS projections. [Levels 1-5, with 5 corresponding to most detailed]", 5)

FLAGNR(Boolean, TraceWin8Allocations , "Trace the win8 memory allocations", false)

FLAGNR(Boolean, TraceWin8DeallocationsImmediate , "Trace the win8 memory deallocations immediately", false)

FLAGNR(Boolean, PrintWin8StatsDetailed , "Print the detailed memory trace report", false)

FLAGNR(Boolean, TraceProtectPages , "Trace calls to protecting pages of custom heap allocated pages", false)

//TraceProjection flag with optional levels:

// Level 1 = error

// Level 2 = warning

// Level 3 = informational

FLAGNR(Number, TraceProjection , "Trace projection related activities, [Levels 1-3, with 3 corresponding to most detailed]", 3)

#endif

FLAGNR(Boolean, TraceAsyncDebugCalls , "Trace calls to async debugging API (default: false)", DEFAULT\_CONFIG\_TraceAsyncDebugCalls)

#ifdef TRACK\_DISPATCH

FLAGNR(Boolean, TrackDispatch , "Save stack traces of where JavascriptDispatch/HostVariant are created", false)

#endif

FLAGNR(Boolean, Verbose , "Dump details", DEFAULT\_CONFIG\_Verbose)

FLAGNR(Boolean, UseFullName , "Enable fully qualified name", DEFAULT\_CONFIG\_UseFullName)

FLAGNR(Boolean, UseFunctionIdForTrace , "Use function id instead of function number for trace output", false)

FLAGNR(Boolean, Utf8 , "Use UTF8 for file output", false)

FLAGR (Number, Version , "Version in which to run the jscript engine. [one of 1,2,3,4,5,6]. Default is latest for jc/jshost, 1 for IE", 6 )

#ifdef ENABLE\_PROJECTION

FLAGR (Number, HostType , "Host type in which to run the jscript engine. [one of 1,2]. Default is 1 = Browser.", 1)

#endif

FLAGR (Boolean, WERExceptionSupport , "WER feature for extended exception support. Enabled when WinRT is enabled", false )

#ifdef ENABLE\_PROJECTION

FLAGR (Boolean, WinRTConstructorAllowed, "Whether WinRT constructors is allowed in WebView host type. Constructor is always allowed in other host type ", false)

#endif

FLAGNR(Boolean, ExtendedErrorStackForTestHost, "Enable passing extended error stack string to test host.", DEFAULT\_CONFIG\_ExtendedErrorStackForTestHost)

FLAGNR(Boolean, errorStackTrace , "error.StackTrace feature. Remove when feature complete", DEFAULT\_CONFIG\_errorStackTrace)

FLAGNR(Boolean, DoHeapEnumOnEngineShutdown, "Perform a heap enumeration whenever shut a script engine down", false)

#ifdef HEAP\_ENUMERATION\_VALIDATION

FLAGNR(Boolean, ValidateHeapEnum , "Validate that heap enumeration is reporting all Js::RecyclableObjects in the heap", false)

#endif

#if ENABLE\_REGEX\_CONFIG\_OPTIONS

//

// Regex flags

//

FLAGR (Boolean, RegexTracing , "Trace all Regex invocations to the output.", DEFAULT\_CONFIG\_RegexTracing)

FLAGR (Boolean, RegexProfile , "Collect usage statistics on all Regex invocations.", DEFAULT\_CONFIG\_RegexProfile)

FLAGR (Boolean, RegexDebug , "Trace compilation of UnifiedRegex expressions.", DEFAULT\_CONFIG\_RegexDebug)

FLAGR (Boolean, RegexOptimize , "Optimize regular expressions in the unified Regex system (default: true)", DEFAULT\_CONFIG\_RegexOptimize)

FLAGR (Number, DynamicRegexMruListSize, "Size of the MRU list for dynamic regexes", DEFAULT\_CONFIG\_DynamicRegexMruListSize)

#endif

FLAGR (Boolean, OptimizeForManyInstances, "Optimize script engine for many instances (low memory footprint per engine, assume low spare CPU cycles) (default: false)", DEFAULT\_CONFIG\_OptimizeForManyInstances)

FLAGNR(Phases, TestTrace , "Test trace for the given phase", )

FLAGNR(Boolean, EnableEvalMapCleanup, "Enable cleaning up the eval map", true)

#ifdef PROFILE\_MEM

FLAGNR(Boolean, TraceObjectAllocation, "Enable cleaning up the eval map", false)

#endif

#ifdef ENABLE\_PROJECTION

FLAGNR(Boolean, EnableThirdPartyGCPressure, "Enable use of GCPressure attribute value on 3rd party WinRT objects (not in Windows namespace) (default: false)", false)

FLAGNR(Number, TargetWinRTVersion, "Specifies WinRT version number to target. [one of 0,1,2,3,4]. Default is 1 = NTDDI\_WIN8", DEFAULT\_CONFIG\_TargetWinRTVersion)

FLAGNR(Boolean, EnableVersioningAllAssemblies, "Enable versioning behavior for all assemblies, regardless of host flag (default: false)", false)

FLAGR(Boolean, FailFastIfDisconnectedDelegate, "When set fail fast if disconnected delegate is invoked", DEFAULT\_CONFIG\_FailFastIfDisconnectedDelegate)

#endif

FLAGNR(Number, Sse, "Virtually disables SSE-based optimizations above the specified SSE level in the Chakra JIT (does not affect CRT SSE usage)", DEFAULT\_CONFIG\_Sse)

FLAGNR(Number, DeletedPropertyReuseThreshold, "Start reusing deleted property indexes after this many properties are deleted. Zero to disable reuse.", DEFAULT\_CONFIG\_DeletedPropertyReuseThreshold)

FLAGNR(Boolean, ForceStringKeyedSimpleDictionaryTypeHandler, "Force switch to string keyed version of SimpleDictionaryTypeHandler on first new property added to a SimpleDictionaryTypeHandler", DEFAULT\_CONFIG\_ForceStringKeyedSimpleDictionaryTypeHandler)

FLAGNR(Number, BigDictionaryTypeHandlerThreshold, "Min Slot Capacity required to convert DictionaryTypeHandler to BigDictionaryTypeHandler.(Advisable to give more than 15 - to avoid false positive cases)", DEFAULT\_CONFIG\_BigDictionaryTypeHandlerThreshold)

FLAGNR(Boolean, TypeSnapshotEnumeration, "Create a true snapshot of the type of an object before enumeration and enumerate only those properties.", DEFAULT\_CONFIG\_TypeSnapshotEnumeration)

FLAGR (Boolean, EnumerationCompat, "When set in IE10 mode, restores enumeration behavior to RC behavior", DEFAULT\_CONFIG\_EnumerationCompat)

FLAGNR(Boolean, IsolatePrototypes, "Should prototypes get unique types not shared with other objects (default: true)?", DEFAULT\_CONFIG\_IsolatePrototypes)

FLAGNR(Boolean, ChangeTypeOnProto, "When becoming a prototype should the object switch to a new type (default: true)?", DEFAULT\_CONFIG\_ChangeTypeOnProto)

FLAGNR(Boolean, ShareInlineCaches, "Determines whether inline caches are shared between all loads (or all stores) of the same property ID", DEFAULT\_CONFIG\_ShareInlineCaches)

FLAGNR(Boolean, DisableDebugObject, "Disable test only Debug object properties", DEFAULT\_CONFIG\_DisableDebugObject)

FLAGNR(Boolean, DumpHeap, "enable Debug.dumpHeap even when DisableDebugObject is set", DEFAULT\_CONFIG\_DumpHeap)

FLAGNR(String, autoProxy, "enable creating proxy for each object creation", L"\_\_msTestHandler")

FLAGNR(Number, PerfHintLevel, "Specifies the perf-hint level (1,2) 1 == critical, 2 == only noisy", DEFAULT\_CONFIG\_PerfHintLevel)

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

FLAGNR(Boolean, MemProtectHeap, "Use the mem protect heap as the default heap", DEFAULT\_CONFIG\_MemProtectHeap)

#endif

#ifdef RECYCLER\_STRESS

FLAGNR(Boolean, MemProtectHeapStress, "Stress the recycler by collect on every allocation call", false)

#ifdef CONCURRENT\_GC\_ENABLED

FLAGNR(Boolean, MemProtectHeapBackgroundStress, "Stress the recycler by collect in the background thread on every allocation call", false)

FLAGNR(Boolean, MemProtectHeapConcurrentStress, "Stress the concurrent recycler by concurrent collect on every allocation call", false)

FLAGNR(Boolean, MemProtectHeapConcurrentRepeatStress, "Stress the concurrent recycler by concurrent collect on every allocation call and repeat mark and rescan in the background thread", false)

#endif

#ifdef PARTIAL\_GC\_ENABLED

FLAGNR(Boolean, MemProtectHeapPartialStress, "Stress the partial recycler by partial collect on every allocation call", false)

#endif

#endif

#ifdef SUPPORT\_FIXED\_FIELDS\_ON\_PATH\_TYPES

FLAGNR(Boolean, FixPropsOnPathTypes, "Mark properties as fixed on path types (default: false).", DEFAULT\_CONFIG\_FixPropsOnPathTypes)

#endif

FLAGNR(NumberSet, BailoutTraceFilter, "Filter the bailout trace messages to specific bailout kinds.", )

FLAGNR(NumberSet, RejitTraceFilter, "Filter the rejit trace messages to specific bailout kinds.", )

// recycler heuristic flags

FLAGNR(Number, MaxBackgroundFinishMarkCount, "Maximum number of background finish mark", 1)

FLAGNR(Number, BackgroundFinishMarkWaitTime, "Millisecond to wait for background finish mark", 15)

FLAGNR(Number, MinBackgroundRepeatMarkRescanBytes, "Minimum number of bytes rescan to trigger background finish mark", -1)

// recycler memory restrict test flags

FLAGNR(Number, MaxMarkStackPageCount , "Restrict recycler mark stack size (in pages)", -1)

FLAGNR(Number, MaxTrackedObjectListCount, "Restrict recycler tracked object count during GC", -1)

#if DBG

FLAGNR(Boolean, InitializeInterpreterSlotsWithInvalidStackVar, "Enable the initialization of the interpreter local slots with invalid stack vars", false)

#endif

FLAGNR(Boolean, ClearInlineCachesOnCollect, "Clear all inline caches on every garbage collection", false)

FLAGNR(Number, InlineCacheInvalidationListCompactionThreshold, "Compact inline cache invalidation lists if their utilization falls below this threshold", DEFAULT\_CONFIG\_InlineCacheInvalidationListCompactionThreshold)

#ifdef IR\_VIEWER

FLAGNR(Boolean, IRViewer, "Enable IRViewer functionality (improved UI for various stages of IR generation)", false)

#endif /\* IR\_VIEWER \*/

FLAGNR(Boolean, InvalidateSolutionContextsForGetStructure, "To reduce memory consumption, in the end of GetStructure call, invalidate script contexts used only for GetStructure -- this would invalidate ones associated with solution files (not top-most references such as helpers.js)", DEFAULT\_CONFIG\_InvalidateSolutionContextsForGetStructure)

FLAGNR(Boolean, GCPauseTel, "Enable GC Pause telemetry in the product code.", false)

FLAGNR(Boolean, ES5LangTel, "Print ES5 language telemetry output.", false)

FLAGNR(Boolean, ES6LangTel, "Print ES6 language telemetry output.", false)

FLAGNR(Boolean, ESBLangTel, "Print ES built-ins telemetry output.", false)

FLAGNR(Boolean, DateParseTel, "Print Date.parse telemetry output.", false)

FLAGNR(Number, GCMemoryThreshold, "Threshold for allocation-based GC initiation (in MB)", 0)

#ifdef \_CONTROL\_FLOW\_GUARD

FLAGNR(Boolean, PreReservedHeapAlloc, "Enable Pre-reserved Heap Page Allocator", true)

FLAGNR(Boolean, CFGInJit, "Enable CFG check in JIT", true)

FLAGNR(Boolean, CFG, "Force enable CFG on jshost. version in the jshost's manifest file disables CFG", true)

#endif

#if DBG

FLAGNR(Number, SimulatePolyCacheWithOneTypeForInlineCacheIndex, "Use with SimulatePolyCacheWithOneTypeForFunction to simulate creating a polymorphic inline cache containing only one type due to a collision, for testing ObjTypeSpec", -1)

#endif

#undef FLAG\_REGOVR\_EXP

#undef FLAG\_REGOVR\_ASMJS

#undef FLAG

#undef FLAGP

#undef FLAGRA

#undef FLAGNR

#undef FLAGNRA

#undef FLAGPNR

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "core\ConfigParser.h"

// Include this file got get the default behavior for JsUtil::ExternalApi functions.

void JsUtil::ExternalApi::RecoverUnusedMemory()

{

}

bool JsUtil::ExternalApi::RaiseOutOfMemoryIfScriptActive()

{

return false;

}

bool JsUtil::ExternalApi::RaiseOnIntOverflow()

{

return false;

}

LPWSTR JsUtil::ExternalApi::GetFeatureKeyName()

{

return L"";

}

bool JsUtil::ExternalApi::RaiseStackOverflowIfScriptActive(Js::ScriptContext \* scriptContext, PVOID returnAddress)

{

return false;

}

ThreadContextId JsUtil::ExternalApi::GetCurrentThreadContextId()

{

return (ThreadContextId)::GetCurrentThreadId();

}

#if DBG || defined(EXCEPTION\_CHECK)

BOOL JsUtil::ExternalApi::IsScriptActiveOnCurrentThreadContext()

{

return false;

}

#endif

bool ConfigParserAPI::FillConsoleTitle(\_\_ecount(cchBufferSize) LPWSTR buffer, size\_t cchBufferSize, \_\_in LPWSTR moduleName)

{

return false;

}

void ConfigParserAPI::DisplayInitialOutput(\_\_in LPWSTR moduleName)

{

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#if PROFILE\_DICTIONARY

class DictionaryStats;

struct DictionaryType

{

int instancesCount;

DictionaryType\* pNext;

char name[256];

DictionaryStats\* instances;

};

class DictionaryStats

{

public:

static DictionaryStats\* Create(const char\* name, uint initialSize);

static void OutputStats();

private:

static void ComputeStats(uint input, double &total, double &max);

static void ComputeStats(double input, double &total, double &max);

static void ClearStats();

static DictionaryType\* dictionaryTypes;

static CRITICAL\_SECTION dictionaryTypesCriticalSection;

public:

void Resize(uint newSize, uint emptyBucketCount);

void Insert(uint depth);

void Remove(bool isBucketEmpty);

void Lookup(uint depth);

DictionaryStats\* Clone();

DictionaryStats\* pNext;

private:

DictionaryStats(const char\* name, uint initialSize);

uint initialSize;

uint finalSize;

uint countOfEmptyBuckets;

uint countOfResize;

uint itemCount;

uint maxDepth;

uint lookupCount;

uint collisionCount;

uint lookupDepthTotal;

uint maxLookupDepth;

char\* pName;

};

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// This macro defines some global operators, and hence must be used at global scope

#define ENUM\_CLASS\_HELPERS(TEnum, TUnderlying) \

inline TEnum operator +(const TEnum e, const TUnderlying n) \

{ \

return static\_cast<TEnum>(static\_cast<TUnderlying>(e) + n); \

} \

\

inline TEnum operator +(const TUnderlying n, const TEnum e) \

{ \

return static\_cast<TEnum>(n + static\_cast<TUnderlying>(e)); \

} \

\

inline TEnum operator +(const TEnum e0, const TEnum e1) \

{ \

return static\_cast<TUnderlying>(e0) + e1; \

} \

\

inline TEnum &operator +=(TEnum &e, const TUnderlying n) \

{ \

return e = e + n; \

} \

\

inline TEnum &operator ++(TEnum &e) \

{ \

return e += 1; \

} \

\

inline TEnum operator ++(TEnum &e, const int) \

{ \

const TEnum old = e; \

++e; \

return old; \

} \

\

inline TEnum operator -(const TEnum e, const TUnderlying n) \

{ \

return static\_cast<TEnum>(static\_cast<TUnderlying>(e) - n); \

} \

\

inline TEnum operator -(const TUnderlying n, const TEnum e) \

{ \

return static\_cast<TEnum>(n - static\_cast<TUnderlying>(e)); \

} \

\

inline TEnum operator -(const TEnum e0, const TEnum e1) \

{ \

return static\_cast<TUnderlying>(e0) - e1; \

} \

\

inline TEnum &operator -=(TEnum &e, const TUnderlying n) \

{ \

return e = e - n; \

} \

\

inline TEnum &operator --(TEnum &e) \

{ \

return e -= 1; \

} \

\

inline TEnum operator --(TEnum &e, const int) \

{ \

const TEnum old = e; \

--e; \

return old; \

} \

\

inline TEnum operator &(const TEnum e0, const TEnum e1) \

{ \

return static\_cast<TEnum>(static\_cast<TUnderlying>(e0) & static\_cast<TUnderlying>(e1)); \

} \

\

inline TEnum &operator &=(TEnum &e0, const TEnum e1) \

{ \

return e0 = e0 & e1; \

} \

\

inline TEnum operator ^(const TEnum e0, const TEnum e1) \

{ \

return static\_cast<TEnum>(static\_cast<TUnderlying>(e0) ^ static\_cast<TUnderlying>(e1)); \

} \

\

inline TEnum &operator ^=(TEnum &e0, const TEnum e1) \

{ \

return e0 = e0 ^ e1; \

} \

\

inline TEnum operator |(const TEnum e0, const TEnum e1) \

{ \

return static\_cast<TEnum>(static\_cast<TUnderlying>(e0) | static\_cast<TUnderlying>(e1)); \

} \

\

inline TEnum &operator |=(TEnum &e0, const TEnum e1) \

{ \

return e0 = e0 | e1; \

} \

\

inline TEnum operator <<(const TEnum e, const TUnderlying n) \

{ \

return static\_cast<TEnum>(static\_cast<TUnderlying>(e) << n); \

} \

\

inline TEnum &operator <<=(TEnum &e, const TUnderlying n) \

{ \

return e = e << n; \

} \

\

inline TEnum operator >>(const TEnum e, const TUnderlying n) \

{ \

return static\_cast<TEnum>(static\_cast<TUnderlying>(e) >> n); \

} \

\

inline TEnum &operator >>=(TEnum &e, const TUnderlying n) \

{ \

return e = e >> n; \

} \

\

inline TEnum operator ~(const TEnum e) \

{ \

return static\_cast<TEnum>(~static\_cast<TUnderlying>(e)); \

} \

\

inline TEnum operator +(const TEnum e) \

{ \

return e; \

} \

\

inline TEnum operator -(const TEnum e) \

{ \

return static\_cast<TEnum>(static\_cast<TUnderlying>(0) - static\_cast<TUnderlying>(e)); \

} \

\

inline bool operator !(const TEnum e) \

{ \

return !static\_cast<TUnderlying>(e); \

} \

// For private enum classes defined inside other classes, this macro can be used inside the class to declare friends

#define ENUM\_CLASS\_HELPER\_FRIENDS(TEnum, TUnderlying) \

friend TEnum operator +(const TEnum e, const TUnderlying n); \

friend TEnum operator +(const TUnderlying n, const TEnum e); \

friend TEnum operator +(const TEnum e0, const TEnum e1); \

friend TEnum &operator +=(TEnum &e, const TUnderlying n); \

friend TEnum &operator ++(TEnum &e); \

friend TEnum operator ++(TEnum &e, const int); \

friend TEnum operator -(const TEnum e, const TUnderlying n); \

friend TEnum operator -(const TUnderlying n, const TEnum e); \

friend TEnum operator -(const TEnum e0, const TEnum e1); \

friend TEnum &operator -=(TEnum &e, const TUnderlying n); \

friend TEnum &operator --(TEnum &e); \

friend TEnum operator --(TEnum &e, const int); \

friend TEnum operator &(const TEnum e0, const TEnum e1); \

friend TEnum &operator &=(TEnum &e0, const TEnum e1); \

friend TEnum operator ^(const TEnum e0, const TEnum e1); \

friend TEnum &operator ^=(TEnum &e0, const TEnum e1); \

friend TEnum operator |(const TEnum e0, const TEnum e1); \

friend TEnum &operator |=(TEnum &e0, const TEnum e1); \

friend TEnum operator <<(const TEnum e, const TUnderlying n); \

friend TEnum &operator <<=(TEnum &e, const TUnderlying n); \

friend TEnum operator >>(const TEnum e, const TUnderlying n); \

friend TEnum &operator >>=(TEnum &e, const TUnderlying n); \

friend TEnum operator ~(const TEnum e); \

friend TEnum operator +(const TEnum e); \

friend TEnum operator -(const TEnum e); \

friend bool operator !(const TEnum e);

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// macro BEGIN\_ENUM

///

/// BEGIN\_ENUM is used to create a C++ 'struct' type around an enum, providing

/// a scope for the enum instead of being defined at the global namespace.

/// Combined with the helper macros of BEGIN\_ENUM\_BYTE(), ..., this enforces

/// that the enum will only be given the intended allocated storage instead of

/// the default storage of a full 4-byte 'int'.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

#define BEGIN\_ENUM(name, storage) \

struct name \

{ \

enum \_E; \

\

inline name() \

{ \

\_value = (storage) 0; \

} \

\

inline name(\_E src) \

{ \

\_value = (storage) src; \

} \

\

inline name(storage n) \

{ \

\_value = n; \

} \

\

inline name(int n) \

{ \

/\* \*/ \

/\* This is needed to enable operations such as "m\_value &= ~Flags::Member; \*/ \

/\* \*/ \

\

\_value = (storage) n; \

AssertMsg(((int) \_value) == n, "Ensure no truncation"); \

} \

\

inline void operator =(\_E e) \

{ \

\_value = (storage) e; \

} \

\

inline void operator =(storage n) \

{ \

\_value = n; \

} \

\

inline bool operator ==(\_E e) const \

{ \

return ((\_E) \_value) == e; \

} \

\

inline bool operator !=(\_E e) const \

{ \

return ((\_E) \_value) != e; \

} \

\

inline bool operator <(\_E e) const \

{ \

return ((\_E) \_value) < e; \

} \

\

inline bool operator <=(\_E e) const \

{ \

return ((\_E) \_value) <= e; \

} \

\

inline bool operator >(\_E e) const \

{ \

return ((\_E) \_value) > e; \

} \

\

inline bool operator >=(\_E e) const \

{ \

return ((\_E) \_value) >= e; \

} \

\

inline \_E operator &(\_E e) const \

{ \

return (\_E) (((\_E) \_value) & e); \

} \

\

inline \_E operator |(name e) const \

{ \

return (\_E) (\_value | e.\_value); \

} \

\

inline void operator |=(name e) \

{ \

\_value = \_value | e.\_value; \

} \

\

inline void operator &=(name e) \

{ \

\_value = \_value & e.\_value; \

} \

\

inline void operator &=(\_E e) \

{ \

\_value = \_value & ((storage) e); \

} \

\

inline operator \_E() const \

{ \

return (\_E) \_value; \

} \

\

enum \_E \

{ \

#define BEGIN\_ENUM\_BYTE(name) BEGIN\_ENUM(name, byte)

#define BEGIN\_ENUM\_USHORT(name) BEGIN\_ENUM(name, uint16)

#define BEGIN\_ENUM\_UINT(name) BEGIN\_ENUM(name, uint32)

#define END\_ENUM\_BYTE() \

Force8BitPadding = (byte) 0xffU \

}; \

\

byte \_value; \

}; \

#define END\_ENUM\_USHORT() \

Force16BitPadding = (uint16) 0xffffU \

}; \

\

uint16 \_value; \

}; \

#define END\_ENUM\_UINT() \

Force32BitPadding = (uint32) 0xffffffffU \

}; \

\

uint32 \_value; \

};

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// macro PREVENT\_ASSIGN

///

/// PREVENT\_ASSIGN is used within a C++ type definition to define and

/// explicitly hide the "operator =()" method, preventing it from accidentally

/// being called by the program. If these are not explicitly defined, the C++

/// compiler will implicitly define them, which usually leads to unintended

/// behavior.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

#define PREVENT\_ASSIGN(ClassName) \

private: \

ClassName & operator =(const ClassName & rhs);

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// macro PREVENT\_COPY

///

/// PREVENT\_COPY is used within a C++ type definition to define and explicitly

/// hide the "C++ copy constructor" and "operator =()" methods, preventing them

/// from accidentally being called by the program. If these are not explicitly

/// defined, the C++ compiler will implicitly define them, which usually leads

/// to unintended behavior.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

#define PREVENT\_COPYCONSTRUCT(ClassName) \

private: \

ClassName(const ClassName & copy);

#define PREVENT\_COPY(ClassName) \

PREVENT\_COPYCONSTRUCT(ClassName); \

PREVENT\_ASSIGN(ClassName);

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// macro DECLARE\_OBJECT

///

/// DECLARE\_OBJECT sets up a class that derives from RcObject, RecyclableObject or

/// ZnObject:

/// - Prevent "C++ copy constructor" and "operator =()" methods.

/// - Must be allocated on heap. Because the copy constructor is hidden, this

/// requires an empty default constructor to be declared.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

#define DECLARE\_OBJECT(ClassName) \

public: \

inline ClassName() { } \

private: \

ClassName(const ClassName & copy); \

ClassName & operator =(const ClassName & rhs);

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

//{{NO\_DEPENDENCIES}}

// Microsoft Visual C++ generated include file.

// Used by Common.rc

//

#define IDS\_STRING101 101

#define IDS\_STRING102 102

// Next default values for new objects

//

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NEXT\_RESOURCE\_VALUE 102

#define \_APS\_NEXT\_COMMAND\_VALUE 40001

#define \_APS\_NEXT\_CONTROL\_VALUE 1001

#define \_APS\_NEXT\_SYMED\_VALUE 101

#endif

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// The following macros define the minimum required platform. The minimum required platform

// is the earliest version of Windows, Internet Explorer etc. that has the necessary features to run

// your application. The macros work by enabling all features available on platform versions up to and

// including the version specified.

// Modify the following defines if you have to target a platform prior to the ones specified below.

// Refer to MSDN for the latest info on corresponding values for different platforms.

#ifndef \_WIN32\_WINNT // Specifies that the minimum required platform is Windows Vista.

#define \_WIN32\_WINNT 0x0501 // Change this to the appropriate value to target other versions of Windows.

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// -----------------------------------------------------------------------------------------------------------------------------

// Template parameter constraints

// See http://www.stroustrup.com/bs\_faq2.html#constraints

// -----------------------------------------------------------------------------------------------------------------------------

namespace TemplateParameter

{

template<class T, class Base>

class SameOrDerivedFrom

{

private:

static void Constrain(T \*const t)

{

#pragma warning(suppress: 4189) // C4189: local variable is initialized but not referenced

Base \*const b = t;

}

public:

SameOrDerivedFrom()

{

#pragma warning(suppress: 4189) // C4189: local variable is initialized but not referenced

void (\*const p)(T \*const t) = Constrain;

}

};

template<class T>

struct Box

{

};

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

//=============================

// Enabled Warnings

//=============================

#pragma warning(default:4242) // conversion possible loss of data

//=============================

// Disabled Warnings

//=============================

// Warnings that we don't care about

#pragma warning(disable: 4100) // unreferenced formal parameter

#pragma warning(disable: 4127) // constant expression for Trace/Assert

#pragma warning(disable: 4200) // nonstandard extension used: zero-sized array in struct/union

#pragma warning(disable: 4201) // nameless unions are part of C++

#pragma warning(disable: 4512) // private operator= are good to have

#pragma warning(disable: 4481) // allow use of abstract and override keywords

#pragma warning(disable: 4324) // structure was padded due to alignment specifier

// warnings caused by normal optimizations

#if DBG

#else // DBG

#pragma warning(disable: 4702) // unreachable code caused by optimizations

#pragma warning(disable: 4189) // initialized but unused variable

#pragma warning(disable: 4390) // empty controlled statement

#endif // DBG

// PREFAST warnings

#ifdef \_PREFAST\_

// Warnings that we don't care about

#pragma warning(disable:6322) // Empty \_except block

#pragma warning(disable:6255) // \_alloca indicates failure by raising a stack overflow exception. Consider using \_malloca instead.

#pragma warning(disable:28112) // A variable (processNativeCodeSize) which is accessed via an Interlocked function must always be accessed via an Interlocked function. See line 1024: It is not always safe to access a variable which is accessed via the Interlocked\* family of functions in any other way.

#pragma warning(disable:28159) // Consider using 'GetTickCount64' instead of 'GetTickCount'. Reason: GetTickCount overflows roughly every 49 days. Code that does not take that into account can loop indefinitely. GetTickCount64 operates on 64 bit values and does not have that problem

#ifndef NTBUILD

// Would be nice to clean these up.

#pragma warning(disable:6054) // String 'dumpName' might not be zero-terminated.

#pragma warning(disable:6244) // Local declaration of 'Completed' hides previous declaration at line '76'

#pragma warning(disable:6246) // Local declaration of 'i' hides declaration of the same name in outer scope.

#pragma warning(disable:6326) // Potential comparison of a constant with another constant.

#endif

#endif // \_PREFAST\_

<?xml version="1.0" encoding="utf-8"?>

<Project DefaultTargets="Build" ToolsVersion="12.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Import Condition="'$(ChakraBuildPathImported)'!='true'" Project="$(SolutionDir)Chakra.Build.Paths.props"/>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.ProjectConfiguration.props" />

<ItemGroup>

<ClCompile Include="$(MSBuildThisFileDirectory)\Utf8Codex.cpp" />

<ClInclude Include="Utf8Codex.h" />

</ItemGroup>

<PropertyGroup Label="Globals">

<TargetName>Chakra.Common.Codex</TargetName>

<ProjectGuid>{1876E800-AD77-48C4-A2F7-E5265F24AC38}</ProjectGuid>

</PropertyGroup>

<PropertyGroup Label="Configuration">

<ConfigurationType>StaticLibrary</ConfigurationType>

</PropertyGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />

<Import Project="$(BuildConfigPropsPath)Chakra.Build.Props" />

<ImportGroup Label="PropertySheets">

<Import Project="$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props" Condition="exists('$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props')" Label="LocalAppDataPlatform" />

</ImportGroup>

<ItemDefinitionGroup>

<ClCompile>

<PreprocessorDefinitions>%(PreprocessorDefinitions);INC\_OLE2</PreprocessorDefinitions>

</ClCompile>

</ItemDefinitionGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.targets" Condition="exists('$(BuildConfigPropsPath)Chakra.Build.targets')"/>

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />

</Project>

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "Utf8Codex.h"

extern void CodexAssert(bool condition);

namespace utf8

{

const unsigned int mAlignmentMask = 0x3;

inline bool IsAligned(LPCUTF8 pch)

{

return (reinterpret\_cast<size\_t>(pch) & mAlignmentMask) == 0;

}

inline bool IsAligned(LPCOLESTR pch)

{

return (reinterpret\_cast<size\_t>(pch) & mAlignmentMask) == 0;

}

inline bool ShouldFastPath(LPCUTF8 pb, LPCOLESTR pch)

{

return (reinterpret\_cast<size\_t>(pb) & mAlignmentMask) == 0 || (reinterpret\_cast<size\_t>(pch) & mAlignmentMask) == 0;

}

inline size\_t EncodedBytes(wchar\_t prefix)

{

CodexAssert(0 == (prefix & 0xFF00)); // prefix must really be a byte. We use wchar\_t for as a convenience for the API.

// The number of bytes in an UTF8 encoding is determined by the 4 high-order bits of the first byte.

// 0xxx -> 1

// 10xx -> 1 (invalid)

// 110x -> 2

// 1110 -> 3

// 1111 -> 4

// If this value is XOR with 0xF0 and shift 3 bits to the right it can be used as an

// index into a 16 element 2 bit array encoded as a uint32 of n - 1 where n is the number

// of bits in the encoding.

// The XOR prefix bits mapped to n - 1.

// 1xxx -> 00 (8 - 15)

// 01xx -> 00 (4 - 7)

// 001x -> 01 (2 - 3)

// 0001 -> 10 (1)

// 0000 -> 11 (0)

// This produces the following bit sequence:

// 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

// 00 00 00 00 00 00 00 00 00 00 00 00 01 01 10 11

// which is 0x5B

return ((0x5B >> (((prefix ^ 0xF0) >> 3) & 0x1E)) & 0x03) + 1;

}

const wchar\_t g\_chUnknown = wchar\_t(UNICODE\_UNKNOWN\_CHAR\_MARK);

const wchar\_t WCH\_UTF16\_HIGH\_FIRST = wchar\_t(0xd800);

const wchar\_t WCH\_UTF16\_HIGH\_LAST = wchar\_t(0xdbff);

const wchar\_t WCH\_UTF16\_LOW\_FIRST = wchar\_t(0xdc00);

const wchar\_t WCH\_UTF16\_LOW\_LAST = wchar\_t(0xdfff);

inline BOOL InRange(const wchar\_t ch, const wchar\_t chMin, const wchar\_t chMax)

{

return (unsigned)(ch - chMin) <= (unsigned)(chMax - chMin);

}

inline BOOL IsValidWideChar(const wchar\_t ch)

{

return (ch < 0xfdd0) || ((ch > 0xfdef) && (ch <= 0xffef)) || ((ch >= 0xfff9) && (ch <= 0xfffd));

}

inline BOOL IsHighSurrogateChar(wchar\_t ch)

{

return InRange( ch, WCH\_UTF16\_HIGH\_FIRST, WCH\_UTF16\_HIGH\_LAST );

}

inline BOOL IsLowSurrogateChar(wchar\_t ch)

{

return InRange( ch, WCH\_UTF16\_LOW\_FIRST, WCH\_UTF16\_LOW\_LAST );

}

\_At\_(ptr, \_In\_reads\_(end - ptr) \_Post\_satisfies\_(ptr >= \_Old\_(ptr) - 1 && ptr <= end))

inline wchar\_t DecodeTail(wchar\_t c1, LPCUTF8& ptr, LPCUTF8 end, DecodeOptions& options)

{

wchar\_t ch = 0;

BYTE c2, c3, c4;

switch (EncodedBytes(c1))

{

case 1:

if (c1 < 0x80) return c1;

if ((options & doSecondSurrogatePair) != 0)

{

// We're in the middle of decoding a surrogate pair from a four-byte utf8 sequence.

// The high word has already been returned, but without advancing ptr, which was on byte 1.

// ptr was then advanced externally when reading c1, which is byte 1, so ptr is now on byte 2.

// byte 1 must have been a continuation byte, hence will be in case 1.

ptr--; // back to byte 1

c1 = ptr[-1]; // the original first byte

// ptr is now on c2. We must also have c3 and c4, otherwise doSecondSurrogatePair won't set.

\_Analysis\_assume\_(ptr + 2 < end);

goto LFourByte;

}

// 10xxxxxx (trail byte appearing in a lead byte position

return g\_chUnknown;

case 2:

// Look for an overlong utf-8 sequence.

if (ptr >= end)

{

if ((options & doChunkedEncoding) != 0)

// The is a sequence that spans a chunk, push ptr back to the beginning of the sequence.

ptr--;

return g\_chUnknown;

}

c2 = \*ptr++;

// 110XXXXx 10xxxxxx

// UTF16 | UTF8 1st byte 2nd byte

// U+0080..U+07FF | C2..DF 80..BF

if (

InRange(c1, 0xC2, 0xDF)

&& InRange(c2, 0x80, 0xBF)

)

{

ch |= WCHAR(c1 & 0x1f) << 6; // 0x0080 - 0x07ff

ch |= WCHAR(c2 & 0x3f);

if (!IsValidWideChar(ch) && ((options & doAllowInvalidWCHARs) == 0))

ch = g\_chUnknown;

}

else

{

ptr--;

ch = g\_chUnknown;

}

break;

case 3:

// 1110XXXX 10Xxxxxx 10xxxxxx

// Look for overlong utf-8 sequence.

if (ptr + 1 >= end)

{

if ((options & doChunkedEncoding) != 0)

// The is a sequence that spans a chunk, push ptr back to the beginning of the sequence.

ptr--;

return g\_chUnknown;

}

// UTF16 | UTF8 1st byte 2nd byte 3rd byte

// U+0800..U+0FFF | E0 A0..BF 80..BF

// U+1000..U+CFFF | E1..EC 80..BF 80..BF

// U+D000..U+D7FF | ED 80..9F 80..BF

// U+E000..U+FFFF | EE..EF 80..BF 80..BF

c2 = ptr[0];

c3 = ptr[1];

if (

// any following be true

(c1 == 0xE0

&& InRange(c2, 0xA0, 0xBF)

&& InRange(c3, 0x80, 0xBF))

||

(InRange(c1, 0xE1, 0xEC)

&& InRange(c2, 0x80, 0xBF)

&& InRange(c3, 0x80, 0xBF))

||

(c1 == 0xED

&& InRange(c2, 0x80, 0x9F)

&& InRange(c3, 0x80, 0xBF))

||

(InRange(c1, 0xEE, 0xEF)

&& InRange(c2, 0x80, 0xBF)

&& InRange(c3, 0x80, 0xBF))

||

(((options & doAllowThreeByteSurrogates) != 0)

&&

c1 == 0xED

&& InRange(c2, 0x80, 0xBF)

&& InRange(c3, 0x80, 0xBF)

)

)

{

ch = WCHAR(c1 & 0x0f) << 12; // 0x0800 - 0xffff

ch |= WCHAR(c2 & 0x3f) << 6; // 0x0080 - 0x07ff

ch |= WCHAR(c3 & 0x3f);

if (!IsValidWideChar(ch) && ((options & (doAllowThreeByteSurrogates | doAllowInvalidWCHARs)) == 0))

ch = g\_chUnknown;

ptr += 2;

}

else

{

ch = g\_chUnknown;

// Windows OS 1713952. Only drop the illegal leading byte

// Retry next byte.

// ptr is already advanced.

}

break;

case 4:

LFourByte:

// 11110XXX 10XXxxxx 10xxxxxx 10xxxxxx or 11111xxx ....

// NOTE: 11111xxx is not supported

if (ptr + 2 >= end)

{

if ((options & doChunkedEncoding) != 0)

// The is a sequence that spans a chunk, push ptr back to the beginning of the sequence.

ptr--;

ch = g\_chUnknown;

break;

}

c2 = ptr[0];

c3 = ptr[1];

c4 = ptr[2];

// UTF16 | UTF8 1st byte 2nd byte 3rd byte 4th byte

// U+10000..U+3FFFF | F0 90..BF 80..BF 80..BF

// U+40000..U+FFFFF | F1..F3 80..BF 80..BF 80..BF

// U+100000..U+10FFFF | F4 80..8F 80..BF 80..BF

if (! // NOT Unicode well-formed byte sequences

(

// any following be true

(c1 == 0xF0

&& InRange(c2, 0x90,0xBF)

&& InRange(c3, 0x80,0xBF)

&& InRange(c4, 0x80,0xBF))

||

(InRange(c1, 0xF1, 0xF3)

&& InRange(c2, 0x80,0xBF)

&& InRange(c3, 0x80,0xBF)

&& InRange(c4, 0x80,0xBF))

||

(c1 == 0xF4

&& InRange(c2, 0x80,0x8F)

&& InRange(c3, 0x80,0xBF)

&& InRange(c4, 0x80,0xBF))

)

)

{

// Windows OS 1713952. Only drop the illegal leading byte.

// Retry next byte.

// ptr is already advanced 1.

ch = g\_chUnknown;

break;

}

if ((options & doSecondSurrogatePair) == 0)

{

// Decode high 10 bits of utf-8 20 bit char

ch = WCHAR(c1 & 0x07) << 2;

ch |= WCHAR(c2 & 0x30) >> 4;

ch = (ch - 1) << 6; // ch == 0000 00ww ww00 0000

ch |= WCHAR(c2 & 0x0f) << 2; // ch == 0000 00ww wwzz zz00

ch |= WCHAR(c3 & 0x30) >> 4; // ch == 0000 00ww wwzz zzyy

// Encode first word of utf-16 surrogate pair

ch += 0xD800;

// Remember next call must return second word

options = (DecodeOptions)(options | doSecondSurrogatePair);

// Leave ptr on byte 1, this way:

// - callers who test that ptr has been advanced by utf8::Decode will see progress for

// both words of the surrogate pair.

// - callers who calculate the number of multi-unit chars by subtracting after from before ptr

// will accumulate 0 for first word and 2 for second, thus utf8 chars equals 2 utf16 chars + 2

// multi-unit chars, as it should be.

}

else

{

// Decode low 10 bits of utf-8 20 bit char

ch = WCHAR(c3 & 0x0f) << 6; // ch == 0000 00yy yy00 0000

ch |= WCHAR(c4 & 0x3f); // ch == 0000 00yy yyxx xxxx

// Encode second word of utf-16 surrogate pair

ch += 0xDC00;

// We're done with this char

options = (DecodeOptions)(options & ~doSecondSurrogatePair);

ptr += 3; // remember, got here by subtracting one from ptr in case 1, so effective increment is 2

}

break;

}

return ch;

}

LPUTF8 EncodeFull(wchar\_t ch, \_\_out\_ecount(3) LPUTF8 ptr)

{

if( ch < 0x0080 )

{

// One byte

\*ptr++ = static\_cast< utf8char\_t >(ch);

}

else if( ch < 0x0800 )

{

// Two bytes : 110yyyxx 10xxxxxx

\*ptr++ = static\_cast<utf8char\_t>(ch >> 6) | 0xc0;

\*ptr++ = static\_cast<utf8char\_t>(ch & 0x3F) | 0x80;

}

else

{

// Three bytes : 1110yyyy 10yyyyxx 10xxxxxx

\*ptr++ = static\_cast<utf8char\_t>(ch >> 12) | 0xE0;

\*ptr++ = static\_cast<utf8char\_t>((ch >> 6) & 0x3F) | 0x80;

\*ptr++ = static\_cast<utf8char\_t>(ch & 0x3F) | 0x80;

}

return ptr;

}

LPCUTF8 NextCharFull(LPCUTF8 ptr)

{

return ptr + EncodedBytes(\*ptr);

}

LPCUTF8 PrevCharFull(LPCUTF8 ptr, LPCUTF8 start)

{

if (ptr > start)

{

LPCUTF8 current = ptr - 1;

while (current > start && (\*current & 0xC0) == 0x80)

current--;

if (NextChar(current) == ptr)

return current;

// It is not a valid encoding, just go back one character.

return ptr - 1;

}

else

return ptr;

}

void DecodeInto(\_\_out\_ecount\_full(cch) wchar\_t \*buffer, LPCUTF8 ptr, size\_t cch, DecodeOptions options)

{

DecodeOptions localOptions = options;

if (!ShouldFastPath(ptr, buffer)) goto LSlowPath;

LFastPath:

while (cch >= 4)

{

uint32 bytes = \*(uint32 \*)ptr;

if ((bytes & 0x80808080) != 0) goto LSlowPath;

((uint32 \*)buffer)[0] = (bytes & 0x7F) | ((bytes << 8) & 0x7F0000);

((uint32 \*)buffer)[1] = ((bytes >> 16) & 0x7F) | ((bytes >> 8) & 0x7F0000);

ptr += 4;

buffer += 4;

cch -= 4;

}

LSlowPath:

while (cch-- > 0)

{

\*buffer++ = Decode(ptr, ptr + 4, localOptions); // WARNING: Assume cch correct, suppress end-of-buffer checking

if (ShouldFastPath(ptr, buffer)) goto LFastPath;

}

}

void DecodeIntoAndNullTerminate(\_\_out\_ecount(cch+1) \_\_nullterminated wchar\_t \*buffer, LPCUTF8 ptr, size\_t cch, DecodeOptions options)

{

DecodeInto(buffer, ptr, cch, options);

buffer[cch] = 0;

}

\_Ret\_range\_(0, pbEnd - \_Old\_(pbUtf8))

size\_t DecodeUnitsInto(\_Out\_writes\_(pbEnd - pbUtf8) wchar\_t \*buffer, LPCUTF8& pbUtf8, LPCUTF8 pbEnd, DecodeOptions options)

{

DecodeOptions localOptions = options;

LPCUTF8 p = pbUtf8;

wchar\_t \*dest = buffer;

if (!ShouldFastPath(p, dest)) goto LSlowPath;

LFastPath:

while (p + 3 < pbEnd)

{

unsigned bytes = \*(unsigned \*)p;

if ((bytes & 0x80808080) != 0) goto LSlowPath;

((uint32 \*)dest)[0] = (wchar\_t(bytes) & 0x00FF) | ((wchar\_t(bytes) & 0xFF00) << 8);

((uint32 \*)dest)[1] = (wchar\_t(bytes >> 16) & 0x00FF) | ((wchar\_t(bytes >> 16) & 0xFF00) << 8);

p += 4;

dest += 4;

}

LSlowPath:

while (p < pbEnd)

{

LPCUTF8 s = p;

wchar\_t chDest = Decode(p, pbEnd, localOptions);

if (s < p)

{

// We decoded the character, store it

\*dest++ = chDest;

}

else

{

// Nothing was converted. This might happen at the end of a buffer with doChunkedEncoding.

break;

}

if (ShouldFastPath(p, dest)) goto LFastPath;

}

pbUtf8 = p;

return dest - buffer;

}

size\_t DecodeUnitsIntoAndNullTerminate(\_\_out\_ecount(pbEnd - pbUtf8 + 1) \_\_nullterminated wchar\_t \*buffer, LPCUTF8& pbUtf8, LPCUTF8 pbEnd, DecodeOptions options)

{

size\_t result = DecodeUnitsInto(buffer, pbUtf8, pbEnd, options);

buffer[(int)result] = 0;

return result;

}

bool CharsAreEqual(\_\_in\_ecount(cch) LPCOLESTR pch, LPCUTF8 bch, size\_t cch, DecodeOptions options)

{

DecodeOptions localOptions = options;

while (cch-- > 0)

{

if (\*pch++ != utf8::Decode(bch, bch + 4, localOptions)) // WARNING: Assume cch correct, suppress end-of-buffer checking

return false;

}

return true;

}

\_\_range(0, cch \* 3)

size\_t EncodeInto(\_\_out\_ecount(cch \* 3) LPUTF8 buffer, \_\_in\_ecount(cch) const wchar\_t \*source, charcount\_t cch)

{

LPUTF8 dest = buffer;

if (!ShouldFastPath(dest, source)) goto LSlowPath;

LFastPath:

while (cch >= 4)

{

uint32 first = ((const uint32 \*)source)[0];

if ( (first & 0xFF80FF80) != 0) goto LSlowPath;

uint32 second = ((const uint32 \*)source)[1];

if ( (second & 0xFF80FF80) != 0) goto LSlowPath;

\*(uint32 \*)dest = (first & 0x0000007F) | ((first & 0x007F0000) >> 8) | ((second & 0x0000007f) << 16) | ((second & 0x007F0000) << 8);

dest += 4;

source += 4;

cch -= 4;

}

LSlowPath:

while( cch-- > 0 )

{

dest = Encode(\*source++, dest);

if (ShouldFastPath(dest, source)) goto LFastPath;

}

return dest - buffer;

}

\_\_range(0, cch \* 3)

size\_t EncodeIntoAndNullTerminate(\_\_out\_ecount(cch \* 3 + 1) utf8char\_t \*buffer, \_\_in\_ecount(cch) const wchar\_t \*source, charcount\_t cch)

{

size\_t result = EncodeInto(buffer, source, cch);

buffer[result] = 0;

return result;

}

// Convert the character index into a byte index.

size\_t CharacterIndexToByteIndex(\_\_in\_ecount(cbLength) LPCUTF8 pch, size\_t cbLength, charcount\_t cchIndex, DecodeOptions options)

{

return CharacterIndexToByteIndex(pch, cbLength, cchIndex, 0, 0, options);

}

size\_t CharacterIndexToByteIndex(\_\_in\_ecount(cbLength) LPCUTF8 pch, size\_t cbLength, const charcount\_t cchIndex, size\_t cbStartIndex, charcount\_t cchStartIndex, DecodeOptions options)

{

DecodeOptions localOptions = options;

LPCUTF8 pchCurrent = pch + cbStartIndex;

LPCUTF8 pchEnd = pch + cbLength;

LPCUTF8 pchEndMinus4 = pch + (cbLength - 4);

charcount\_t i = cchIndex - cchStartIndex;

// Avoid using a reinterpret\_cast to start a misaligned read.

if (!IsAligned(pchCurrent)) goto LSlowPath;

LFastPath:

// Skip 4 bytes at a time.

while (pchCurrent < pchEndMinus4 && i > 4)

{

uint32 ch4 = \*reinterpret\_cast<const uint32 \*>(pchCurrent);

if ((ch4 & 0x80808080) == 0)

{

pchCurrent += 4;

i -= 4;

}

else break;

}

LSlowPath:

while (pchCurrent < pchEnd && i > 0)

{

Decode(pchCurrent, pchEnd, localOptions);

i--;

// Try to return to the fast path avoiding misaligned reads.

if (i > 4 && IsAligned(pchCurrent)) goto LFastPath;

}

return i > 0 ? cbLength : pchCurrent - pch;

}

// Convert byte index into character index

charcount\_t ByteIndexIntoCharacterIndex(\_\_in\_ecount(cbIndex) LPCUTF8 pch, size\_t cbIndex, DecodeOptions options)

{

DecodeOptions localOptions = options;

LPCUTF8 pchCurrent = pch;

LPCUTF8 pchEnd = pch + cbIndex;

LPCUTF8 pchEndMinus4 = pch + (cbIndex - 4);

charcount\_t i = 0;

// Avoid using a reinterpret\_cast to start a misaligned read.

if (!IsAligned(pchCurrent)) goto LSlowPath;

LFastPath:

// Skip 4 bytes at a time.

while (pchCurrent < pchEndMinus4)

{

uint32 ch4 = \*reinterpret\_cast<const uint32 \*>(pchCurrent);

if ((ch4 & 0x80808080) == 0)

{

pchCurrent += 4;

i += 4;

}

else break;

}

LSlowPath:

while (pchCurrent < pchEnd)

{

LPCUTF8 s = pchCurrent;

Decode(pchCurrent, pchEnd, localOptions);

if (s == pchCurrent) break;

i++;

// Try to return to the fast path avoiding misaligned reads.

if (IsAligned(pchCurrent)) goto LFastPath;

}

return i;

}

} // namespace utf8

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include <windows.h>

#include <wtypes.h>

typedef unsigned \_\_int32 uint32;

// charcount\_t represents a count of characters in a String

// It is unsigned and the maximum value is (INT\_MAX-1)

typedef uint32 charcount\_t;

typedef BYTE utf8char\_t;

typedef const utf8char\_t CUTF8;

typedef utf8char\_t\* LPUTF8;

typedef const utf8char\_t \*LPCUTF8;

// Unicode 4.0, unknown char should be converted to replace mark, U+FFFD.

#define UNICODE\_UNKNOWN\_CHAR\_MARK 0xFFFD

#define UNICODE\_TCHAR\_UKNOWN\_CHAR\_MARK \_T('\xFFFD')

namespace utf8

{

// Terminology -

// Code point - A ordinal value mapped to an standard ideograph as defined by ISO/IEC 10646-1. Here

// also referred to as a UCS code point but can also be often be referred to as a UNICODE

// code point.

// UTF-8 - An encoding of UCS code points as defined by RFC-3629.

// UTF-16 - An encoding of UCS code points as defined by RFC-2781. Use as a synonym for UNICODE or

// UCS-2. This is technically incorrect but usually harmless. This file assumes wchar\_t \*

// maps to an UTF-16LE (little-endian) encoded sequence of words.

// Unit - The unit of encoding. For UTF-8 it is a byte (octet). For UTF-16 it is a word (two octets).

// Valid - A UTF-8 byte sequence conforming to RFC-3629.

// Well-formed - A sequence of bytes that conform to the encoding pattern of UTF8 but might be too long or

// otherwise invalid. For example C0 80 is a well-formed but invalid encoding of U+0000.

// Start byte - A byte can start a well-formed UTF-8 sequence.

// Lead byte - A byte can start a well-formed multi-unit sequence but not a single byte sequence.

// Trail byte - A byte that can appear after a lead-byte in a well-formed multi-unit sequence.

// Surrogate pair - A UTF-16 word pair to encode characters outside the Unicode base plain as defined by

// RFC-2781. Two wchar\_t values are used to encode one UCS code point.

// character index - The index into a UTF-16 sequence.

// byte index - The index into a UTF-8 sequence.

// Return the number of bytes needed to encode the given character (ignoring surrogate pairs)

inline size\_t EncodedSize(wchar\_t ch)

{

if (ch < 0x0080) return 1;

if (ch < 0x0800) return 2;

return 3;

}

enum DecodeOptions

{

doDefault = 0x00,

doAllowThreeByteSurrogates = 0x01, // Allow invalid 3 byte encodings as would be encoded by CSEU-8

doChunkedEncoding = 0x02, // For sequences at the end of a buffer do not advance into incomplete sequences

// If incomplete UTF-8 sequence is encountered at the end of a buffer, this

// option will cause Decode() to not advance the ptr value and DecodeTail to

// move the pointer back one position so it again points to where c1 was read by

// Decode(). In effect, incomplete sequences are treated as if end pointed to the

// beginning incomplete sequence instead of in the middle of it.

doSecondSurrogatePair = 0x04, // A previous call to DecodeTail returned the first word of a UTF-16

// surrogate pair. The second call will return the second word and reset

// this 'option'.

doAllowInvalidWCHARs = 0x08, // Don't replace invalid wide chars with 0xFFFD

};

DEFINE\_ENUM\_FLAG\_OPERATORS(DecodeOptions);

// Decode the trail bytes after the UTF8 lead byte c1 but returning 0xFFFD if trail bytes are expected after end.

\_At\_(ptr, \_In\_reads\_(end - ptr) \_Post\_satisfies\_(ptr >= \_Old\_(ptr) - 1 && ptr <= end))

wchar\_t DecodeTail(wchar\_t c1, LPCUTF8& ptr, LPCUTF8 end, DecodeOptions& options);

// Decode the UTF8 sequence into a UTF16 encoding. Code points outside the Unicode base plain will generate

// surrogate pairs, using the 'doSecondSurrogatePair' option to remember the first word has already been returned.

// If ptr == end 0x0000 is emitted. If ptr < end but the lead byte of the UTF8 sequence

// expects trail bytes past end then 0xFFFD are emitted until ptr == end.

\_At\_(ptr, \_In\_reads\_(end - ptr) \_Post\_satisfies\_(ptr >= \_Old\_(ptr) && ptr <= end))

inline wchar\_t Decode(LPCUTF8& ptr, LPCUTF8 end, DecodeOptions& options)

{

if (ptr >= end) return 0;

utf8char\_t c1 = \*ptr++;

if (c1 < 0x80) return static\_cast<wchar\_t>(c1);

return DecodeTail(c1, ptr, end, options);

}

// Encode ch into a UTF8 sequence ignoring surrogate pairs (which are encoded as two

// separate code points). Use Encode() instead of EncodeFull() directly because it

// special cases ASCII to avoid a call the most common characters.

LPUTF8 EncodeFull(wchar\_t ch, \_\_out\_ecount(3) LPUTF8 ptr);

// Encode ch into a UTF8 sequence ignoring surrogate pairs (which are encoded as two

// separate code points).

inline LPUTF8 Encode(wchar\_t ch, \_\_out\_ecount(3) LPUTF8 ptr)

{

if (ch < 0x80)

{

\*ptr = static\_cast<utf8char\_t>(ch);

return ptr + 1;

}

return EncodeFull(ch, ptr);

}

// Return true if ch is a lead byte of a UTF8 multi-unit sequence.

inline bool IsLeadByte(utf8char\_t ch)

{

return ch >= 0xC0;

}

// Return true if ch is a byte that starts a well-formed UTF8 sequence (i.e. is a ASCII character or a valid UTF8 lead byte)

inline bool IsStartByte(utf8char\_t ch)

{

return ch < 0x80 || ch >= 0xC0;

}

// Returns true if ch is a UTF8 multi-unit sequence trail byte.

inline bool IsTrailByte(utf8char\_t ch)

{

return (ch & 0xC0) == 0x80;

}

// Returns true if ptr points to a well-formed UTF8

inline bool IsCharStart(LPCUTF8 ptr)

{

return IsStartByte(\*ptr);

}

// Return the start of the next well-formed UTF-8 sequence. Use NextChar() instead of

// NextCharFull() since NextChar() avoid a call if ptr references a single byte sequence.

LPCUTF8 NextCharFull(LPCUTF8 ptr);

// Return the start of the next well-formed UTF-8 sequence.

inline LPCUTF8 NextChar(LPCUTF8 ptr)

{

if (\*ptr < 0x80) return ptr + 1;

return NextCharFull(ptr);

}

// Return the start of the previous well-formed UTF-8 sequence prior to start or start if

// if ptr is already start or no well-formed sequence starts a start. Use PrevChar() instead of

// PrevCharFull() since PrevChar() avoids a call if the previous sequence is a single byte

// sequence.

LPCUTF8 PrevCharFull(LPCUTF8 ptr, LPCUTF8 start);

// Return the start of the previous well-formed UTF-8 sequence prior to start or start if

// if ptr is already start or no well-formed sequence starts a start.

inline LPCUTF8 PrevChar(LPCUTF8 ptr, LPCUTF8 start)

{

if (ptr > start && \*(ptr - 1) < 0x80) return ptr - 1;

return PrevCharFull(ptr, start);

}

// Decode a UTF-8 sequence of cch UTF-16 characters into buffer. ptr could advance up to 3 times

// longer than cch so DecodeInto should only be used when it is already known that

// ptr refers to at least cch number of UTF-8 sequences.

void DecodeInto(\_\_out\_ecount\_full(cch) wchar\_t \*buffer, LPCUTF8 ptr, size\_t cch, DecodeOptions options = doDefault);

// Provided for dual-mode templates

inline void DecodeInto(\_\_out\_ecount\_full(cch) wchar\_t \*buffer, const wchar\_t \*ptr, size\_t cch, DecodeOptions /\* options \*/ = doDefault)

{

memcpy\_s(buffer, cch \* sizeof(wchar\_t), ptr, cch \* sizeof(wchar\_t));

}

// Like DecodeInto but ensures buffer ends with a NULL at buffer[cch].

void DecodeIntoAndNullTerminate(\_\_out\_ecount(cch+1) \_\_nullterminated wchar\_t \*buffer, LPCUTF8 ptr, size\_t cch, DecodeOptions options = doDefault);

// Decode cb bytes from ptr to into buffer returning the number of characters converted and written to buffer

\_Ret\_range\_(0, pbEnd - \_Old\_(pbUtf8))

size\_t DecodeUnitsInto(\_Out\_writes\_(pbEnd - pbUtf8) wchar\_t \*buffer, LPCUTF8& pbUtf8, LPCUTF8 pbEnd, DecodeOptions options = doDefault);

// Decode cb bytes from ptr to into buffer returning the number of characters converted and written to buffer (excluding the null terminator)

size\_t DecodeUnitsIntoAndNullTerminate(\_\_out\_ecount(pbEnd - pbUtf8 + 1) \_\_nullterminated wchar\_t \*buffer, LPCUTF8& pbUtf8, LPCUTF8 pbEnd, DecodeOptions options = doDefault);

// Encode a UTF-8 sequence into a UTF-8 sequence (which is just a memcpy). This is included for convenience in templates

// when the character encoding is a template parameter.

\_\_range(cch, cch)

inline size\_t EncodeInto(\_\_out\_ecount(cch) utf8char\_t \*buffer, const utf8char\_t \*source, size\_t cch)

{

memcpy\_s(buffer, cch \* sizeof(utf8char\_t), source, cch \* sizeof(utf8char\_t));

return cch;

}

// Encode a UTF16-LE sequence of cch words into a UTF-8 sequence returning the number of bytes needed.

// Since a UTF16 encoding can take up to 3 bytes buffer must refer to a buffer at least 3 times larger

// than cch.

// Returns the number of bytes copied into the buffer.

\_\_range(0, cch \* 3)

size\_t EncodeInto(\_\_out\_ecount(cch \* 3) LPUTF8 buffer, \_\_in\_ecount(cch) const wchar\_t \*source, charcount\_t cch);

// Like EncodeInto but ensures that buffer[return value] == 0.

\_\_range(0, cch \* 3)

size\_t EncodeIntoAndNullTerminate(\_\_out\_ecount(cch \* 3 + 1) utf8char\_t \*buffer, \_\_in\_ecount(cch) const wchar\_t \*source, charcount\_t cch);

// Returns true if the pch refers to a UTF-16LE encoding of the given UTF-8 encoding bch.

bool CharsAreEqual(\_\_in\_ecount(cch) LPCOLESTR pch, LPCUTF8 bch, size\_t cch, DecodeOptions options = doDefault);

// Convert the character index into a byte index.

size\_t CharacterIndexToByteIndex(\_\_in\_ecount(cbLength) LPCUTF8 pch, size\_t cbLength, const charcount\_t cchIndex, size\_t cbStartIndex, charcount\_t cchStartIndex, DecodeOptions options = doDefault);

size\_t CharacterIndexToByteIndex(\_\_in\_ecount(cbLength) LPCUTF8 pch, size\_t cbLength, const charcount\_t cchIndex, DecodeOptions options = doDefault);

// Convert byte index into character index

charcount\_t ByteIndexIntoCharacterIndex(\_\_in\_ecount(cbIndex) LPCUTF8 pch, size\_t cbIndex, DecodeOptions options = doDefault);

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

\_\_inline void \_\_stdcall js\_memcpy\_s(\_\_bcount(sizeInBytes) void \*dst, size\_t sizeInBytes, \_\_in\_bcount(count) const void \*src, size\_t count)

{

Assert((count) <= (sizeInBytes));

if ((count) <= (sizeInBytes))

memcpy((dst), (src), (count));

else

Js::Throw::FatalInternalError();

}

\_\_inline void \_\_stdcall js\_wmemcpy\_s(\_\_ecount(sizeInWords) wchar\_t \*dst, size\_t sizeInWords, \_\_in\_ecount(count) const wchar\_t \*src, size\_t count)

{

//Multiplication Overflow check

Assert(count <= sizeInWords && count <= SIZE\_MAX/sizeof(wchar\_t));

if(!(count <= sizeInWords && count <= SIZE\_MAX/sizeof(wchar\_t)))

{

Js::Throw::FatalInternalError();

}

memcpy(dst, src, count \* sizeof(wchar\_t));

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#if (defined(\_M\_IX86) && (\_MSC\_FULL\_VER > 13009037)) || ((defined(\_M\_AMD64) || defined(\_M\_IA64)) && (\_MSC\_FULL\_VER > 13009175)) || defined(\_M\_ARM32\_OR\_ARM64)

#ifdef \_\_cplusplus

extern "C" {

#endif

\_Check\_return\_ unsigned short \_\_cdecl \_byteswap\_ushort(\_In\_ unsigned short \_Short);

\_Check\_return\_ unsigned long \_\_cdecl \_byteswap\_ulong(\_In\_ unsigned long \_Long);

\_Check\_return\_ unsigned \_\_int64 \_\_cdecl \_byteswap\_uint64(\_In\_ unsigned \_\_int64 \_Int64);

#ifdef \_\_cplusplus

}

#endif

#pragma intrinsic(\_byteswap\_ushort)

#pragma intrinsic(\_byteswap\_ulong)

#pragma intrinsic(\_byteswap\_uint64)

#define RtlUshortByteSwap(\_x) \_byteswap\_ushort((USHORT)(\_x))

#define RtlUlongByteSwap(\_x) \_byteswap\_ulong((\_x))

#define RtlUlonglongByteSwap(\_x) \_byteswap\_uint64((\_x))

#else

#if (NTDDI\_VERSION >= NTDDI\_WIN2K)

NTSYSAPI

USHORT

FASTCALL

RtlUshortByteSwap(

\_In\_ USHORT Source

);

#endif

#if (NTDDI\_VERSION >= NTDDI\_WIN2K)

NTSYSAPI

ULONG

FASTCALL

RtlUlongByteSwap(

\_In\_ ULONG Source

);

#endif

#if (NTDDI\_VERSION >= NTDDI\_WIN2K)

NTSYSAPI

ULONGLONG

FASTCALL

RtlUlonglongByteSwap(

\_In\_ ULONGLONG Source

);

#endif

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#ifdef CONTROL\_FLOW\_GUARD\_LOGGER

#include "common\CFGLogger.h"

#include "core\DbgHelpSymbolManager.h"

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAQ")

bool CFGLogger::inGuard;

CriticalSection CFGLogger::cs;

JsUtil::BaseDictionary<uintptr\_t, uint, NoCheckHeapAllocator> CFGLogger::guardCheckRecord(&NoCheckHeapAllocator::Instance);

CFGLogger CFGLogger::Instance;

CFGLogger::PfnGuardCheckFunction CFGLogger::oldGuardCheck;

extern "C" PVOID \_\_guard\_check\_icall\_fptr;

void

CFGLogger::Enable()

{

DWORD oldProtect;

::VirtualProtect(&\_\_guard\_check\_icall\_fptr, sizeof(void \*), PAGE\_READWRITE, &oldProtect);

oldGuardCheck = (PfnGuardCheckFunction)\_\_guard\_check\_icall\_fptr;

\_\_guard\_check\_icall\_fptr = &CFGLogger::GuardCheck;

::VirtualProtect(&\_\_guard\_check\_icall\_fptr, sizeof(void \*), oldProtect, &oldProtect);

}

CFGLogger::~CFGLogger()

{

if (oldGuardCheck)

{

DWORD oldProtect;

::VirtualProtect(&\_\_guard\_check\_icall\_fptr, sizeof(void \*), PAGE\_READWRITE, &oldProtect);

\_\_guard\_check\_icall\_fptr = (PVOID)oldGuardCheck;

::VirtualProtect(&\_\_guard\_check\_icall\_fptr, sizeof(void \*), oldProtect, &oldProtect);

DbgHelpSymbolManager::EnsureInitialized();

size\_t total = 0;

guardCheckRecord.Map([&total](uintptr\_t Target, uint count)

{

DbgHelpSymbolManager::PrintSymbol((PVOID)Target);

Output::Print(L"%8d\n", count);

total += count;

});

Output::Print(L"Total: %d\n", total);

Output::Flush();

}

}

#ifdef \_CONTROL\_FLOW\_GUARD

\_\_declspec(guard(ignore))

#endif

void \_\_fastcall

CFGLogger::GuardCheck(\_In\_ uintptr\_t Target)

{

if (Target >= AutoSystemInfo::Data.dllLoadAddress && Target < AutoSystemInfo::Data.dllHighAddress)

{

AutoCriticalSection autocs(&cs);

if (inGuard) { return; }

inGuard = true;

uint \* count;

if (guardCheckRecord.TryGetReference(Target, &count))

{

(\*count)++;

}

else

{

guardCheckRecord.AddNew(Target, 1);

}

inGuard = false;

}

oldGuardCheck(Target);

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#ifdef CONTROL\_FLOW\_GUARD\_LOGGER

class CFGLogger

{

public:

static void Enable();

static void \_\_fastcall GuardCheck(\_In\_ uintptr\_t Target);

private:

CFGLogger() {}

~CFGLogger();

typedef void(\_\_fastcall \* PfnGuardCheckFunction)(\_In\_ uintptr\_t Target);

static PfnGuardCheckFunction oldGuardCheck;

static bool inGuard;

static CriticalSection cs;

static JsUtil::BaseDictionary<uintptr\_t, uint, NoCheckHeapAllocator> guardCheckRecord;

static CFGLogger Instance;

};

#endif

<?xml version="1.0" encoding="utf-8"?>

<Project DefaultTargets="Build" ToolsVersion="12.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Import Condition="'$(ChakraBuildPathImported)'!='true'" Project="$(SolutionDir)Chakra.Build.Paths.props" />

<Import Project="$(BuildConfigPropsPath)Chakra.Build.ProjectConfiguration.props" />

<PropertyGroup Label="Globals">

<TargetName>Chakra.Common.Common</TargetName>

<ProjectGuid>{BB4153FF-AC3E-4734-B562-CC23812DF31B}</ProjectGuid>

<RootNamespace>JS</RootNamespace>

<Keyword>Win32Proj</Keyword>

</PropertyGroup>

<PropertyGroup Label="Configuration">

<ConfigurationType>StaticLibrary</ConfigurationType>

</PropertyGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />

<ImportGroup Label="ExtensionSettings">

<Import Project="$(BuildConfig\_ARMASM\_Path)armasm.props" />

</ImportGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.props" />

<PropertyGroup>

<\_ProjectFileVersion>10.0.30319.1</\_ProjectFileVersion>

</PropertyGroup>

<ItemDefinitionGroup>

<ClCompile>

<AdditionalIncludeDirectories>

$(MSBuildThisFileDirectory)..;

%(AdditionalIncludeDirectories)

</AdditionalIncludeDirectories>

<!-- For windowsdatetimep.h -->

<AdditionalIncludeDirectories>$(ComIncPath)\winrt;%(AdditionalIncludeDirectories)</AdditionalIncludeDirectories>

<PrecompiledHeader>Use</PrecompiledHeader>

<PrecompiledHeaderFile>CommonCommonPch.h</PrecompiledHeaderFile>

</ClCompile>

</ItemDefinitionGroup>

<ItemGroup>

<ClCompile Include="$(MSBuildThisFileDirectory)Api.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)cfglogger.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)DaylightTimeHelper.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)DateUtilities.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Event.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Int32Math.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Int64Math.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Jobs.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)MathUtil.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)NumberUtilities.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)NumberUtilities\_strtod.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RejitReason.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmartFPUControl.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Tick.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)unicode.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)vtinfo.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CommonCommonPch.cpp">

<PrecompiledHeader>Create</PrecompiledHeader>

</ClCompile>

</ItemGroup>

<ItemGroup>

<ClInclude Include="Int64Math.h" />

<ClInclude Include="ByteSwap.h" />

<ClInclude Include="CommonCommonPch.h" />

<ClInclude Include="cfglogger.h" />

<ClInclude Include="DaylightTimeHelper.h" />

<ClInclude Include="DateUtilities.h" />

<ClInclude Include="Event.h" />

<ClInclude Include="GetCurrentFrameID.h" />

<ClInclude Include="Int32Math.h" />

<ClInclude Include="Jobs.h" />

<ClInclude Include="MathUtil.h" />

<ClInclude Include="NumberUtilities.h" />

<ClInclude Include="NumberUtilitiesBase.h" />

<ClInclude Include="RejitReason.h" />

<ClInclude Include="RejitReasons.h" />

<ClInclude Include="SmartFPUControl.h" />

<ClInclude Include="Tick.h" />

<ClInclude Include="UInt16Math.h" />

<ClInclude Include="UInt32Math.h" />

<ClInclude Include="unicode.h" />

<ClInclude Include="vtinfo.h" />

<ClInclude Include="vtregistry.h" />

</ItemGroup>

<ItemGroup>

<None Include="NumberUtilities.inl" />

<None Include="Jobs.inl" />

<None Include="vtinfo.inl" />

<ARMASM Include="$(MSBuildThisFileDirectory)arm\arm\_GET\_CURRENT\_FRAME.asm">

<ExcludedFromBuild Condition="'$(Platform)'!='ARM'">true</ExcludedFromBuild>

</ARMASM>

<ARMASM Include="$(MSBuildThisFileDirectory)arm64\arm64\_GET\_CURRENT\_FRAME.asm">

<ExcludedFromBuild Condition="'$(Platform)'!='ARM64'">true</ExcludedFromBuild>

</ARMASM>

</ItemGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.targets" Condition="exists('$(BuildConfigPropsPath)Chakra.Build.targets')" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />

<ImportGroup Label="ExtensionTargets">

<Import Project="$(BuildConfig\_ARMASM\_Path)armasm.targets" />

</ImportGroup>

</Project>

<?xml version="1.0" encoding="utf-8"?>

<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<ItemGroup>

<ClCompile Include="$(MSBuildThisFileDirectory)Api.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)DateUtilities.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Event.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Int32Math.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Jobs.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)MathUtil.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)NumberUtilities.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RejitReason.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Tick.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)vtinfo.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)cfglogger.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)DaylightTimeHelper.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)NumberUtilities\_strtod.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)unicode.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmartFPUControl.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CommonCommonPch.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Int64Math.cpp" />

</ItemGroup>

<ItemGroup>

<ClInclude Include="DateUtilities.h" />

<ClInclude Include="Event.h" />

<ClInclude Include="Jobs.h" />

<ClInclude Include="Int32Math.h" />

<ClInclude Include="MathUtil.h" />

<ClInclude Include="NumberUtilities.h" />

<ClInclude Include="RejitReason.h" />

<ClInclude Include="RejitReasons.h" />

<ClInclude Include="Tick.h" />

<ClInclude Include="UInt16Math.h" />

<ClInclude Include="UInt32Math.h" />

<ClInclude Include="vtinfo.h" />

<ClInclude Include="cfglogger.h" />

<ClInclude Include="DaylightTimeHelper.h" />

<ClInclude Include="unicode.h" />

<ClInclude Include="vtregistry.h" />

<ClInclude Include="NumberUtilities.inc" />

<ClInclude Include="ByteSwap.h" />

<ClInclude Include="CommonCommonPch.h" />

<ClInclude Include="GetCurrentFrameID.h" />

<ClInclude Include="NumberUtilitiesBase.h" />

<ClInclude Include="SmartFPUControl.h" />

<ClInclude Include="StdAfx.h" />

<ClInclude Include="Int64Math.h" />

</ItemGroup>

<ItemGroup>

<None Include="Jobs.inl" />

<None Include="vtinfo.inl" />

</ItemGroup>

<ItemGroup>

<Filter Include="arm">

<UniqueIdentifier>{e98481ce-aadd-448b-abe3-5278ade26549}</UniqueIdentifier>

</Filter>

<Filter Include="arm64">

<UniqueIdentifier>{1a6cbeee-b0c4-487f-828d-57990bee2978}</UniqueIdentifier>

</Filter>

</ItemGroup>

<ItemGroup>

<ARMASM Include="$(MSBuildThisFileDirectory)arm\arm\_GET\_CURRENT\_FRAME.asm">

<Filter>arm</Filter>

</ARMASM>

<ARMASM Include="$(MSBuildThisFileDirectory)arm64\arm64\_GET\_CURRENT\_FRAME.asm">

<Filter>arm64</Filter>

</ARMASM>

</ItemGroup>

</Project>

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonMin.h"

// === C Runtime Header Files ===

#if defined(\_UCRT)

#include <cmath>

#else

#include <math.h>

#endif

// === Codex Header Files ===

#include "codex\Utf8Codex.h"

// === Common Header Files ===

#include "Common\NumberUtilitiesBase.h"

#include "Common\NumberUtilities.h"

#pragma warning(push)

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(HEAP\_TRACK\_ALLOC) || defined(ENABLE\_DEBUG\_CONFIG\_OPTIONS)

#include <typeinfo.h>

#endif

#pragma warning(pop)

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#define ENABLE\_INTSAFE\_SIGNED\_FUNCTIONS 1

#include <intsafe.h>

#include "Common\DaylightTimeHelper.h"

#include "Common\DateUtilities.h"

#include <Windows.Foundation.h>

namespace Js

{

const INT64 DateUtilities::ticksPerMillisecond = 10000;

const double DateUtilities::ticksPerMillisecondDouble = 10000.0;

const INT64 DateUtilities::ticksPerSecond = ticksPerMillisecond \* 1000;

const INT64 DateUtilities::ticksPerMinute = ticksPerSecond \* 60;

const INT64 DateUtilities::ticksPerHour = ticksPerMinute \* 60;

const INT64 DateUtilities::ticksPerDay = ticksPerHour \* 24;

const INT64 DateUtilities::jsEpochMilliseconds = 11644473600000;

const INT64 DateUtilities::jsEpochTicks = jsEpochMilliseconds \* ticksPerMillisecond;

// The day numbers for the months of a leap year.

static const int g\_rgday[12] =

{

0, 31, 60, 91, 121, 152,

182, 213, 244, 274, 305, 335,

};

const double g\_kdblJanuary1st1970 = 25569.0;

const wchar\_t g\_rgpszDay[7][4] =

{

L"Sun",

L"Mon",

L"Tue",

L"Wed",

L"Thu",

L"Fri",

L"Sat"

};

const wchar\_t g\_rgpszMonth[12][4] =

{

L"Jan",

L"Feb",

L"Mar",

L"Apr",

L"May",

L"Jun",

L"Jul",

L"Aug",

L"Sep",

L"Oct",

L"Nov",

L"Dec"

};

const wchar\_t g\_rgpszZone[8][4] =

{

L"EST",

L"EDT",

L"CST",

L"CDT",

L"MST",

L"MDT",

L"PST",

L"PDT"

};

//

// Convert a WinRT DateTime date (in 100ns precision ticks) to an ES5 date

//

// We convert ticks to milliseconds and shift by JS epoch to get the double date

// We go in that order to skip doing underflow checks

//

HRESULT DateUtilities::WinRTDateToES5Date(INT64 ticks, \_\_out double\* pRet)

{

Assert(pRet != NULL);

if (pRet == NULL)

{

return E\_INVALIDARG;

}

// Divide as INT64 to ensure truncation of all decimal digits,

// since any remaining after conversion will be truncated as a Date value.

INT64 milliseconds = ticks / ticksPerMillisecond;

(\*pRet) = (double)(milliseconds - jsEpochMilliseconds);

return S\_OK;

}

//

// Convert an ES5 date based on double to a WinRT DateTime

// DateTime is the number of ticks that have elapsed since 1/1/1601 00:00:00 in 100ns precision

// If we return a failure HRESULT other than E\_INVALIDARG, the es5 date can't be expressed

// in the WinRT scheme

//

HRESULT DateUtilities::ES5DateToWinRTDate(double es5Date, \_\_out INT64\* pRet)

{

Assert(pRet != NULL);

if (pRet == NULL)

{

return E\_INVALIDARG;

}

INT64 es5DateAsInt64 = NumberUtilities::TryToInt64(es5Date);

if (!NumberUtilities::IsValidTryToInt64(es5DateAsInt64)) return INTSAFE\_E\_ARITHMETIC\_OVERFLOW;

INT64 numTicks;

// We use the LongLong\* functions since that's typedef'd to int64

// First, we rebase it to the WinRT epoch, then we convert the time in milliseconds to ticks

if (SUCCEEDED(::Int64Add(es5DateAsInt64, jsEpochMilliseconds, reinterpret\_cast<LONGLONG\*>(&numTicks))))

{

INT64 adjustedTicks = 0;

if (SUCCEEDED(::Int64Mult(numTicks, ticksPerMillisecond, reinterpret\_cast<LONGLONG\*>(&adjustedTicks))))

{

(\*pRet) = adjustedTicks;

return S\_OK;

}

}

return INTSAFE\_E\_ARITHMETIC\_OVERFLOW;

}

//

// Version 6 Change:

// Previously we would round the TimeSpan to ms precision in order to avoid having the double contain decimal digits.

// Now we allow for the timespan to be represented with integers and digits (no truncation).

//

HRESULT DateUtilities::WinRTTimeSpanToNumberV6(INT64 ticks, \_\_out double\* pRet)

{

Assert(pRet != NULL);

if (pRet == NULL)

{

return E\_INVALIDARG;

}

// We want to preserve precision as best we could, and for low enough timespan values

// Hence perform a division of doubles, to convert from ticks to milliseconds.

double result = (double)ticks / ticksPerMillisecondDouble;

\*pRet = result;

return S\_OK;

}

//

// Version 6 Change:

// Same as for WinRTTimeSpanToNumberV6, remove truncation when converting between Number and WinRT TimeSpan.

//

HRESULT DateUtilities::NumberToWinRTTimeSpanV6(double span, \_\_out INT64\* pRet)

{

Assert(pRet != NULL);

if (pRet == NULL)

{

return E\_INVALIDARG;

}

//Otherwise the double multiplication might overflow

if (span > MAXINT64 / ticksPerMillisecond)

{

return INTSAFE\_E\_ARITHMETIC\_OVERFLOW;

}

//Multiply before converting to Int64, in order to get the 100-nanosecond precision which will get truncated

INT64 spanAsInt64 = NumberUtilities::TryToInt64(span \* ticksPerMillisecondDouble);

if (!NumberUtilities::IsValidTryToInt64(spanAsInt64))

{

return INTSAFE\_E\_ARITHMETIC\_OVERFLOW;

}

(\*pRet) = spanAsInt64;

return S\_OK;

}

///------------------------------------------------------------------------------

/// Get a time value from SYSTEMTIME structure.

///

/// Returns number of milliseconds since Jan 1, 1970

///------------------------------------------------------------------------------

double

DateUtilities::TimeFromSt(SYSTEMTIME \*pst)

{

return TvFromDate(pst->wYear,pst->wMonth-1,pst->wDay-1, DayTimeFromSt(pst));

}

///------------------------------------------------------------------------------

/// Get a time value from SYSTEMTIME structure within a day

///

/// Returns number of milliseconds since 12:00 AM

///------------------------------------------------------------------------------

double

DateUtilities::DayTimeFromSt(SYSTEMTIME \*pst)

{

return (pst->wHour \* 3600000.0) + (pst->wMinute \* 60000.0) + (pst->wSecond \* 1000.0) + pst->wMilliseconds;

}

///------------------------------------------------------------------------------

/// Get a time value from (year, mon, day, time) values.

///------------------------------------------------------------------------------

double

DateUtilities::TvFromDate(double year, double mon, double day, double time)

{

// For positive month, use fast path: '/' and '%' rather than 'floor()' and 'fmod()'.

// But make sure there is no overflow when casting double -> int -- WOOB 1142298.

if (mon >= 0 && mon <= INT\_MAX)

{

year += ((int)mon) / 12;

mon = ((int)mon) % 12;

}

else

{

year += floor(mon/12);

mon = DblModPos(mon,12);

}

day += DayFromYear(year);

AssertMsg(mon >= 0 && mon <= 11, "'mon' must be in the range of [0..11].");

day += g\_rgday[(int)mon];

if (mon >= 2 && !FLeap((int)year))

{

day -= 1;

}

return day \* 86400000 + time;

}

///------------------------------------------------------------------------------

/// Get the non-negative remainder.

///------------------------------------------------------------------------------

double

DateUtilities::DblModPos(double dbl, double dblDen)

{

AssertMsg(dblDen > 0, "value not positive");

dbl = fmod(dbl, dblDen);

if (dbl < 0)

{

dbl += dblDen;

}

AssertMsg(dbl >= 0 && dbl < dblDen, "");

return dbl;

}

///------------------------------------------------------------------------------

/// DayFromYear is:

/// 365 \* y + floor((y+1)/4) - floor((y+69)/100) + floor((y+369)/400).

/// where y is the calendar year minus 1970.

///------------------------------------------------------------------------------

double

DateUtilities::DayFromYear(double year)

{

double day = 365 \* (year -= 1970);

if (day > 0)

{

day += ((int)((year + 1) / 4)) - ((int)((year + 69) / 100)) +

((int)((year + 369) / 400));

}

else

{

day += floor((year + 1) / 4) - floor((year + 69) / 100) +

floor((year + 369) / 400);

}

return day;

}

///------------------------------------------------------------------------------

/// Return whether the given year is a leap year.

///------------------------------------------------------------------------------

bool

DateUtilities::FLeap(int year)

{

return (0 == (year & 3)) && (0 != (year % 100) || 0 == (year % 400));

}

///------------------------------------------------------------------------------

/// Get the first day of the year, and if the first day is bigger than the passed day, then get the first day of the previous year.

///------------------------------------------------------------------------------

/\*static\*/

int DateUtilities::GetDayMinAndUpdateYear(int day, int &year)

{

int dayMin = (int)DayFromYear(year);

if (day < dayMin)

{

year--;

dayMin = (int)DayFromYear(year);

}

return dayMin;

}

///------------------------------------------------------------------------------

/// Converts the time value relative to Jan 1, 1970 into a YMD.

///

/// The year number y and day number d relative to Jan 1, 1970 satisfy the

/// inequalities:

/// floor((400\*d-82)/146097) <= y <= floor((400\*d+398)/146097)

/// These inequalities get us within one of the correct answer for the year.

/// We then use DayFromYear to adjust if necessary.

///------------------------------------------------------------------------------

void

DateUtilities::GetYmdFromTv(double tv, Js::YMD \*pymd)

{

// AssertMem(pymd);

int day;

int dayMin;

int yday;

if (tv > 0)

{

day = (int)(tv / 86400000);

pymd->time = (int)DblModPos(tv, 86400000);

pymd->wday = (day + 4) % 7;

pymd->year = 1970 + (int)((400 \* (double)day + 398) / 146097);

dayMin = GetDayMinAndUpdateYear(day, pymd->year);

pymd->yt = (int)((dayMin + 4) % 7);

}

else

{

day = (int)floor(tv / 86400000);

pymd->time = (int)DblModPos(tv, 86400000);

pymd->wday = (int)DblModPos(day + 4, 7);

pymd->year = 1970 + (int)floor(((400 \* (double)day + 398) / 146097));

dayMin = GetDayMinAndUpdateYear(day, pymd->year);

pymd->yt = (int)DblModPos(dayMin + 4, 7);

}

yday = (int)(day - dayMin);

// Assert(yday >= 0 && (yday < 365 || yday == 365 && FLeap(pymd->year)));

pymd->yday = yday;

if (FLeap(pymd->year))

{

pymd->yt += 7;

}

else if (yday >= 59)

{

yday++;

}

// Get the month.

if (yday < 182)

{

if (yday < 60)

{

pymd->mon = 0 + ((yday >= 31) ? 1 : 0);

}

else if (yday < 121)

{

pymd->mon = 2 + ((yday >= 91) ? 1 : 0);

}

else

{

pymd->mon = 4 + ((yday >= 152) ? 1 : 0);

}

}

else

{

if (yday < 244)

{

pymd->mon = 6 + ((yday >= 213) ? 1 : 0);

}

else if (yday < 305)

{

pymd->mon = 8 + ((yday >= 274) ? 1 : 0);

}

else

{

pymd->mon = 10 + ((yday >= 335) ? 1 : 0);

}

}

// Assert(pymd->mon >= 0 && pymd->mon < 12);

pymd->mday = yday - g\_rgday[pymd->mon];

}

void DateUtilities::GetYearFromTv(double tv, int &year, int &yearType)

{

// AssertMem(pymd);

int day;

int dayMin;

if (tv > 0)

{

day = (int)(tv / 86400000);

year = 1970 + (int)((400 \* (double)day + 398) / 146097);

dayMin = GetDayMinAndUpdateYear(day, year);

yearType = (int)((dayMin + 4) % 7);

}

else

{

day = (int)floor(tv / 86400000);

year = 1970 + (int)floor(((400 \* (double)day + 398) / 146097));

dayMin = GetDayMinAndUpdateYear(day, year);

yearType = (int)DblModPos(dayMin + 4, 7);

}

if (FLeap(year))

{

yearType += 7;

}

}

double DateUtilities::JsLocalTimeFromVarDate(double dbl)

{

// So that the arithmetic works even for negative dates, convert the

// date to the \_actual number of days\_ since 0000h 12/30/1899.

if (dbl < 0.0)

dbl = 2.0 \* ceil(dbl) - dbl;

// Get the local time value.

dbl = (dbl - g\_kdblJanuary1st1970) \* 86400000;

if (NumberUtilities::IsNan(dbl))

{

return dbl;

}

return NumberUtilities::IsFinite(dbl) ? floor( dbl + 0.5) : dbl;

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

// Global date constants

extern const double g\_kdblJanuary1st1970;

extern const wchar\_t g\_rgpszDay[7][4];

extern const wchar\_t g\_rgpszMonth[12][4];

extern const wchar\_t g\_rgpszZone[8][4];

// Utility methods to manipulate various date formats

class DateUtilities

{

static const INT64 ticksPerMillisecond;

static const double ticksPerMillisecondDouble;

static const INT64 ticksPerSecond;

static const INT64 ticksPerMinute;

static const INT64 ticksPerHour;

static const INT64 ticksPerDay;

static const INT64 jsEpochTicks;

static const INT64 jsEpochMilliseconds;

public:

static HRESULT WinRTDateToES5Date(INT64 winrtDate, \_\_out double\* pResult);

static HRESULT ES5DateToWinRTDate(double es5Date, \_\_out INT64\* pResult);

static HRESULT WinRTTimeSpanToNumberV6(INT64 span, \_\_out double\* pResult);

static HRESULT NumberToWinRTTimeSpanV6(double span, \_\_out INT64\* pResult);

static double TimeFromSt(SYSTEMTIME \*pst);

static double DayTimeFromSt(SYSTEMTIME \*pst);

static double TvFromDate(double year, double mon, double day, double time);

static double DblModPos(double dbl, double dblDen);

static double DayFromYear(double year);

static int GetDayMinAndUpdateYear(int day, int &year);

static bool FLeap(int year);

static void GetYmdFromTv(double tv, Js::YMD \*pymd);

static void GetYearFromTv(double tv, int &year, int &yearType);

// Used for VT\_DATE conversions

static double JsLocalTimeFromVarDate(double dbl);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include <time.h>

#include "Common\DaylightTimeHelper.h"

#include "Common\DateUtilities.h"

namespace Js {

static const double TicksPerMinute = 60000.0;

static const double TicksPerDay = TicksPerMinute \* 60 \* 24;

static const double TicksPerlargestTZOffset = TicksPerMinute \* 60 \* 24 + 1;

static const double TicksPerNonLeapYear = TicksPerDay \* 365;

static const double TicksPerSafeEndOfYear = TicksPerNonLeapYear - TicksPerlargestTZOffset;

static const int daysInMonthLeap[] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

static const double criticalMin = DateUtilities::TvFromDate(1601, 0, 1, 0); // minimal year for which windows has time zone information

static const double criticalMax = DateUtilities::TvFromDate(USHRT\_MAX-1, 0, 0, 0);

// The day numbers for the months of a leap year.

static const int daysUpToMonthLeap[12] =

{

0, 31, 60, 91, 121, 152,

182, 213, 244, 274, 305, 335,

};

typedef BOOL(\*DateConversionFunction)(

\_In\_opt\_ CONST PVOID lpTimeZoneInformation,

\_In\_ CONST SYSTEMTIME \* lpLocalTime,

\_Out\_ LPSYSTEMTIME lpUniversalTime

);

static HINSTANCE g\_timezonedll = NULL;

static DateConversionFunction sysLocalToUtc = NULL;

static DateConversionFunction sysUtcToLocal = NULL;

// Cache should be invalid at the moment of creation

// if january1 > nextJanuary1 cache is always invalid, so we don't care about other fields, because cache will be updated.

DaylightTimeHelper::TimeZoneInfo::TimeZoneInfo()

{

january1 = 1;

nextJanuary1 = 0;

}

// Cache is valid for given time if this time is within a year for which cache was created, and cache was updated within 1 second of current moment

bool DaylightTimeHelper::TimeZoneInfo::IsValid(double time)

{

return GetTickCount() - lastUpdateTickCount < updatePeriod && time >= january1 && time < nextJanuary1;

}

void DaylightTimeHelper::TimeZoneInfo::Update(double inputTime)

{

int year, yearType;

DateUtilities::GetYearFromTv(inputTime, year, yearType);

int yearForInfo = year;

// GetTimeZoneInformationForYear() works only with years > 1600, but JS works with wider range of years. So we take year closest to given.

if (year < 1601)

{

yearForInfo = 1601;

}

else if (year > 2100)

{

yearForInfo = 2100;

}

TIME\_ZONE\_INFORMATION timeZoneInfo;

GetTimeZoneInformationForYear((USHORT)yearForInfo, NULL, &timeZoneInfo);

isDaylightTimeApplicable = timeZoneInfo.StandardDate.wMonth != 0 && timeZoneInfo.DaylightDate.wMonth != 0;

bias = timeZoneInfo.Bias;

daylightBias = timeZoneInfo.DaylightBias;

standardBias = timeZoneInfo.StandardBias;

double day = DaylightTimeHelper::DayNumber(yearType, timeZoneInfo.DaylightDate);

double time = DateUtilities::DayTimeFromSt(&timeZoneInfo.DaylightDate);

daylightDate = DateUtilities::TvFromDate(year, timeZoneInfo.DaylightDate.wMonth-1, day-1, time);

day = DayNumber(yearType, timeZoneInfo.StandardDate);

time = DateUtilities::DayTimeFromSt(&timeZoneInfo.StandardDate);

standardDate = DateUtilities::TvFromDate(year, timeZoneInfo.StandardDate.wMonth-1, day-1, time);

GetTimeZoneInformationForYear((USHORT)yearForInfo-1, NULL, &timeZoneInfo);

isJanuary1Critical = timeZoneInfo.Bias + timeZoneInfo.DaylightBias + timeZoneInfo.StandardBias != bias + daylightBias + standardBias;

january1 = DateUtilities::TvFromDate(year, 0, 0, 0);

nextJanuary1 = january1 + TicksPerNonLeapYear + DateUtilities::FLeap(year) \* TicksPerDay;

lastUpdateTickCount = GetTickCount();

}

DaylightTimeHelper::TimeZoneInfo\* DaylightTimeHelper::GetTimeZoneInfo(double time)

{

if (cache1.IsValid(time)) return &cache1;

if (cache2.IsValid(time)) return &cache2;

if (useFirstCache)

{

cache1.Update(time);

useFirstCache = false;

return &cache1;

}

else

{

cache2.Update(time);

useFirstCache = true;

return &cache2;

}

}

HINSTANCE DaylightTimeHelper::TryLoadLibrary()

{

if (g\_timezonedll == NULL)

{

HMODULE hLocal = LoadLibraryExW(L"api-ms-win-core-timezone-l1-1-0.dll", nullptr, LOAD\_LIBRARY\_SEARCH\_SYSTEM32);

if (hLocal != NULL)

{

if (InterlockedCompareExchangePointer((PVOID\*) &g\_timezonedll, hLocal, NULL) != NULL)

{

FreeLibrary(hLocal);

}

}

}

if (g\_timezonedll == NULL)

{

HMODULE hLocal = LoadLibraryExW(L"kernel32.dll", nullptr, LOAD\_LIBRARY\_SEARCH\_SYSTEM32);

if (hLocal != NULL)

{

if (InterlockedCompareExchangePointer((PVOID\*) &g\_timezonedll, hLocal, NULL) != NULL)

{

FreeLibrary(hLocal);

}

}

}

return g\_timezonedll;

}

BOOL DaylightTimeHelper::SysLocalToUtc(SYSTEMTIME \*local, SYSTEMTIME \*utc)

{

if (sysLocalToUtc == NULL)

{

HINSTANCE library = TryLoadLibrary();

if (library != NULL && !ForceOldDateAPIFlag())

{

sysLocalToUtc = (DateConversionFunction)GetProcAddress(library, "TzSpecificLocalTimeToSystemTimeEx");

}

if (sysLocalToUtc == NULL)

{

sysLocalToUtc = (DateConversionFunction)TzSpecificLocalTimeToSystemTime;

}

}

return sysLocalToUtc(NULL, local, utc);

}

BOOL DaylightTimeHelper::SysUtcToLocal(SYSTEMTIME \*utc, SYSTEMTIME \*local)

{

if (sysUtcToLocal == NULL)

{

HINSTANCE library = TryLoadLibrary();

if (library != NULL)

{

sysUtcToLocal = (DateConversionFunction)GetProcAddress(library, "SystemTimeToTzSpecificLocalTimeEx");

}

if (sysUtcToLocal == NULL)

{

sysUtcToLocal = (DateConversionFunction)SystemTimeToTzSpecificLocalTime;

}

}

return sysUtcToLocal(NULL, utc, local);

}

int DaylightTimeHelper::DayNumber(int yearType, const SYSTEMTIME &date)

{

if (date.wYear == 0)

{

BOOL isLeap = yearType / 7; // yearType is a day of week of January 1st (number within range [0,6]) (+ 7 if year is a leap)

int dayOfWeekOf1stOfMonth = (yearType + daysUpToMonthLeap[date.wMonth-1] - (int)(!isLeap && date.wMonth >= 3)) % 7;

int numberOfDaysInThisMonth = daysInMonthLeap[date.wMonth-1] - (int)(!isLeap && date.wMonth == 2);

int delta = date.wDayOfWeek - dayOfWeekOf1stOfMonth;

return min((numberOfDaysInThisMonth - delta - 1) / 7, date.wDay - (int)(delta >= 0)) \* 7 + delta + 1;

}

else

{

return date.wDay;

}

}

bool DaylightTimeHelper::IsDaylightSavings(double time, bool utcTime, TimeZoneInfo \*timeZoneInfo)

{

if (!timeZoneInfo->isDaylightTimeApplicable)

{

return false;

}

double localDaylight = timeZoneInfo->daylightDate;

double localStandard = timeZoneInfo->standardDate;

if (utcTime)

{

double biasInTicks = timeZoneInfo->bias \* TicksPerMinute;

localDaylight += biasInTicks + (timeZoneInfo->standardBias \* TicksPerMinute);

localStandard += biasInTicks + (timeZoneInfo->daylightBias \* TicksPerMinute);

}

else

{

localDaylight -= (timeZoneInfo->daylightBias \* TicksPerMinute);

localStandard -= (timeZoneInfo->standardBias \* TicksPerMinute);

}

return (localDaylight < localStandard)

? localDaylight <= time && time < localStandard

: time < localStandard || localDaylight <= time;

}

// in slow path we use system API to perform conversion, but we still need to know whether current time is

// standard or daylight savings in order to create a string representation of a date.

// So just compare whether difference between local and utc time equal to bias.

bool DaylightTimeHelper::IsDaylightSavings(double utcTime, double localTime, int bias)

{

return ((int)(utcTime - localTime)) / ((int)(TicksPerMinute)) != bias;

}

// This function does not properly handle boundary cases.

// But while we use IsCritical we don't care about it.

bool DaylightTimeHelper::IsDaylightSavingsUnsafe(double time, TimeZoneInfo \*timeZoneInfo)

{

return timeZoneInfo->isDaylightTimeApplicable && ((timeZoneInfo->daylightDate < timeZoneInfo->standardDate)

? timeZoneInfo->daylightDate <= time && time < timeZoneInfo->standardDate

: time < timeZoneInfo->standardDate || timeZoneInfo->daylightDate <= time);

}

void DaylightTimeHelper::YmdToSystemTime(YMD\* ymd, SYSTEMTIME \*sys)

{

sys->wYear = (WORD)ymd->year;

sys->wMonth = (WORD)(ymd->mon + 1);

sys->wDay =(WORD)(ymd->mday + 1);

int time = ymd->time;

sys->wMilliseconds = (WORD)(time % 1000);

time /= 1000;

sys->wSecond = (WORD)(time % 60);

time /= 60;

sys->wMinute = (WORD)(time % 60);

time /= 60;

sys->wHour = (WORD)time;

}

double DaylightTimeHelper::UtcToLocalFast(double utcTime, TimeZoneInfo \*timeZoneInfo, int &bias, int &offset, bool &isDaylightSavings)

{

double localTime;

localTime = utcTime - TicksPerMinute \* timeZoneInfo->bias;

isDaylightSavings = IsDaylightSavingsUnsafe(utcTime, timeZoneInfo);

if (isDaylightSavings)

{

localTime -= TicksPerMinute \* timeZoneInfo->daylightBias;

} else {

localTime -= TicksPerMinute \* timeZoneInfo->standardBias;

}

bias = timeZoneInfo->bias;

offset = ((int)(localTime - utcTime)) / ((int)(TicksPerMinute));

return localTime;

}

double DaylightTimeHelper::UtcToLocalCritical(double utcTime, TimeZoneInfo \*timeZoneInfo, int &bias, int &offset, bool &isDaylightSavings)

{

double localTime;

SYSTEMTIME utcSystem, localSystem;

YMD ymd;

DateUtilities::GetYmdFromTv(utcTime, &ymd);

YmdToSystemTime(&ymd, &utcSystem);

if (!SysUtcToLocal(&utcSystem, &localSystem))

{

// SysUtcToLocal can fail if the date is beyond extreme internal boundaries (e.g. > ~30000 years).

// Fall back to our fast (but less accurate) version if the call fails.

return UtcToLocalFast(utcTime, timeZoneInfo, bias, offset, isDaylightSavings);

}

localTime = Js::DateUtilities::TimeFromSt(&localSystem);

if (localSystem.wYear != utcSystem.wYear)

{

timeZoneInfo = GetTimeZoneInfo(localTime);

}

bias = timeZoneInfo->bias;

isDaylightSavings = IsDaylightSavings(utcTime, localTime, timeZoneInfo->bias + timeZoneInfo->standardBias);

offset = ((int)(localTime - utcTime)) / ((int)(TicksPerMinute));

return localTime;

}

double DaylightTimeHelper::UtcToLocal(double utcTime, int &bias, int &offset, bool &isDaylightSavings)

{

TimeZoneInfo \*timeZoneInfo = GetTimeZoneInfo(utcTime);

if (IsCritical(utcTime, timeZoneInfo))

{

return UtcToLocalCritical(utcTime, timeZoneInfo, bias, offset, isDaylightSavings);

}

else

{

return UtcToLocalFast(utcTime, timeZoneInfo, bias, offset, isDaylightSavings);

}

}

double DaylightTimeHelper::LocalToUtcFast(double localTime, TimeZoneInfo \*timeZoneInfo)

{

double utcTime = localTime + TicksPerMinute \* timeZoneInfo->bias;

bool isDaylightSavings = IsDaylightSavingsUnsafe(localTime, timeZoneInfo);

if (isDaylightSavings)

{

utcTime += TicksPerMinute \* timeZoneInfo->daylightBias;

} else {

utcTime += TicksPerMinute \* timeZoneInfo->standardBias;

}

return utcTime;

}

double DaylightTimeHelper::LocalToUtcCritical(double localTime, TimeZoneInfo \*timeZoneInfo)

{

SYSTEMTIME localSystem, utcSystem;

YMD ymd;

DateUtilities::GetYmdFromTv(localTime, &ymd);

YmdToSystemTime(&ymd, &localSystem);

if (!SysLocalToUtc(&localSystem, &utcSystem))

{

// Fall back to our fast (but less accurate) version if the call fails.

return LocalToUtcFast(localTime, timeZoneInfo);

}

return Js::DateUtilities::TimeFromSt(&utcSystem);

}

double DaylightTimeHelper::LocalToUtc(double localTime)

{

TimeZoneInfo \*timeZoneInfo = GetTimeZoneInfo(localTime);

if (IsCritical(localTime, timeZoneInfo))

{

return LocalToUtcCritical(localTime, timeZoneInfo);

}

else

{

return LocalToUtcFast(localTime, timeZoneInfo);

}

}

// we consider January 1st, December 31st and days when daylight savings time starts and ands to be critical,

// because there might be ambiguous cases in local->utc->local conversions,

// so in order to be consistent with Windows we rely on it to perform conversions. But it is slow.

bool DaylightTimeHelper::IsCritical(double time, TimeZoneInfo \*timeZoneInfo)

{

return time > criticalMin && time < criticalMax &&

(abs(time - timeZoneInfo->daylightDate) < TicksPerlargestTZOffset ||

abs(time - timeZoneInfo->standardDate) < TicksPerlargestTZOffset ||

time > timeZoneInfo->january1 + TicksPerSafeEndOfYear ||

(timeZoneInfo->isJanuary1Critical && time - timeZoneInfo->january1 < TicksPerlargestTZOffset));

}

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

// Decomposed date (Year-Month-Date).

struct YMD

{

int year; // year

int yt; // year type: wkd of Jan 1 (plus 7 if a leap year).

int mon; // month (0 to 11)

int mday; // day in month (0 to 30)

int yday; // day in year (0 to 365)

int wday; // week day (0 to 6)

int time; // time of day (in milliseconds: 0 to 86399999)

};

class DaylightTimeHelper

{

private:

static const uint updatePeriod = 1000;

class TimeZoneInfo

{

public:

double daylightDate;

double standardDate;

double january1;

double nextJanuary1;

LONG daylightBias;

LONG standardBias;

LONG bias;

uint lastUpdateTickCount;

bool isDaylightTimeApplicable;

bool isJanuary1Critical;

TimeZoneInfo();

bool IsValid(double time);

void Update(double time);

};

TimeZoneInfo cache1, cache2;

bool useFirstCache;

static HINSTANCE TryLoadLibrary();

static BOOL SysLocalToUtc(SYSTEMTIME \*local, SYSTEMTIME \*utc);

static BOOL SysUtcToLocal(SYSTEMTIME \*utc, SYSTEMTIME \*local);

static inline int DayNumber(int yearType, const SYSTEMTIME &date);

static bool IsDaylightSavings(double time, bool isUtcTime, TimeZoneInfo \*timeZoneInfo);

static bool IsDaylightSavings(double utcTime, double localTime, int bias);

static bool IsDaylightSavingsUnsafe(double time, TimeZoneInfo \*timeZoneInfo);

static inline bool IsCritical(double time, TimeZoneInfo \*timeZoneInfo);

TimeZoneInfo \*GetTimeZoneInfo(double time);

public:

double UtcToLocal(double utcTime, int &bias, int &offset, bool &isDaylightSavings);

double LocalToUtc(double time);

static void YmdToSystemTime(YMD\* ymd, SYSTEMTIME \*sys);

// Moved DaylightTimeHelper to common.lib to share with hybrid debugging. However this function depends

// on runtime. Runtime and hybrid debugging needs to provide implementation.

static bool ForceOldDateAPIFlag();

private:

static inline double UtcToLocalFast(double utcTime, TimeZoneInfo \*timeZoneInfo, int &bias, int &offset, bool &isDaylightSavings);

inline double UtcToLocalCritical(double utcTime, TimeZoneInfo \*timeZoneInfo, int &bias, int &offset, bool &isDaylightSavings);

static inline double LocalToUtcFast(double localTime, TimeZoneInfo \*timeZoneInfo);

static inline double LocalToUtcCritical(double localTime, TimeZoneInfo \*timeZoneInfo);

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include "Common\Event.h"

Event::Event(const bool autoReset, const bool signaled) : handle(CreateEvent(0, !autoReset, signaled, 0))

{

if(!handle)

Js::Throw::OutOfMemory();

}

bool Event::Wait(const unsigned int milliseconds) const

{

const unsigned int result = WaitForSingleObject(handle, milliseconds);

if(!(result == WAIT\_OBJECT\_0 || (result == WAIT\_TIMEOUT && milliseconds != INFINITE)))

Js::Throw::FatalInternalError();

return result == WAIT\_OBJECT\_0;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

class Event

{

private:

const HANDLE handle;

public:

Event(const bool autoReset, const bool signaled = false);

private:

Event(const Event &) : handle(0)

{

}

Event &operator =(const Event &)

{

return \*this;

}

public:

~Event()

{

CloseHandle(handle);

}

public:

HANDLE Handle() const

{

return handle;

}

void Set() const

{

SetEvent(handle);

}

void Reset() const

{

ResetEvent(handle);

}

bool Wait(const unsigned int milliseconds = INFINITE) const;

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef \_M\_AMD64

#include "amd64.h"

#endif

#ifdef \_M\_ARM

#include "arm.h"

#endif

#ifdef \_M\_ARM64

#include "arm64.h"

#endif

#ifndef GET\_CURRENT\_FRAME\_ID

#if defined(\_M\_IX86)

#define GET\_CURRENT\_FRAME\_ID(f) \

\_\_asm { mov f, ebp }

#elif defined(\_M\_X64)

#define GET\_CURRENT\_FRAME\_ID(f) \

(f = \_ReturnAddress())

#elif defined(\_M\_ARM)

// ARM, like x86, uses the frame pointer rather than code address

#define GET\_CURRENT\_FRAME\_ID(f) \

(f = arm\_GET\_CURRENT\_FRAME())

#elif defined(\_M\_ARM64)

#define GET\_CURRENT\_FRAME\_ID(f) \

(f = arm64\_GET\_CURRENT\_FRAME())

#else

#define GET\_CURRENT\_FRAME\_ID(f) \

Js::Throw::NotImplemented()

#endif

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

class Int16Math

{

public:

template< class Func >

static int16 Add(int16 lhs, int16 rhs, \_\_inout Func& overflowFn)

{

int16 result = lhs + rhs;

// If the result is smaller than the LHS, then we overflowed

if( result < lhs )

{

overflowFn();

}

return result;

}

template< class Func >

static void Inc(int16& lhs, \_\_inout Func& overflowFn)

{

++lhs;

// If lhs becomes 0, then we overflowed

if(!lhs)

{

overflowFn();

}

}

// Convenience function which uses DefaultOverflowPolicy (throws OOM when overflow)

static int16 Add(int16 lhs, uint16 rhs)

{

return Add(lhs, rhs, ::Math::DefaultOverflowPolicy);

}

// Convenience functions which return a bool indicating overflow

static bool Add(int16 lhs, int16 rhs, \_\_out int16\* result)

{

::Math::RecordOverflowPolicy overflowGuard;

\*result = Add(lhs, rhs, overflowGuard);

return overflowGuard.HasOverflowed();

}

// Convenience function which uses DefaultOverflowPolicy (throws OOM when overflow)

static void Inc(int16& lhs)

{

Inc(lhs, ::Math::DefaultOverflowPolicy);

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include "Common\Int32Math.h"

bool

Int32Math::Add(int32 left, int32 right, int32 \*pResult)

{

if (sizeof(void \*) == 4)

{

// Overflow occurs when the result has a different sign from both the left and right operands

\*pResult = left + right;

return ((left ^ \*pResult) & (right ^ \*pResult)) < 0;

}

Assert(sizeof(void \*) == 8);

int64 result64 = (int64)left + (int64)right;

\*pResult = (int32)result64;

return result64 != (int64)(\*pResult);

}

bool

Int32Math::Mul(int32 left, int32 right, int32 \*pResult)

{

bool fOverflow;

#if \_M\_IX86

\_\_asm

{

mov eax, left

imul right

seto fOverflow

mov ecx, pResult

mov[ecx], eax

}

#else

int64 result64 = (int64)left \* (int64)right;

\*pResult = (int32)result64;

fOverflow = (result64 != (int64)(\*pResult));

#endif

return fOverflow;

}

bool

Int32Math::Mul(int32 left, int32 right, int32 \*pResult, int32\* pOverflowValue)

{

bool fOverflow;

#if \_M\_IX86

\_\_asm

{

mov eax, left

imul right

seto fOverflow

mov ecx, pResult

mov[ecx], eax

mov ecx, pOverflowValue

mov[ecx], edx

}

#else

int64 result64 = (int64)left \* (int64)right;

\*pResult = (int32)result64;

\*pOverflowValue = (int32)(result64 >> 32);

fOverflow = (result64 != (int64)(\*pResult));

#endif

return fOverflow;

}

bool

Int32Math::Shl(int32 left, int32 right, int32 \*pResult)

{

\*pResult = left << (right & 0x1F);

return (left != (int32)((uint32)\*pResult >> right));

}

bool

Int32Math::Sub(int32 left, int32 right, int32 \*pResult)

{

if(sizeof(void \*) == 4)

{

// Overflow occurs when the result has a different sign from the left operand, and the result has the same sign as the

// right operand

\*pResult = left - right;

return ((left ^ \*pResult) & ~(right ^ \*pResult)) < 0;

}

Assert(sizeof(void \*) == 8);

int64 result64 = (int64)left - (int64)right;

\*pResult = (int32)result64;

return result64 != (int64)(\*pResult);

}

bool

Int32Math::Div(int32 left, int32 right, int32 \*pResult)

{

AssertMsg(right != 0, "Divide by zero...");

if (right == -1 && left == INT\_MIN)

{

//Special check for INT\_MIN/-1

return true;

}

\*pResult = left / right;

return false;

}

bool

Int32Math::Mod(int32 left, int32 right, int32 \*pResult)

{

AssertMsg(right != 0, "Mod by zero...");

if (right == -1 && left == INT\_MIN)

{

//Special check for INT\_MIN/-1

return true;

}

\*pResult = left % right;

return false;

}

bool

Int32Math::Shr(int32 left, int32 right, int32 \*pResult)

{

\*pResult = left >> (right & 0x1F);

return false;

}

bool

Int32Math::ShrU(int32 left, int32 right, int32 \*pResult)

{

uint32 uResult = ((uint32)left) >> (right & 0x1F);

\*pResult = uResult;

return false;

}

bool

Int32Math::And(int32 left, int32 right, int32 \*pResult)

{

\*pResult = left & right;

return false;

}

bool

Int32Math::Or(int32 left, int32 right, int32 \*pResult)

{

\*pResult = left | right;

return false;

}

bool

Int32Math::Xor(int32 left, int32 right, int32 \*pResult)

{

\*pResult = left ^ right;

return false;

}

bool

Int32Math::Neg(int32 val, int32 \*pResult)

{

\*pResult = -val;

return \*pResult == INT\_MIN;

}

bool

Int32Math::Not(int32 val, int32 \*pResult)

{

\*pResult = ~val;

return false;

}

bool

Int32Math::Inc(int32 val, int32 \*pResult)

{

\*pResult = val + 1;

// Overflow if result ends up less than input

return \*pResult <= val;

}

bool

Int32Math::Dec(int32 val, int32 \*pResult)

{

\*pResult = val - 1;

// Overflow if result ends up greater than input

return \*pResult >= val;

}

int32

Int32Math::NearestInRangeTo(const int32 value, const int32 minimum, const int32 maximum) // inclusive

{

Assert(minimum <= maximum);

return minimum >= value ? minimum : maximum <= value ? maximum : value;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

class Int32Math

{

public:

static bool Add(int32 left, int32 right, int32 \*pResult);

static bool Sub(int32 left, int32 right, int32 \*pResult);

static bool Mul(int32 left, int32 right, int32 \*pResult);

static bool Mul(int32 left, int32 right, int32 \*pResult, int32\* pOverflowValue);

static bool Div(int32 left, int32 right, int32 \*pResult);

static bool Mod(int32 left, int32 right, int32 \*pResult);

static bool Shl(int32 left, int32 right, int32 \*pResult);

static bool Shr(int32 left, int32 right, int32 \*pResult);

static bool ShrU(int32 left, int32 right, int32 \*pResult);

static bool And(int32 left, int32 right, int32 \*pResult);

static bool Or(int32 left, int32 right, int32 \*pResult);

static bool Xor(int32 left, int32 right, int32 \*pResult);

static bool Neg(int32 val, int32 \*pResult);

static bool Not(int32 val, int32 \*pResult);

static bool Inc(int32 val, int32 \*pResult);

static bool Dec(int32 val, int32 \*pResult);

static int32 NearestInRangeTo(const int32 value, const int32 minimum, const int32 maximum); // inclusive

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include "Common\Int64Math.h"

#include <intrin.h>

#if \_M\_X64

#pragma intrinsic(\_mul128)

#endif

bool

Int64Math::Add(int64 left, int64 right, int64 \*pResult)

{

// Overflow occurs when the result has a different sign from both the left and right operands

\*pResult = left + right;

return ((left ^ \*pResult) & (right ^ \*pResult)) < 0;

}

bool

Int64Math::Mul(int64 left, int64 right, int64 \*pResult)

{

#if \_M\_X64

int64 high;

\*pResult = \_mul128(left, right, &high);

return high != 0;

#else

\*pResult = left \* right;

return (left != 0 && right != 0 && (\*pResult / left) != right);

#endif

}

bool

Int64Math::Shl(int64 left, int64 right, int64 \*pResult)

{

\*pResult = left << (right & 63);

return (left != (int64)((uint64)\*pResult >> right));

}

bool

Int64Math::Sub(int64 left, int64 right, int64 \*pResult)

{

// Overflow occurs when the result has a different sign from the left operand, and the result has the same sign as the

// right operand

\*pResult = left - right;

return ((left ^ \*pResult) & ~(right ^ \*pResult)) < 0;

}

bool

Int64Math::Div(int64 left, int64 right, int64 \*pResult)

{

AssertMsg(right != 0, "Divide by zero...");

if (right == -1 && left == MININT64)

{

//Special check for INT64\_MIN/-1

return true;

}

\*pResult = left / right;

return false;

}

bool

Int64Math::Mod(int64 left, int64 right, int64 \*pResult)

{

AssertMsg(right != 0, "Mod by zero...");

if (right == -1 && left == MININT64)

{

//Special check for INT64\_MIN/-1

return true;

}

\*pResult = left % right;

return false;

}

bool

Int64Math::Shr(int64 left, int64 right, int64 \*pResult)

{

\*pResult = left >> (right & 63);

return false;

}

bool

Int64Math::ShrU(int64 left, int64 right, int64 \*pResult)

{

uint64 uResult = ((uint64)left) >> (right & 63);

\*pResult = uResult;

return false;

}

bool

Int64Math::And(int64 left, int64 right, int64 \*pResult)

{

\*pResult = left & right;

return false;

}

bool

Int64Math::Or(int64 left, int64 right, int64 \*pResult)

{

\*pResult = left | right;

return false;

}

bool

Int64Math::Xor(int64 left, int64 right, int64 \*pResult)

{

\*pResult = left ^ right;

return false;

}

bool

Int64Math::Neg(int64 val, int64 \*pResult)

{

\*pResult = -val;

return \*pResult == MININT64;

}

bool

Int64Math::Not(int64 val, int64 \*pResult)

{

\*pResult = ~val;

return false;

}

bool

Int64Math::Inc(int64 val, int64 \*pResult)

{

\*pResult = val + 1;

// Overflow if result ends up less than input

return \*pResult <= val;

}

bool

Int64Math::Dec(int64 val, int64 \*pResult)

{

\*pResult = val - 1;

// Overflow if result ends up greater than input

return \*pResult >= val;

}

uint64

Int64Math::Log2(int64 val)

{

uint64 uval = (uint64)val;

uint64 ret;

for (ret = 0; uval >>= 1; ret++);

return ret;

}

int64

Int64Math::NearestInRangeTo(const int64 value, const int64 minimum, const int64 maximum) // inclusive

{

Assert(minimum <= maximum);

return minimum >= value ? minimum : maximum <= value ? maximum : value;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

class Int64Math

{

public:

static bool Add(int64 left, int64 right, int64 \*pResult);

static bool Sub(int64 left, int64 right, int64 \*pResult);

static bool Mul(int64 left, int64 right, int64 \*pResult);

static bool Div(int64 left, int64 right, int64 \*pResult);

static bool Mod(int64 left, int64 right, int64 \*pResult);

static bool Shl(int64 left, int64 right, int64 \*pResult);

static bool Shr(int64 left, int64 right, int64 \*pResult);

static bool ShrU(int64 left, int64 right, int64 \*pResult);

static bool And(int64 left, int64 right, int64 \*pResult);

static bool Or(int64 left, int64 right, int64 \*pResult);

static bool Xor(int64 left, int64 right, int64 \*pResult);

static bool Neg(int64 val, int64 \*pResult);

static bool Not(int64 val, int64 \*pResult);

static bool Inc(int64 val, int64 \*pResult);

static bool Dec(int64 val, int64 \*pResult);

static uint64 Log2(int64 val);

static int64 NearestInRangeTo(const int64 value, const int64 minimum, const int64 maximum); // inclusive

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include <process.h>

#include "core\EtwTraceCore.h"

#include "Exceptions\ExceptionBase.h"

#include "Exceptions\OperationAbortedException.h"

#include "Exceptions\OutOfMemoryException.h"

#include "Exceptions\StackOverflowException.h"

#include "TemplateParameter.h"

#include "DataStructures\DoublyLinkedListElement.h"

#include "DataStructures\DoublyLinkedList.h"

#include "DataStructures\DoublyLinkedListElement.inl"

#include "DataStructures\DoublyLinkedList.inl"

#include "Common\Event.h"

#include "Common\ThreadService.h"

#include "Common\Jobs.h"

#include "Common\Jobs.inl"

namespace Js

{

class JavascriptExceptionObject;

};

namespace JsUtil

{

// -------------------------------------------------------------------------------------------------------------------------

// Job

// -------------------------------------------------------------------------------------------------------------------------

Job::Job(const bool isCritical) : manager(0), isCritical(isCritical)

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

, failureReason(FailureReason::NotFailed)

#endif

{

}

Job::Job(JobManager \*const manager, const bool isCritical) : manager(manager), isCritical(isCritical)

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

, failureReason(FailureReason::NotFailed)

#endif

{

Assert(manager);

}

JobManager \*Job::Manager() const

{

return manager;

}

bool Job::IsCritical() const

{

return isCritical;

}

// -------------------------------------------------------------------------------------------------------------------------

// JobManager

// -------------------------------------------------------------------------------------------------------------------------

JobManager::JobManager(JobProcessor \*const processor)

: processor(processor), numJobsAddedToProcessor(0), isWaitable(false)

{

Assert(processor);

}

JobManager::JobManager(JobProcessor \*const processor, const bool isWaitable)

: processor(processor), numJobsAddedToProcessor(0), isWaitable(isWaitable)

{

Assert(processor);

}

JobProcessor \*JobManager::Processor() const

{

return processor;

}

void JobManager::LastJobProcessed()

{

}

void JobManager::ProcessorThreadSpecificCallBack(PageAllocator \*pageAllocator)

{

}

Job \*JobManager::GetJobToProcessProactively()

{

return 0;

}

bool JobManager::ShouldProcessInForeground(const bool willWaitForJob, const unsigned int numJobsInQueue) const

{

return false;

}

void JobManager::Prioritize(JsUtil::Job \*const job, const bool forceAddJobToProcessor, void\* function) const

{

}

void JobManager::PrioritizedButNotYetProcessed(JsUtil::Job \*const job) const

{

}

void JobManager::OnDecommit(ParallelThreadData \*threadData)

{

}

void JobManager::BeforeWaitForJob(bool) const

{

}

void JobManager::AfterWaitForJob(bool) const

{

}

// -------------------------------------------------------------------------------------------------------------------------

// WaitableJobManager

// -------------------------------------------------------------------------------------------------------------------------

WaitableJobManager::WaitableJobManager(JobProcessor \*const processor)

: JobManager(processor, true),

jobBeingWaitedUpon(0),

jobBeingWaitedUponProcessed(false),

isWaitingForQueuedJobs(false),

queuedJobsProcessed(false)

{

}

// -------------------------------------------------------------------------------------------------------------------------

// SingleJobManager

// -------------------------------------------------------------------------------------------------------------------------

SingleJobManager::SingleJobManager(JobProcessor \*const processor, const bool isCritical)

: JobManager(processor), job(isCritical), processed(false)

{

job.manager = this;

}

void SingleJobManager::AddJobToProcessor(const bool prioritize)

{

AutoOptionalCriticalSection lock(Processor()->GetCriticalSection());

Processor()->AddJob(&job, prioritize);

}

void SingleJobManager::JobProcessed(JsUtil::Job \*const job, const bool succeeded)

{

processed = true;

}

JsUtil::Job \*SingleJobManager::GetJob(bool)

{

return processed ? 0 : &job;

}

bool SingleJobManager::WasAddedToJobProcessor(JsUtil::Job \*const job) const

{

return true;

}

// -------------------------------------------------------------------------------------------------------------------------

// WaitableSingleJobManager

// -------------------------------------------------------------------------------------------------------------------------

WaitableSingleJobManager::WaitableSingleJobManager(JobProcessor \*const processor, const bool isCritical)

: WaitableJobManager(processor), job(isCritical), processed(false)

{

job.manager = this;

}

void WaitableSingleJobManager::AddJobToProcessor(const bool prioritize)

{

AutoOptionalCriticalSection lock(Processor()->GetCriticalSection());

Processor()->AddJob(&job, prioritize);

}

void WaitableSingleJobManager::WaitForJobProcessed()

{

Processor()->PrioritizeJobAndWait(this, false);

}

void WaitableSingleJobManager::JobProcessed(JsUtil::Job \*const job, const bool succeeded)

{

processed = true;

}

JsUtil::Job \*WaitableSingleJobManager::GetJob(bool)

{

return processed ? 0 : &job;

}

bool WaitableSingleJobManager::WasAddedToJobProcessor(JsUtil::Job \*const job) const

{

return true;

}

// -------------------------------------------------------------------------------------------------------------------------

// JobProcessor

// -------------------------------------------------------------------------------------------------------------------------

JobProcessor::JobProcessor(const bool processesInBackground) : processesInBackground(processesInBackground), isClosed(false)

{

}

bool JobProcessor::ProcessesInBackground() const

{

return processesInBackground;

}

bool JobProcessor::IsClosed() const

{

return isClosed;

}

CriticalSection \*JobProcessor::GetCriticalSection()

{

return processesInBackground ? static\_cast<BackgroundJobProcessor \*>(this)->GetCriticalSection() : 0;

}

void JobProcessor::AddManager(JobManager \*const manager)

{

Assert(manager);

Assert(!isClosed);

managers.LinkToEnd(manager);

}

bool JobProcessor::HasManager(JobManager \*const manager) const

{

for (JobManager \*curManager = managers.Head(); curManager != NULL; curManager = curManager->Next())

{

if (manager == curManager)

{

return true;

}

}

return false;

}

template<class Fn>

void JobProcessor::ForEachManager(Fn fn)

{

for (JobManager \*curManager = managers.Head(); curManager != NULL; curManager = curManager->Next())

{

fn(curManager);

}

}

void JobProcessor::PrioritizeManager(JobManager \*const manager)

{

Assert(manager);

Assert(!isClosed);

managers.MoveToBeginning(manager);

if (manager->numJobsAddedToProcessor == 0)

{

return;

}

// Move this manager's jobs to the beginning too. Find sequences of this manager's jobs backwards so that their relative

// order remains intact after the sequences are moved.

Job \*const originalHead = jobs.Head();

Job \*lastJob = 0;

for (Job \*job = jobs.Tail(); job; job = job->Previous())

{

if (job->Manager() == manager)

{

if (!lastJob)

lastJob = job;

}

else if (lastJob)

{

jobs.MoveSubsequenceToBeginning(job->Next(), lastJob);

lastJob = 0;

}

if (job == originalHead)

{

break;

}

}

if (lastJob)

{

jobs.MoveSubsequenceToBeginning(originalHead, lastJob);

}

}

void JobProcessor::AddJob(Job \*const job, const bool prioritize)

{

// This function is called from inside the lock

Assert(job);

Assert(managers.Contains(job->Manager()));

Assert(!IsClosed());

if (job->Manager()->numJobsAddedToProcessor + 1 == 0)

Js::Throw::OutOfMemory(); // Overflow: job counts we use are int32's.

++job->Manager()->numJobsAddedToProcessor;

if (prioritize)

jobs.LinkToBeginning(job);

else

jobs.LinkToEnd(job);

}

bool JobProcessor::RemoveJob(Job \*const job)

{

// This function is called from inside the lock

Assert(job);

Assert(managers.Contains(job->Manager()));

Assert(!IsClosed());

jobs.Unlink(job);

Assert(job->Manager()->numJobsAddedToProcessor != 0);

--job->Manager()->numJobsAddedToProcessor;

return true;

}

void JobProcessor::JobProcessed(JobManager \*const manager, Job \*const job, const bool succeeded)

{

Assert(manager);

Assert(job);

Assert(manager == job->Manager());

BEGIN\_NO\_EXCEPTION

{

manager->JobProcessed(job, succeeded);

}

END\_NO\_EXCEPTION;

}

void JobProcessor::LastJobProcessed(JobManager \*const manager)

{

Assert(manager);

BEGIN\_NO\_EXCEPTION

{

manager->LastJobProcessed();

}

END\_NO\_EXCEPTION;

}

void JobProcessor::Close()

{

isClosed = true;

}

// -------------------------------------------------------------------------------------------------------------------------

// ForegroundJobProcessor

// -------------------------------------------------------------------------------------------------------------------------

ForegroundJobProcessor::ForegroundJobProcessor() : JobProcessor(false)

{

}

void ForegroundJobProcessor::RemoveManager(JobManager \*const manager)

{

Assert(manager);

// Managers must remove themselves. Hence, Close does not remove managers. So, not asserting on !IsClosed().

managers.Unlink(manager);

if (manager->numJobsAddedToProcessor == 0)

return;

// Remove this manager's jobs from the queue

Job \*firstJob = 0;

for (Job \*job = jobs.Head(); job; job = job->Next())

{

if (job->Manager() == manager)

{

if (!firstJob)

firstJob = job;

}

else if (firstJob)

{

jobs.UnlinkSubsequence(firstJob, job->Previous());

for (Job \*removedJob = firstJob; removedJob;)

{

Job \*const next = removedJob->Next();

Assert(!removedJob->IsCritical());

JobProcessed(manager, removedJob, false); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

--manager->numJobsAddedToProcessor;

removedJob = next;

}

firstJob = 0;

}

}

if (firstJob)

{

jobs.UnlinkSubsequenceFromEnd(firstJob);

for (Job \*removedJob = firstJob; removedJob;)

{

Job \*const next = removedJob->Next();

Assert(!removedJob->IsCritical());

JobProcessed(manager, removedJob, false); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

--manager->numJobsAddedToProcessor;

removedJob = next;

}

}

Assert(manager->numJobsAddedToProcessor == 0);

LastJobProcessed(manager);

}

bool ForegroundJobProcessor::Process(Job \*const job)

{

try

{

return job->Manager()->Process(job, 0);

}

catch (Js::JavascriptExceptionObject \*)

{

// Treat OOM or stack overflow to be a non-terminal failure. The foreground job processor processes jobs when the

// jobs are prioritized, on the calling thread. The script would be active (at the time of this writing), so a

// JavascriptExceptionObject would be thrown for OOM or stack overflow.

}

catch (Js::OperationAbortedException)

{

// This can happen for any reason a job needs to be aborted while executing

}

// Any of the above exceptions will result in the job failing. The return value of this function will cause the job

// manager to get a JobProcessed call with succeeded = false, so that it can handle the failure appropriately.

return false;

}

void ForegroundJobProcessor::Close()

{

if (IsClosed())

return;

for (Job \*job = jobs.Head(); job;)

{

Job \*const next = job->Next();

JobManager \*const manager = job->Manager();

JobProcessed(

manager,

job,

job->IsCritical() ? Process(job) : false); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

--manager->numJobsAddedToProcessor;

if (manager->numJobsAddedToProcessor == 0)

LastJobProcessed(manager); // the manager may be deleted during this and should not be used afterwards

job = next;

}

jobs.Clear();

JobProcessor::Close();

}

void ForegroundJobProcessor::AssociatePageAllocator(PageAllocator\* const pageAllocator)

{

// Do nothing

}

void ForegroundJobProcessor::DissociatePageAllocator(PageAllocator\* const pageAllocator)

{

// Do nothing

}

// -------------------------------------------------------------------------------------------------------------------------

// BackgroundJobProcessor

// -------------------------------------------------------------------------------------------------------------------------

void BackgroundJobProcessor::InitializeThreadCount()

{

if (CONFIG\_FLAG(ForceMaxJitThreadCount))

{

this->maxThreadCount = CONFIG\_FLAG(MaxJitThreadCount);

}

else if (AutoSystemInfo::Data.IsLowMemoryProcess())

{

// In a low-memory scenario, don't spin up multiple threads, regardless of how many cores we have.

this->maxThreadCount = 1;

}

else

{

int processorCount = AutoSystemInfo::Data.GetNumberOfPhysicalProcessors();

//There is 2 threads already in play, one UI (main) thread and a GC thread. So subtract 2 from processorCount to account for the same.

this->maxThreadCount = max(1, min(processorCount - 2, CONFIG\_FLAG(MaxJitThreadCount)));

}

}

void BackgroundJobProcessor::InitializeParallelThreadData(AllocationPolicyManager\* policyManager, bool disableParallelThreads)

{

if (!disableParallelThreads)

{

InitializeThreadCount();

}

else

{

this->maxThreadCount = 1;

}

Assert(this->maxThreadCount >= 1);

this->parallelThreadData = HeapNewArrayZ(ParallelThreadData\*, this->maxThreadCount);

for (uint i = 0; i < this->maxThreadCount; i++)

{

this->parallelThreadData[i] = HeapNewNoThrow(ParallelThreadData, policyManager);

if (this->parallelThreadData[i] == nullptr)

{

if (i == 0)

{

HeapDeleteArray(this->maxThreadCount, this->parallelThreadData);

Js::Throw::OutOfMemory();

}

// At least one thread is created, continue

break;

}

this->parallelThreadData[i]->processor = this;

// Make sure to create the thread suspended so the thread handle can be assigned before the thread starts running

this->parallelThreadData[i]->threadHandle = reinterpret\_cast<HANDLE>(\_beginthreadex(0, 0, &StaticThreadProc, this->parallelThreadData[i], CREATE\_SUSPENDED, 0));

if (!this->parallelThreadData[i]->threadHandle)

{

HeapDelete(parallelThreadData[i]);

parallelThreadData[i] = nullptr;

if (i == 0)

{

Js::Throw::OutOfMemory();

}

// At least one thread is created, continue

break;

}

if (ResumeThread(this->parallelThreadData[i]->threadHandle) == static\_cast<DWORD>(-1))

{

CloseHandle(this->parallelThreadData[i]->threadHandle);

HeapDelete(parallelThreadData[i]);

this->parallelThreadData[i] = nullptr;

if (i == 0)

{

Js::Throw::OutOfMemory();

}

// At least one thread is created, continue

break;

}

this->threadCount++;

// Wait for the thread to fully start. This is necessary because Close may be called before the thread starts and if

// Close is called while holding the loader lock during DLL\_THREAD\_DETACH, the thread may be stuck waiting for the

// loader lock for DLL\_THREAD\_ATTACH to start up, and Close would then end up waiting forever, causing a deadlock.

WaitWithThreadForThreadStartedOrClosingEvent(this->parallelThreadData[i]);

this->parallelThreadData[i]->threadStartedOrClosing.Reset(); // after this, the event will be used to wait for the thread to close

#if DBG\_DUMP

if (i < (sizeof(DebugThreadNames) / sizeof(DebugThreadNames[i])))

{

this->parallelThreadData[i]->backgroundPageAllocator.debugName = DebugThreadNames[i];

}

else

{

this->parallelThreadData[i]->backgroundPageAllocator.debugName = L"BackgroundJobProcessor thread";

}

#endif

}

Assert(this->threadCount >= 1);

}

void BackgroundJobProcessor::InitializeParallelThreadDataForThreadServiceCallBack(AllocationPolicyManager\* policyManager)

{

//thread is provided by service callback, no need to create thread here. Currently only one thread in service callback supported.

this->maxThreadCount = 1;

this->parallelThreadData = HeapNewArrayZ(ParallelThreadData \*, this->maxThreadCount);

this->parallelThreadData[0] = HeapNewNoThrow(ParallelThreadData, policyManager);

if (this->parallelThreadData[0] == nullptr)

{

HeapDeleteArray(this->maxThreadCount, this->parallelThreadData);

Js::Throw::OutOfMemory();

}

this->parallelThreadData[0]->processor = this;

this->parallelThreadData[0]->isWaitingForJobs = true;

#if DBG\_DUMP

this->parallelThreadData[0]->backgroundPageAllocator.debugName = L"BackgroundJobProcessor";

#endif

this->threadCount = 1;

return;

}

BackgroundJobProcessor::BackgroundJobProcessor(AllocationPolicyManager\* policyManager, JsUtil::ThreadService \*threadService, bool disableParallelThreads)

: JobProcessor(true),

jobReady(true),

wakeAllBackgroundThreads(false),

numJobs(0),

threadId(GetCurrentThreadContextId()),

threadService(threadService),

threadCount(0),

maxThreadCount(0)

{

if (!threadService->HasCallback())

{

// We don't have a thread service, so create a dedicated thread to handle background jobs.

InitializeParallelThreadData(policyManager, disableParallelThreads);

}

else

{

InitializeParallelThreadDataForThreadServiceCallBack(policyManager);

}

}

BackgroundJobProcessor::~BackgroundJobProcessor()

{

// This should appear to be called from the same thread from which this instance was created

Assert(IsClosed());

if (parallelThreadData)

{

for (unsigned int i = 0; i < this->threadCount; i++)

{

HeapDelete(parallelThreadData[i]);

}

HeapDeleteArray(this->maxThreadCount, parallelThreadData);

}

}

void BackgroundJobProcessor::WaitWithAllThreadsForThreadStartedOrClosingEvent()

{

bool continueWaiting = true;

this->IterateBackgroundThreads([&](ParallelThreadData \*threadData)

{

if (continueWaiting)

{

continueWaiting = WaitWithThreadForThreadStartedOrClosingEvent(threadData);

}

else

{

//one of the thread is terminated, its sure shutdown scenario. Just reset the waitingForjobs.

threadData->isWaitingForJobs = false;

}

return false;

});

}

template<class Fn>

void BackgroundJobProcessor::ForEachManager(Fn fn)

{

AutoCriticalSection lock(&criticalSection);

JobProcessor::ForEachManager(fn);

}

//This function waits on two events jobReady or wakeAllBackgroundThreads

//It first waits for 1sec and if it times out it will decommit the allocator and wait infinitely.

bool BackgroundJobProcessor::WaitForJobReadyOrShutdown(ParallelThreadData \*threadData)

{

const HANDLE handles[] = { jobReady.Handle(), wakeAllBackgroundThreads.Handle() };

//Wait for 1 sec on jobReady and shutdownBackgroundThread events.

unsigned int result = WaitForMultipleObjectsEx(\_countof(handles), handles, false, 1000, false);

while (result == WAIT\_TIMEOUT)

{

if (threadData->CanDecommit())

{

// If its 1sec time out decommit and wait for INFINITE

threadData->backgroundPageAllocator.DecommitNow();

this->ForEachManager([&](JobManager \*manager){

manager->OnDecommit(threadData);

});

result = WaitForMultipleObjectsEx(\_countof(handles), handles, false, INFINITE, false);

}

else

{

result = WaitForMultipleObjectsEx(\_countof(handles), handles, false, 1000, false);

}

}

if (!(result == WAIT\_OBJECT\_0 || result == WAIT\_OBJECT\_0 + 1))

{

Js::Throw::FatalInternalError();

}

return result == WAIT\_OBJECT\_0;

}

bool BackgroundJobProcessor::WaitWithThreadForThreadStartedOrClosingEvent(ParallelThreadData \*parallelThreadData, const unsigned int milliseconds)

{

return WaitWithThread(parallelThreadData, parallelThreadData->threadStartedOrClosing, milliseconds);

}

bool BackgroundJobProcessor::WaitWithThread(ParallelThreadData \*parallelThreadData, const Event &e, const unsigned int milliseconds)

{

const HANDLE handles[] = { e.Handle(), parallelThreadData->threadHandle };

// If we have a thread service, then only wait on the event, not the actual thread handle.

DWORD handleCount = 2;

if (threadService->HasCallback())

{

handleCount = 1;

}

const unsigned int result = WaitForMultipleObjectsEx(handleCount, handles, false, milliseconds, false);

if (!(result == WAIT\_OBJECT\_0 || result == WAIT\_OBJECT\_0 + 1 || (result == WAIT\_TIMEOUT && milliseconds != INFINITE)))

{

Js::Throw::FatalInternalError();

}

if (result == WAIT\_OBJECT\_0 + 1)

{

// Apparently, sometimes the thread dies while waiting for an event. It should only be during process shutdown but

// we can't know because DLL\_PROCESS\_DETACH may not have been called yet, which is bizarre. It seems unclear why

// this happens and this could cause unpredictable behavior since the behavior of this object is undefined if the

// thread is killed arbitrarily, or if there are incoming calls after Close. In any case, uses of this function have

// been ported from BackgroundCodeGenThread. For now, we assume that Close will be called eventually and set the

// state to what it should be before Close is called.

parallelThreadData->isWaitingForJobs = false;

}

return result == WAIT\_OBJECT\_0;

}

void BackgroundJobProcessor::AddManager(JobManager \*const manager)

{

Assert(manager);

IterateBackgroundThreads([&manager](ParallelThreadData \*threadData){

manager->ProcessorThreadSpecificCallBack(threadData->GetPageAllocator());

return false;

});

AutoCriticalSection lock(&criticalSection);

Assert(!IsClosed());

JobProcessor::AddManager(manager);

IndicateNewJob();

}

void BackgroundJobProcessor::IndicateNewJob()

{

Assert(criticalSection.IsLocked());

if(NumberOfThreadsWaitingForJobs ())

{

if (threadService->HasCallback())

{

Assert(this->threadCount == 1);

this->parallelThreadData[0]->isWaitingForJobs = false;

// Reset the thread event, so we can wait for it on shutdown.

this->parallelThreadData[0]->threadStartedOrClosing.Reset();

// Submit a request to the thread service.

bool success = threadService->Invoke(ThreadServiceCallback, this);

if (!success)

{

// The thread service denied our request.

// Leave the job in the queue. If it's needed, it will be processed

// in-thread during PrioritizeJob. Or alternatively, if a subsequent

// thread service request succeeds, this job will be processed then.

this->parallelThreadData[0]->isWaitingForJobs = true;

}

}

else

{

// Signal the background thread to wake up and process jobs.

jobReady.Set();

}

}

}

Job \* BackgroundJobProcessor::GetCurrentJobOfManager(JobManager \*const manager)

{

Assert(criticalSection.IsLocked());

Job \*currentJob = nullptr;

this->IterateBackgroundThreads([&](ParallelThreadData\* threadData)

{

if (!threadData->currentJob)

{

return false;

}

if (threadData->currentJob->Manager() != manager)

{

return false;

}

currentJob = threadData->currentJob;

return true;

}

);

return currentJob;

}

ParallelThreadData \* BackgroundJobProcessor::GetThreadDataFromCurrentJob(Job\* job)

{

Assert(criticalSection.IsLocked());

ParallelThreadData \*currentThreadData = nullptr;

this->IterateBackgroundThreads([&](ParallelThreadData\* threadData)

{

if (!threadData->currentJob)

{

return false;

}

if (threadData->currentJob != job)

{

return false;

}

currentThreadData = threadData;

return true;

}

);

return currentThreadData;

}

void BackgroundJobProcessor::RemoveManager(JobManager \*const manager)

{

Assert(manager);

ParallelThreadData \*threadDataProcessingCurrentJob = nullptr;

WaitableJobManager \*waitableManager;

{

AutoCriticalSection lock(&criticalSection);

// Managers must remove themselves. Hence, Close does not remove managers. So, not asserting on !IsClosed().

managers.Unlink(manager);

if(manager->numJobsAddedToProcessor == 0)

{

Assert(!GetCurrentJobOfManager(manager));

return;

}

// Remove this manager's jobs from the queue

Job \*firstJob = 0;

for(Job \*job = jobs.Head(); job; job = job->Next())

{

if(job->Manager() == manager)

{

if(!firstJob)

firstJob = job;

}

else if(firstJob)

{

jobs.UnlinkSubsequence(firstJob, job->Previous());

for(Job \*removedJob = firstJob; removedJob;)

{

Job \*const next = removedJob->Next();

Assert(!removedJob->IsCritical());

Assert(numJobs != 0);

--numJobs;

JobProcessed(manager, removedJob, false); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

--manager->numJobsAddedToProcessor;

if(manager->isWaitable)

{

WaitableJobManager \*const waitableManager = static\_cast<WaitableJobManager \*>(manager);

if(waitableManager->jobBeingWaitedUpon == removedJob)

{

waitableManager->jobBeingWaitedUponProcessed.Set();

waitableManager->jobBeingWaitedUpon = 0;

}

}

removedJob = next;

}

firstJob = 0;

}

}

if(firstJob)

{

jobs.UnlinkSubsequenceFromEnd(firstJob);

for(Job \*removedJob = firstJob; removedJob;)

{

Job \*const next = removedJob->Next();

Assert(!removedJob->IsCritical());

Assert(numJobs != 0);

--numJobs;

JobProcessed(manager, removedJob, false); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

--manager->numJobsAddedToProcessor;

if(manager->isWaitable)

{

WaitableJobManager \*const waitableManager = static\_cast<WaitableJobManager \*>(manager);

if(waitableManager->jobBeingWaitedUpon == removedJob)

{

waitableManager->jobBeingWaitedUponProcessed.Set();

waitableManager->jobBeingWaitedUpon = 0;

}

}

removedJob = next;

}

}

if(manager->numJobsAddedToProcessor == 0)

{

LastJobProcessed(manager);

return;

}

Assert(manager->numJobsAddedToProcessor >= 1);

Assert(manager->isWaitable);

Assert(GetCurrentJobOfManager(manager));

}

//Wait for all the on going jobs to complete.

criticalSection.Enter();

while (true)

{

Job \*job = GetCurrentJobOfManager(manager);

if (!job)

{

break;

}

waitableManager = static\_cast<WaitableJobManager \*>(manager);

Assert(!waitableManager->jobBeingWaitedUpon);

waitableManager->jobBeingWaitedUpon = job;

waitableManager->jobBeingWaitedUponProcessed.Reset();

threadDataProcessingCurrentJob = GetThreadDataFromCurrentJob(waitableManager->jobBeingWaitedUpon);

criticalSection.Leave();

WaitWithThread(threadDataProcessingCurrentJob, waitableManager->jobBeingWaitedUponProcessed);

criticalSection.Enter();

waitableManager->jobBeingWaitedUpon = 0;

}

criticalSection.Leave();

}

void BackgroundJobProcessor::AddJob(Job \*const job, const bool prioritize)

{

// This function is called from inside the lock

Assert(job);

Assert(managers.Contains(job->Manager()));

Assert(!IsClosed());

if(numJobs + 1 == 0)

Js::Throw::OutOfMemory(); // Overflow: job counts we use are int32's.

++numJobs;

\_\_super::AddJob(job, prioritize);

IndicateNewJob();

}

bool BackgroundJobProcessor::RemoveJob(Job \*const job)

{

// This function is called from inside the lock

Assert(job);

Assert(managers.Contains(job->Manager()));

Assert(!IsClosed());

if (IsBeingProcessed(job))

{

return false;

}

return \_\_super::RemoveJob(job);

}

bool BackgroundJobProcessor::Process(Job \*const job, ParallelThreadData \*threadData)

{

try

{

AUTO\_HANDLED\_EXCEPTION\_TYPE(static\_cast<ExceptionType>(ExceptionType\_OutOfMemory | ExceptionType\_StackOverflow));

return job->Manager()->Process(job, threadData);

}

catch(Js::OutOfMemoryException)

{

// Treat OOM to be a non-terminal failure

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

job->failureReason = Job::FailureReason::OOM;

#endif

}

catch(Js::StackOverflowException)

{

// Treat stack overflow to be a non-terminal failure

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

job->failureReason = Job::FailureReason::StackOverflow;

#endif

}

catch(Js::OperationAbortedException)

{

// This can happen for any reason a job needs to be aborted while executing, like for instance, if the script

// context is closed while the job is being processed in the background

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

job->failureReason = Job::FailureReason::Aborted;

#endif

}

// Since the background job processor processes jobs on a background thread, out-of-memory and stack overflow need to be

// caught here. Script would not be active in the background thread, so (at the time of this writing) a

// JavascriptException would never be thrown and instead the corresponding exceptions caught above would be thrown.

// Any of the above exceptions will result in the job failing. The return value of this function will cause the job

// manager to get a JobProcessed call with succeeded = false, so that it can handle the failure appropriately.

return false;

}

void BackgroundJobProcessor::Run(ParallelThreadData\* threadData)

{

JS\_ETW(EventWriteJSCRIPT\_NATIVECODEGEN\_START(this, 0));

ArenaAllocator threadArena(L"ThreadArena", threadData->GetPageAllocator(), Js::Throw::OutOfMemory);

threadData->threadArena = &threadArena;

{

// Make sure we take decommit action before the threadArena is torn down, in case the

// thread context goes away and the loop exits.

struct AutoDecommit

{

AutoDecommit(JobProcessor \*proc, ParallelThreadData \*data) : processor(proc), threadData(data) {}

~AutoDecommit()

{

processor->ForEachManager([this](JobManager \*manager){

manager->OnDecommit(this->threadData);

});

}

ParallelThreadData \*threadData;

JobProcessor \*processor;

} autoDecommit(this, threadData);

criticalSection.Enter();

while (!IsClosed() || jobs.Head() && jobs.Head()->IsCritical())

{

Job \*job = jobs.UnlinkFromBeginning();

if(!job)

{

// No jobs in queue, wait for one

Assert(!IsClosed());

Assert(!threadData->isWaitingForJobs);

threadData->isWaitingForJobs = true;

criticalSection.Leave();

JS\_ETW(EventWriteJSCRIPT\_NATIVECODEGEN\_STOP(this, 0));

if (threadService->HasCallback())

{

// We have a thread service, so simply return the thread back now.

// When new jobs are submitted, we will be called to process again.

return;

}

WaitForJobReadyOrShutdown(threadData);

JS\_ETW(EventWriteJSCRIPT\_NATIVECODEGEN\_START(this, 0));

criticalSection.Enter();

threadData->isWaitingForJobs = false;

continue;

}

Assert(numJobs != 0);

--numJobs;

threadData->currentJob = job;

criticalSection.Leave();

const bool succeeded = Process(job, threadData);

criticalSection.Enter();

threadData->currentJob = 0;

JobManager \*const manager = job->Manager();

JobProcessed(manager, job, succeeded); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

--manager->numJobsAddedToProcessor;

if(manager->isWaitable)

{

WaitableJobManager \*const waitableManager = static\_cast<WaitableJobManager \*>(manager);

Assert(!(waitableManager->jobBeingWaitedUpon && waitableManager->isWaitingForQueuedJobs));

if(waitableManager->jobBeingWaitedUpon == job)

{

waitableManager->jobBeingWaitedUpon = 0;

waitableManager->jobBeingWaitedUponProcessed.Set();

}

else if(waitableManager->isWaitingForQueuedJobs && manager->numJobsAddedToProcessor == 0)

{

waitableManager->isWaitingForQueuedJobs = false;

waitableManager->queuedJobsProcessed.Set();

}

}

if(manager->numJobsAddedToProcessor == 0)

LastJobProcessed(manager); // the manager may be deleted during this and should not be used afterwards

}

criticalSection.Leave();

JS\_ETW(EventWriteJSCRIPT\_NATIVECODEGEN\_STOP(this, 0));

}

}

void BackgroundJobProcessor::Close()

{

// The contract for Close is that from the time it's called, job managers and jobs may no longer be added to the job

// processor. If there is potential background work that needs to be done after Close, it must be done directly on the

// foreground thread.

if(IsClosed())

return;

bool waitForThread = true;

uint threadsWaitingForJobs = 0;

{

AutoCriticalSection lock(&criticalSection);

if(IsClosed())

return;

Job \*nextJob = jobs.Head();

while(nextJob)

{

Job \*const job = nextJob;

nextJob = job->Next();

if(job->IsCritical())

{

// Critical jobs need to be left in the queue. After this instance is flagged as closed, the background

// thread will continue processing critical jobs, for which this function will wait before returning.

continue;

}

jobs.Unlink(job);

Assert(numJobs != 0);

--numJobs;

JobManager \*const manager = job->Manager();

JobProcessed(manager, job, false); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

--manager->numJobsAddedToProcessor;

if(manager->isWaitable)

{

WaitableJobManager \*const waitableManager = static\_cast<WaitableJobManager \*>(manager);

if(waitableManager->jobBeingWaitedUpon == job)

{

waitableManager->jobBeingWaitedUponProcessed.Set();

waitableManager->jobBeingWaitedUpon = 0;

}

}

if(manager->numJobsAddedToProcessor == 0)

{

Assert(!GetCurrentJobOfManager(manager));

LastJobProcessed(manager); // the manager may be deleted during this and should not be used afterwards

}

}

// Managers will remove themselves, so not removing managers here

JobProcessor::Close();

if (threadService->HasCallback())

{

Assert(this->threadCount == 1);

// If there are no outstanding jobs, then we don't currently have a thread, so there's no reason to wait for it.

waitForThread = !this->parallelThreadData[0]->isWaitingForJobs;

}

else

{

threadsWaitingForJobs = NumberOfThreadsWaitingForJobs ();

}

}

if (threadsWaitingForJobs)

{

//There is no reset for this. It will be signaled until all the threads get out of their hibernation.

wakeAllBackgroundThreads.Set();

}

// We cannot wait for the background thread to terminate because this function may be called from DLL\_THREAD\_DETACH, and

// waiting for the background thread would then deadlock because the background thread would also be blocked from

// detaching. Instead, we just wait for this event to be signaled, which indicates that the thread will promptly end

// naturally. The caller should wait as necessary for the thread to terminate.

if (waitForThread)

{

WaitWithAllThreadsForThreadStartedOrClosingEvent();

}

#if DBG

if (!threadService->HasCallback())

{

AutoCriticalSection lock(&criticalSection);

Assert(!NumberOfThreadsWaitingForJobs());

this->IterateBackgroundThreads([](ParallelThreadData \*threadData)

{

threadData->backgroundPageAllocator.ClearConcurrentThreadId();

return false;

}

);

}

#endif

if (threadService->HasCallback())

{

Assert(this->threadCount == 1 && (this->parallelThreadData[0]->threadHandle == 0));

return;

}

else

{

// Close all the handles

this->IterateBackgroundThreads([&](ParallelThreadData \*threadData)

{

CloseHandle(threadData->threadHandle);

threadData->threadHandle = 0;

return false;

});

}

}

unsigned int WINAPI BackgroundJobProcessor::StaticThreadProc(void \*lpParam)

{

Assert(lpParam);

#if !defined(\_UCRT)

HMODULE dllHandle = NULL;

if (!GetModuleHandleEx(0, AutoSystemInfo::GetJscriptDllFileName(), &dllHandle))

{

dllHandle = NULL;

}

#endif

ParallelThreadData \* threadData = static\_cast<ParallelThreadData \*>(lpParam);

BackgroundJobProcessor \*const processor = threadData->processor;

// Indicate to the constructor that the thread has fully started.

threadData->threadStartedOrClosing.Set();

#if DBG

threadData->backgroundPageAllocator.SetConcurrentThreadId(GetCurrentThreadId());

#endif

\_\_try

{

processor->Run(threadData);

}

\_\_except(ExceptFilter(GetExceptionInformation()))

{

Assert(false);

}

// Indicate to Close that the thread is about to exit. This has to be done before CoUninitialize because CoUninitialize

// may require the loader lock and if Close was called while holding the loader lock during DLL\_THREAD\_DETACH, it could

// end up waiting forever, causing a deadlock.

threadData->threadStartedOrClosing.Set();

#if !defined(\_UCRT)

if (dllHandle)

{

FreeLibraryAndExitThread(dllHandle, 0);

}

else

#endif

{

return 0;

}

}

int BackgroundJobProcessor::ExceptFilter(LPEXCEPTION\_POINTERS pEP)

{

#if DBG

// Assert exception code

if (pEP->ExceptionRecord->ExceptionCode == STATUS\_ASSERTION\_FAILURE)

{

return EXCEPTION\_CONTINUE\_SEARCH;

}

#endif

#ifdef GENERATE\_DUMP

if (Js::Configuration::Global.flags.IsEnabled(Js::DumpOnCrashFlag))

{

Js::Throw::GenerateDump(pEP, Js::Configuration::Global.flags.DumpOnCrash);

}

#endif

#if DBG && \_M\_IX86

int callerEBP = \*((int\*)pEP->ContextRecord->Ebp);

Output::Print(L"BackgroundJobProcessor: Uncaught exception: EIP: 0x%X ExceptionCode: 0x%X EBP: 0x%X ReturnAddress: 0x%X ReturnAddress2: 0x%X\n",

pEP->ExceptionRecord->ExceptionAddress, pEP->ExceptionRecord->ExceptionCode, pEP->ContextRecord->Eip,

pEP->ContextRecord->Ebp, \*((int\*)pEP->ContextRecord->Ebp + 1), \*((int\*) callerEBP + 1));

#endif

Output::Flush();

return EXCEPTION\_CONTINUE\_SEARCH;

}

void BackgroundJobProcessor::ThreadServiceCallback(void \* callbackData)

{

BackgroundJobProcessor \* jobProcessor = (BackgroundJobProcessor \*)callbackData;

Assert(jobProcessor->threadCount == 1);

#if DBG

jobProcessor->parallelThreadData[0]->backgroundPageAllocator.SetConcurrentThreadId(GetCurrentThreadId());

#endif

jobProcessor->Run(jobProcessor->parallelThreadData[0]);

#if DBG

jobProcessor->parallelThreadData[0]->backgroundPageAllocator.ClearConcurrentThreadId();

#endif

// Set the thread event, in case we are waiting for it on shutdown.

jobProcessor->parallelThreadData[0]->threadStartedOrClosing.Set();

}

bool BackgroundJobProcessor::AreAllThreadsWaitingForJobs()

{

Assert(criticalSection.IsLocked());

bool isAnyThreadNotWaitingForJobs = false;

this->IterateBackgroundThreads([&](ParallelThreadData \*parallelThreadData)

{

if (parallelThreadData->isWaitingForJobs)

{

return false;

}

// At least one thread was not waiting for jobs.

isAnyThreadNotWaitingForJobs = true;

return true;

});

return !isAnyThreadNotWaitingForJobs;

}

uint BackgroundJobProcessor::NumberOfThreadsWaitingForJobs ()

{

Assert(criticalSection.IsLocked());

uint countOfThreadsWaitingForJobs = 0;

this->IterateBackgroundThreads([&](ParallelThreadData \*parallelThreadData)

{

if (parallelThreadData->isWaitingForJobs)

{

// At least one thread is waiting for jobs.

countOfThreadsWaitingForJobs++;

}

return false;

});

return countOfThreadsWaitingForJobs;

}

bool BackgroundJobProcessor::IsBeingProcessed(Job\* job)

{

Assert(criticalSection.IsLocked());

bool isBeingProcessed = false;

this->IterateBackgroundThreads([&](ParallelThreadData \*parallelThreadData)

{

if (parallelThreadData->currentJob == job)

{

isBeingProcessed = true;

return true;

}

return false;

});

return isBeingProcessed;

}

void BackgroundJobProcessor::AssociatePageAllocator(PageAllocator\* const pageAllocator)

{

#if DBG

pageAllocator->SetConcurrentThreadId(::GetCurrentThreadId());

#endif

}

void BackgroundJobProcessor::DissociatePageAllocator(PageAllocator\* const pageAllocator)

{

// This function is called from the foreground thread

#if DBG

// Assert that the dissociation is happening in the same thread that created the background job processor

Assert(GetCurrentThreadContextId() == this->threadId);

pageAllocator->ClearConcurrentThreadId();

#endif

}

#if DBG\_DUMP

//Just for debugging purpose

wchar\_t const \* const BackgroundJobProcessor::DebugThreadNames[16] = {

L"BackgroundJobProcessor thread 1",

L"BackgroundJobProcessor thread 2",

L"BackgroundJobProcessor thread 3",

L"BackgroundJobProcessor thread 4",

L"BackgroundJobProcessor thread 5",

L"BackgroundJobProcessor thread 6",

L"BackgroundJobProcessor thread 7",

L"BackgroundJobProcessor thread 8"

L"BackgroundJobProcessor thread 9",

L"BackgroundJobProcessor thread 10",

L"BackgroundJobProcessor thread 11",

L"BackgroundJobProcessor thread 12",

L"BackgroundJobProcessor thread 13",

L"BackgroundJobProcessor thread 14",

L"BackgroundJobProcessor thread 15",

L"BackgroundJobProcessor thread 16" };

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// Undefine name #define in OS headers

#undef AddJob

#undef GetJob

class Parser;

class CompileScriptException;

namespace JsUtil

{

class JobManager;

class WaitableJobManager;

class SingleJobManager;

class WaitableSingleJobManager;

class JobProcessor;

class ForegroundJobProcessor;

class BackgroundJobProcessor;

struct ParallelThreadData;

// -------------------------------------------------------------------------------------------------------------------------

// Job

//

// Base class for jobs that can be sent to a job processor for processing

// -------------------------------------------------------------------------------------------------------------------------

class Job : public DoublyLinkedListElement<Job>

{

friend SingleJobManager;

friend WaitableSingleJobManager;

private:

JobManager \*manager;

// Jobs may be aborted if the job processor is closed while there are still queued jobs, or if a job manager is removed

// while it still has jobs queued to the job processor. Critical jobs are not aborted and rather processed during the

// JobProcessor::Close call. Aborted jobs are not processed and instead the job manager is notified with

// JobManager::JobProcessed(succeeded = false).

const bool isCritical;

private:

Job(const bool isCritical = false);

public:

Job(JobManager \*const manager, const bool isCritical = false);

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

enum class FailureReason

{

NotFailed,

OOM,

StackOverflow,

Aborted

};

FailureReason failureReason;

#endif

public:

JobManager \*Manager() const;

bool IsCritical() const;

};

// -------------------------------------------------------------------------------------------------------------------------

// JobManager

//

// Base class for job managers that provide and know how to process jobs. Implementers who wish to be able to wait for a job

// or job manager should derive from WaitableJobManager.

// -------------------------------------------------------------------------------------------------------------------------

class JobManager : public DoublyLinkedListElement<JobManager>

{

friend WaitableJobManager;

friend JobProcessor;

friend ForegroundJobProcessor;

friend BackgroundJobProcessor;

private:

JobProcessor \*const processor;

unsigned int numJobsAddedToProcessor;

// Only job managers derived from WaitableJobManager support waiting for a job or the job manager's queued jobs

const bool isWaitable;

protected:

JobManager(JobProcessor \*const processor);

private:

JobManager(JobProcessor \*const processor, const bool isWaitable); // only for use by WaitableJobManager

public:

JobProcessor \*Processor() const;

protected:

// Called by the job processor (outside the lock) to process a job. A job manager may choose to return false to indicate

// a failure. Throwing OutOfMemoryException or OperationAbortedException also indicate a processing failure.

// 'pageAllocator' will be null if the job is being processed in the foreground.

virtual bool Process(Job \*const job, ParallelThreadData \*threadData) = 0;

// Called soon after Process by the job processor (inside the lock) to indicate that a job was processed. The

// 'succeeded' parameter will be false if Process returned false, or threw one of the exceptions that trigger a failure

// (see comments for Process). A job manager may choose to delete the job during this call.

virtual void JobProcessed(Job \*const job, const bool succeeded) = 0;

// Called by the job processor (inside the lock) after the last job queued by the job manager to the job processor, is

// processed. A job manager may choose to remove itself from the job processor and optionally delete itself during this

// call, provided that it doesn't happen during a call that passes in the job manager as a parameter. For instance, a

// job manager must not remove or delete itself during any calls to AddManager, RemoveManager, or Prioritize functions.

virtual void LastJobProcessed();

// JobProcessor (particularly background jobProcessor) can invoke multiple threads to process the jobs.

// Manager will get one call back for each spawned thread before the AddManager call completes.

// This is called outside any lock.

virtual void ProcessorThreadSpecificCallBack(PageAllocator \*pageAllocator);

// Called when background thread page allocator is decommitted (currently after 1 idle second).

virtual void OnDecommit(ParallelThreadData \*threadData);

// The following functions are called by the job processor in response to JobProcessor::AddJobAndProcessProactively,

// PrioritizeManagerAndWait, PrioritizeJob, or PrioritizeJobAndWait. They are not virtual functions because the

// aforementioned functions are templates that take the actual type of the manager, so derived job managers that wish to

// use the aforementioned functions should implement these functions (and hence hide the base bare-bones

// implementations).

//

// Job \*GetJobToProcessProactively();

// Called by the job processor as part of PrioritizeManagerAndWait (outside the lock) when its job queue depletes,

// to obtain a job to process proactively. A job manager that does not have a job to provide should return 0. Jobs

// provided by this function will be processed in the calling thread, so the job manager must support this.

//

// Job \*GetJob(const <job holder type> jobHolder) const;

// Called in response to PrioritizeJob or PrioritizeJobAndWait (inside the lock), to get the job from the job

// holder. The purpose of this is that the job processor does not know the lifetime of a job. When a job manager

// asks the job processor to prioritize a job and wait for it, by passing in a job directly, that contract doesn't

// make sense because the job manager must ensure that the job object is not deleted during the call. If the job

// manager wants to delete jobs after they're processed, it cannot guarantee that. So, the job manager provides a

// job holder, and GetJob would then return the associated Job object if it's not yet processed, or 0 if it was

// already processed. This function and JobProcessed are both called from inside the lock to enable deterministic

// determination of whether a job is already processed, and if it hasn't been processed, to ensure that is won't be

// processed until it is prioritized and the lock is released.

//

// bool WasAddedToJobProcessor(JsUtil::Job \*const job) const;

// Called in response to PrioritizeJob or PrioritizeJobAndWait (inside the lock), to determine whether the job to

// prioritize was actually added to the job processor. Again, the job processor does not know this if the job

// manager has other jobs that are not yet queued to the job processor. However, since doing this check and

// prioritizing the job must both happen within a lock, a job manager should always call PrioritizeJob or

// PrioritizeJobAndWait on the job processor to prioritize jobs that have or have not been added to the job

// processor, and then use the return value of this function to indicate to the job processor whether the job was

// added to the job processor.

//

// bool ShouldProcessInForeground(const bool willWaitForJob, const unsigned int numJobsInQueue) const;

// Called by the BackgroundJobProcessor in response to PrioritizeJob or PrioritizeJobAndWait (inside the lock), to

// determine whether the job may be processed in the current (foreground) thread synchronously.

//

// void Prioritize(JsUtil::Job \*const job, const bool forceAddJobToProcessor = false);

// Called in response to PrioritizeJob or PrioritizeJobAndWait (inside the lock), if false is returned from

// WasAddedToJobProcessor. If the initial call was PrioritizeJobAndWait, the job processor will pass

// forceAddToJobProcessor = true since it needs to wait for the job, and the job manager should ensure to add the

// job to the job processor during the Prioritize call in that case.

//

// void PrioritizedButNotYetProcessed(JsUtil::Job \*const job) const;

// Called in response to PrioritizeJob (inside the lock), if the job was not yet processed. May be useful for

// tracking how often the job manager asks to prioritize jobs and the jobs are not yet processed. Although the

// return value of PrioritizeJob also indicates whether the job was already processed, work done in this function

// may need to use the job object, in which case it's necessary to ensure that the job won't be deleted during that

// time, and hence the existence of this function and why it's called inside the lock.

//

// void BeforeWaitForJob(Js::FunctionBody \*const functionBody) const;

// void AfterWaitForJob(Js::FunctionBody \*const functionBody) const;

// Called in response to PrioritizeJobAndWait (outside the lock), before and after waiting for the prioritized job

// to be processed, respectively.

//

// The following are bare-bones implementations of some of the above functions and should be hidden if a job manager

// wishes to provide its own implementation.

Job \*GetJobToProcessProactively();

bool ShouldProcessInForeground(const bool willWaitForJob, const unsigned int numJobsInQueue) const;

void Prioritize(JsUtil::Job \*const job, const bool forceAddJobToProcessor = false, void\* function = nullptr) const;

void PrioritizedButNotYetProcessed(JsUtil::Job \*const job) const;

void BeforeWaitForJob(bool) const;

void AfterWaitForJob(bool) const;

};

// -------------------------------------------------------------------------------------------------------------------------

// WaitableJobManager

//

// Base class for job managers that provide and know how to process jobs, and wish to be able to wait for a job or job

// manager.

// -------------------------------------------------------------------------------------------------------------------------

class WaitableJobManager : public JobManager

{

friend BackgroundJobProcessor;

private:

Job \*jobBeingWaitedUpon;

Event jobBeingWaitedUponProcessed;

bool isWaitingForQueuedJobs;

Event queuedJobsProcessed;

protected:

WaitableJobManager(JobProcessor \*const processor);

};

// -------------------------------------------------------------------------------------------------------------------------

// SingleJobManager

//

// Base class for a job manager that queues a single job, to quickly create the job processing implementation inline and

// queue the job. Since this class derives from JobManager, it does not support waiting and hence, it must be heap-allocated

// and a derived class should override LastJobProcessed to remove itself from the job processor and delete itself. Derived

// classes need only override Process and LastJobProcessed.

// -------------------------------------------------------------------------------------------------------------------------

class SingleJobManager : public JobManager

{

friend ForegroundJobProcessor;

friend BackgroundJobProcessor;

private:

JsUtil::Job job;

bool processed;

protected:

SingleJobManager(JobProcessor \*const processor, const bool isCritical = false);

public:

void AddJobToProcessor(const bool prioritize);

protected:

virtual void JobProcessed(JsUtil::Job \*const job, const bool succeeded) override;

JsUtil::Job \*GetJob(bool);

bool WasAddedToJobProcessor(JsUtil::Job \*const job) const;

};

// -------------------------------------------------------------------------------------------------------------------------

// WaitableSingleJobManager

//

// Base class for a job manager that queues a single job, to quickly create the job processing implementation inline and

// queue the job. This class supports waiting for a job and hence, does not delete itself automatically. As a result, it's

// possible to instantiate this class on the stack, and queue the job and wait for it, all inline without any heap

// allocation. Derived classes need only override the Process function.

// -------------------------------------------------------------------------------------------------------------------------

class WaitableSingleJobManager : public WaitableJobManager

{

friend ForegroundJobProcessor;

friend BackgroundJobProcessor;

private:

JsUtil::Job job;

bool processed;

protected:

WaitableSingleJobManager(JobProcessor \*const processor, const bool isCritical = false);

public:

void AddJobToProcessor(const bool prioritize);

void WaitForJobProcessed();

protected:

virtual void JobProcessed(JsUtil::Job \*const job, const bool succeeded) override;

JsUtil::Job \*GetJob(bool);

bool WasAddedToJobProcessor(JsUtil::Job \*const job) const;

};

// -------------------------------------------------------------------------------------------------------------------------

// JobProcessor

//

// Base class for job processors.

// -------------------------------------------------------------------------------------------------------------------------

class JobProcessor

{

private:

bool processesInBackground;

protected:

DoublyLinkedList<JobManager> managers;

DoublyLinkedList<Job> jobs;

private:

bool isClosed;

protected:

JobProcessor(const bool processesInBackground);

public:

// Ideally, a job manager should not need to depend on this, but there may be cases where it's needed (such as if

// processing jobs needs to support the -profile switch)

bool ProcessesInBackground() const;

// Ideally, a job manager should not need to depend on this, but there may be cases where it's needed (such as if the

// job manager needs to queue a critical cleanup job to the job processor when it's being destructed, it may need to

// check first if the job processor is already closed, because it's illegal then)

bool IsClosed() const;

// A job manager generally should not need to take the lock, as most functions called by the job processor are called

// from inside the lock as necessary. The main exception is when adding a job to the job processor; a job manager must

// call JobProcessor::AddJob inside the lock. Use it sparingly.

CriticalSection \*GetCriticalSection();

// Adds or removes a job manager

virtual void AddManager(JobManager \*const manager);

virtual void RemoveManager(JobManager \*const manager) = 0;

bool HasManager(JobManager \*const manager) const;

template<class Fn> void ForEachManager(Fn fn);

// Prioritizes a job manager, and optionally processes its jobs for a certain amount of time. When a job manager is

// prioritized, its jobs are moved to the front of the queue. If the queue depletes of this job manager's jobs, the job

// processor will call JobManager::GetJobToProcessProactively to proactively process jobs that have not yet been queued,

// until the time limit. See comments in JobManager for details on why templates are used.

void PrioritizeManager(JobManager \*const manager);

template<class TJobManager> void PrioritizeManagerAndWait(

TJobManager \*const manager,

const unsigned int milliseconds = INFINITE);

// Add a job to the queue, and optionally put it in front of the queue. Must be called from inside the lock. A job

// manager should use JobManager::AcquireLock and JobManager::ReleaseLock for this purpose.

virtual void AddJob(Job \*const job, const bool prioritize = false);

// Must be called from inside the lock

virtual bool RemoveJob(Job \*const job);

template<class TJobManager, class TJobHolder>

void AddJobAndProcessProactively(TJobManager \*const jobManager, const TJobHolder holder);

// Prioritizes a job by moving it to the front of the queue. For details on what more happens during this and on why

// templates are used, see comments in JobManager. PrioritizeJob returns true if the job was already processed.

template<class TJobManager, class TJobHolder> bool PrioritizeJob(

TJobManager \*const manager,

const TJobHolder holder,

void\* function = nullptr);

template<class TJobManager, class TJobHolder> void PrioritizeJobAndWait(

TJobManager \*const manager,

const TJobHolder holder,

void\* function = nullptr);

virtual void AssociatePageAllocator(PageAllocator\* const pageAllocator) = 0;

virtual void DissociatePageAllocator(PageAllocator\* const pageAllocator) = 0;

protected:

void JobProcessed(JobManager \*const manager, Job \*const job, const bool succeeded);

void LastJobProcessed(JobManager \*const manager);

public:

// Closes the job processor and closes the handle of background threads.

virtual void Close();

};

// -------------------------------------------------------------------------------------------------------------------------

// ForegroundJobProcessor

// -------------------------------------------------------------------------------------------------------------------------

class ForegroundJobProcessor sealed : public JobProcessor

{

friend BackgroundJobProcessor;

public:

ForegroundJobProcessor();

public:

virtual void RemoveManager(JobManager \*const manager) override;

template<class TJobManager> void PrioritizeManagerAndWait(

TJobManager \*const manager,

const unsigned int milliseconds = INFINITE);

template<class TJobManager, class TJobHolder>

void AddJobAndProcessProactively(TJobManager \*const jobManager, const TJobHolder holder);

template<class TJobManager, class TJobHolder> bool PrioritizeJob(

TJobManager \*const manager,

const TJobHolder holder,

void\* function = nullptr);

template<class TJobManager, class TJobHolder> void PrioritizeJobAndWait(

TJobManager \*const manager,

const TJobHolder holder,

void\* function = nullptr);

private:

// Calls JobManager::Process, handling specific exception types. The return value indicates whether the job succeeded

// (true) or failed (false).

static bool Process(Job \*const job);

virtual void Close() override;

virtual void AssociatePageAllocator(PageAllocator\* const pageAllocator) override;

virtual void DissociatePageAllocator(PageAllocator\* const pageAllocator) override;

};

// -------------------------------------------------------------------------------------------------------------------------

// BackgroundJobProcessor

// -------------------------------------------------------------------------------------------------------------------------

struct ParallelThreadData

{

HANDLE threadHandle;

bool isWaitingForJobs;

bool canDecommit;

Job \*currentJob;

Event threadStartedOrClosing; //This is only used for shutdown scenario to indicate background thread is shutting down or starting

PageAllocator backgroundPageAllocator;

ArenaAllocator \*threadArena;

BackgroundJobProcessor \*processor;

Parser \*parser;

CompileScriptException \*pse;

ParallelThreadData(AllocationPolicyManager\* policyManager) :

threadHandle(0),

isWaitingForJobs(false),

canDecommit(true),

currentJob(nullptr),

threadStartedOrClosing(false),

backgroundPageAllocator(policyManager, Js::Configuration::Global.flags, PageAllocatorType\_BGJIT,

(AutoSystemInfo::Data.IsLowMemoryProcess() ?

PageAllocator::DefaultLowMaxFreePageCount :

PageAllocator::DefaultMaxFreePageCount)),

threadArena(nullptr),

processor(nullptr),

parser(nullptr),

pse(nullptr)

{

}

PageAllocator\* const GetPageAllocator() { return &backgroundPageAllocator; }

bool CanDecommit() const { return canDecommit; }

};

class BackgroundJobProcessor sealed : public JobProcessor

{

private:

CriticalSection criticalSection;

Event jobReady; //This is an auto reset event, only one thread wakes up when the event is signaled.

Event wakeAllBackgroundThreads; //This is a manual reset event.

unsigned int numJobs;

ThreadContextId threadId;

ThreadService \*threadService;

unsigned int threadCount;

unsigned int maxThreadCount;

ParallelThreadData \*\*parallelThreadData;

#if DBG\_DUMP

static wchar\_t const \* const DebugThreadNames[16];

#endif

public:

BackgroundJobProcessor(AllocationPolicyManager\* policyManager, ThreadService \*threadService, bool disableParallelThreads);

~BackgroundJobProcessor();

private:

bool WaitWithThread(ParallelThreadData \*parallelThreadData, const Event &e, const unsigned int milliseconds = INFINITE);

bool WaitWithThreadForThreadStartedOrClosingEvent(ParallelThreadData \*parallelThreadData, const unsigned int milliseconds = INFINITE);

void WaitWithAllThreadsForThreadStartedOrClosingEvent();

bool WaitForJobReadyOrShutdown(ParallelThreadData \*threadData); //Returns true for JobReady event is signaled.

void IndicateNewJob();

bool AreAllThreadsWaitingForJobs();

uint NumberOfThreadsWaitingForJobs ();

Job\* GetCurrentJobOfManager(JobManager \*const manager);

ParallelThreadData \* GetThreadDataFromCurrentJob(Job\* job);

void InitializeThreadCount();

void InitializeParallelThreadData(AllocationPolicyManager\* policyManager, bool disableParallelThreads);

void InitializeParallelThreadDataForThreadServiceCallBack(AllocationPolicyManager\* policyManager);

public:

virtual void AddManager(JobManager \*const manager) override;

virtual void RemoveManager(JobManager \*const manager) override;

template<class Fn> void ForEachManager(Fn fn); //This takes lock for criticalSection

template<class TJobManager> void PrioritizeManagerAndWait(

TJobManager \*const manager,

const unsigned int milliseconds = INFINITE);

virtual void AddJob(Job \*const job, const bool prioritize = false) override;

virtual bool RemoveJob(Job \*const job) override;

template<class TJobManager, class TJobHolder>

void AddJobAndProcessProactively(TJobManager \*const jobManager, const TJobHolder holder);

template<class TJobManager, class TJobHolder> bool PrioritizeJob(

TJobManager \*const manager,

const TJobHolder holder,

void\* function = nullptr);

template<class TJobManager, class TJobHolder> void PrioritizeJobAndWait(

TJobManager \*const manager,

const TJobHolder holder,

void\* function = nullptr);

// Calls JobManager::Process, handling specific exception types. The return value indicates whether the job succeeded

// (true) or failed (false).

bool Process(Job \*const job, ParallelThreadData \*threadData);

bool IsBeingProcessed(Job \*job);

CriticalSection \* GetCriticalSection() { return &criticalSection; }

//Iterates each background thread, callback returns true when it needs to terminate the iteration.

template<class Fn>

bool IterateBackgroundThreads(Fn callback)

{

for (uint i = 0; i < this->threadCount; i++)

{

if (callback(this->parallelThreadData[i]))

{

return false;

}

}

return true;

}

private:

void Run(ParallelThreadData \*threadData);

public:

virtual void Close() override;

virtual void AssociatePageAllocator(PageAllocator\* const pageAllocator) override;

virtual void DissociatePageAllocator(PageAllocator\* const pageAllocator) override;

private:

static unsigned int WINAPI StaticThreadProc(void \*lpParam);

static int ExceptFilter(LPEXCEPTION\_POINTERS pEP);

static void CALLBACK ThreadServiceCallback(void \* callbackData);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

// -------------------------------------------------------------------------------------------------------------------------

// JobProcessor

// -------------------------------------------------------------------------------------------------------------------------

template<class TJobManager>

void JobProcessor::PrioritizeManagerAndWait(TJobManager \*const manager, const unsigned int milliseconds)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, WaitableJobManager>;

Assert(manager);

Assert(!isClosed);

if(processesInBackground)

static\_cast<BackgroundJobProcessor \*>(this)->PrioritizeManagerAndWait(manager, milliseconds);

else

static\_cast<ForegroundJobProcessor \*>(this)->PrioritizeManagerAndWait(manager, milliseconds);

}

template<class TJobManager, class TJobHolder>

void JobProcessor::AddJobAndProcessProactively(TJobManager \*const manager, const TJobHolder holder)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, JobManager>;

Assert(manager);

Assert(!isClosed);

if(processesInBackground)

return static\_cast<BackgroundJobProcessor \*>(this)->AddJobAndProcessProactively(manager, holder);

else

return static\_cast<ForegroundJobProcessor \*>(this)->AddJobAndProcessProactively(manager, holder);

}

template<class TJobManager, class TJobHolder>

bool JobProcessor::PrioritizeJob(TJobManager \*const manager, const TJobHolder holder, void\* function)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, JobManager>;

Assert(manager);

Assert(!isClosed);

if(processesInBackground)

return static\_cast<BackgroundJobProcessor \*>(this)->PrioritizeJob(manager, holder, function);

else

return static\_cast<ForegroundJobProcessor \*>(this)->PrioritizeJob(manager, holder, function);

}

template<class TJobManager, class TJobHolder>

void JobProcessor::PrioritizeJobAndWait(TJobManager \*const manager, const TJobHolder holder, void\* function)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, WaitableJobManager>;

Assert(manager);

Assert(!isClosed);

if(processesInBackground)

static\_cast<BackgroundJobProcessor \*>(this)->PrioritizeJobAndWait(manager, holder, function);

else

static\_cast<ForegroundJobProcessor \*>(this)->PrioritizeJobAndWait(manager, holder, function);

}

// -------------------------------------------------------------------------------------------------------------------------

// ForegroundJobProcessor

// -------------------------------------------------------------------------------------------------------------------------

template<class TJobManager, class TJobHolder>

void ForegroundJobProcessor::AddJobAndProcessProactively(TJobManager \*const manager, const TJobHolder holder)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, JobManager>;

Assert(manager);

Assert(!IsClosed());

Job \*const job = manager->GetJob(holder);

Assert(job);

manager->BeforeWaitForJob(holder);

JobProcessed(manager, job, Process(job)); // the job may be deleted during this and should not be used afterwards

if(manager->numJobsAddedToProcessor == 0)

LastJobProcessed(manager);

manager->AfterWaitForJob(holder);

}

template<class TJobManager>

void ForegroundJobProcessor::PrioritizeManagerAndWait(TJobManager \*const manager, const unsigned int milliseconds)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, WaitableJobManager>;

Assert(manager);

Assert(manager->isWaitable);

Assert(!IsClosed());

PrioritizeManager(manager);

if(milliseconds == 0)

return;

// We have been given some time to process jobs proactively, so process as many jobs as possible, trying not to exceed

// the specified amount of time

Js::Tick startTick = Js::Tick::Now();

Js::TickDelta endTickDelta = Js::TickDelta::FromMicroseconds((\_\_int64)milliseconds \* 1000);

do

{

if(manager->numJobsAddedToProcessor != 0)

{

// Process only jobs from this manager

Job \*job = jobs.Head();

for(; job && job->Manager() != manager; job = job->Next());

if(job)

{

jobs.Unlink(job);

JobProcessed(manager, job, Process(job)); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

if(--manager->numJobsAddedToProcessor == 0)

LastJobProcessed(manager);

continue;

}

}

// No jobs in queue, check this job manager for more jobs

Job \*const job = manager->GetJobToProcessProactively();

if(job)

{

JobProcessed(manager, job, Process(job)); // the job may be deleted during this and should not be used afterwards

continue;

}

// No jobs from job managers either

break;

} while (milliseconds == INFINITE || Js::Tick::Now() - startTick < endTickDelta);

}

template<class TJobManager, class TJobHolder>

bool ForegroundJobProcessor::PrioritizeJob(TJobManager \*const manager, const TJobHolder holder, void\* function)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, JobManager>;

Assert(manager);

Assert(!IsClosed());

Job \*const job = manager->GetJob(holder);

if(!job)

return true; // job was processed

Assert(job->Manager() == manager);

if(!manager->WasAddedToJobProcessor(job))

{

// The job wasn't added for processing, so ask the manager to prioritize it

manager->Prioritize(job, false, function);

manager->PrioritizedButNotYetProcessed(job);

return false;

}

jobs.Unlink(job);

manager->BeforeWaitForJob(holder);

JobProcessed(manager, job, Process(job)); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

if(--manager->numJobsAddedToProcessor == 0)

LastJobProcessed(manager);

manager->AfterWaitForJob(holder);

return true;

}

template<class TJobManager, class TJobHolder>

void ForegroundJobProcessor::PrioritizeJobAndWait(TJobManager \*const manager, const TJobHolder holder, void\* function)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, WaitableJobManager>;

Assert(manager);

Assert(manager->isWaitable);

Assert(!IsClosed());

Job \*const job = manager->GetJob(holder);

if(!job)

return; // job was processed

Assert(job->Manager() == manager);

if(!manager->WasAddedToJobProcessor(job))

{

// The job wasn't added for processing, so ask the manager to prioritize it and add the job to the job processor

// since we need to process it and wait for it

manager->Prioritize(job, /\*forceaddToProcessor\*/ true, function);

Assert(manager->WasAddedToJobProcessor(job));

}

jobs.Unlink(job);

manager->BeforeWaitForJob(holder);

JobProcessed(manager, job, Process(job)); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

if(--manager->numJobsAddedToProcessor == 0)

LastJobProcessed(manager);

manager->AfterWaitForJob(holder);

}

// -------------------------------------------------------------------------------------------------------------------------

// BackgroundJobProcessor

// -------------------------------------------------------------------------------------------------------------------------

template<class TJobManager, class TJobHolder>

void BackgroundJobProcessor::AddJobAndProcessProactively(TJobManager \*const manager, const TJobHolder holder)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, JobManager>;

Assert(manager);

Assert(!IsClosed());

Job \*const job = manager->GetJob(holder);

AddJob(job, /\*prioritize\*/ true);

}

template<class TJobManager>

void BackgroundJobProcessor::PrioritizeManagerAndWait(TJobManager \*const manager, const unsigned int milliseconds)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, WaitableJobManager>;

Assert(manager);

Assert(manager->isWaitable);

bool waitForQueuedJobs;

{

AutoCriticalSection lock(&criticalSection);

Assert(!IsClosed());

waitForQueuedJobs = manager->numJobsAddedToProcessor != 0;

if(waitForQueuedJobs)

{

PrioritizeManager(manager);

if (threadService->HasCallback() && AreAllThreadsWaitingForJobs())

{

// Thread service denied our background request, so we must process in foreground.

waitForQueuedJobs = false;

}

Assert(!manager->isWaitingForQueuedJobs);

manager->isWaitingForQueuedJobs = true;

manager->queuedJobsProcessed.Reset();

}

}

Js::Tick startTick = Js::Tick::Now();

Js::TickDelta endTickDelta = Js::TickDelta::FromMicroseconds((\_\_int64)milliseconds \* 1000);

if(waitForQueuedJobs)

{

// Wait for the event, background thread should be alive

const bool timeout = !(manager->queuedJobsProcessed.Wait(milliseconds));

manager->isWaitingForQueuedJobs = false;

if (timeout)

{

return;

}

}

if (milliseconds == 0)

{

return;

}

// We have been given some time to process jobs proactively, so process as many jobs as possible, trying not to exceed

// the specified amount of time

do

{

Job \* job = NULL;

if(!waitForQueuedJobs && manager->numJobsAddedToProcessor != 0)

{

AutoCriticalSection lock(&criticalSection);

// Process only jobs from this manager

job = jobs.Head();

for(; job && job->Manager() != manager; job = job->Next());

if(job)

{

jobs.Unlink(job);

}

}

if (job == NULL)

{

job = manager->GetJobToProcessProactively();

if(!job)

break;

}

const bool succeeded = ForegroundJobProcessor::Process(job);

{

AutoCriticalSection lock(&criticalSection);

manager->JobProcessed(job, succeeded); // the job may be deleted during this and should not be used afterwards

if (!waitForQueuedJobs && manager->numJobsAddedToProcessor != 0)

{

Assert(manager->numJobsAddedToProcessor != 0);

if(--manager->numJobsAddedToProcessor == 0)

LastJobProcessed(manager);

}

}

} while (milliseconds == INFINITE || Js::Tick::Now() - startTick < endTickDelta);

}

template<class TJobManager, class TJobHolder>

bool BackgroundJobProcessor::PrioritizeJob(TJobManager \*const manager, const TJobHolder holder, void\* function)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, JobManager>;

Assert(manager);

Assert(!IsClosed());

// Fast, nondeterministic check to see if the job was already processed, without using a memory barrier or lock

Job \*job = manager->GetJob(holder);

if(!job)

{

// The memory barrier ensures that other state changes made in JobManager::JobProcessed, before nullifying the job

// in the job holder, are up to date when this function returns

MemoryBarrier();

return true; // job was processed

}

{

AutoCriticalSection lock(&criticalSection);

Assert(!IsClosed());

// Get the job again inside the lock to deterministically verify whether the job was already processed, and if not,

// that the job won't be processed until the lock is released

job = manager->GetJob(holder);

if (!job)

{

return true;

}

Assert(job->Manager() == manager);

if(!manager->WasAddedToJobProcessor(job))

{

// The job wasn't added for processing, so ask the manager to prioritize it

manager->Prioritize(job, false, function);

manager->PrioritizedButNotYetProcessed(job);

return false;

}

if (IsBeingProcessed(job))

{

manager->PrioritizedButNotYetProcessed(job);

return false;

}

// If isWaitingForJobs is true, then we have failed a thread service request.

// So we want to force processing in-thread here.

// Otherwise, ask the manager whether to force it in thread or not.

bool forcedInThread = (threadService->HasCallback() && this->parallelThreadData[0]->isWaitingForJobs);

if (!forcedInThread && !manager->ShouldProcessInForeground(false, numJobs))

{

jobs.MoveToBeginning(job);

manager->PrioritizedButNotYetProcessed(job);

return false;

}

jobs.Unlink(job);

Assert(numJobs != 0);

--numJobs;

}

manager->BeforeWaitForJob(holder);

const bool succeeded = ForegroundJobProcessor::Process(job);

{

AutoCriticalSection lock(&criticalSection);

JobProcessed(manager, job, succeeded); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

if(--manager->numJobsAddedToProcessor == 0)

LastJobProcessed(manager);

}

manager->AfterWaitForJob(holder);

return true;

}

template<class TJobManager, class TJobHolder>

void BackgroundJobProcessor::PrioritizeJobAndWait(TJobManager \*const manager, const TJobHolder holder, void\* function)

{

TemplateParameter::SameOrDerivedFrom<TJobManager, WaitableJobManager>;

Assert(manager);

Assert(manager->isWaitable);

Assert(!IsClosed());

// Fast, nondeterministic check to see if the job was already processed, without using a memory barrier or lock

Job \*job = manager->GetJob(holder);

if(!job)

{

// The memory barrier ensures that other state changes made in JobManager::JobProcessed, before nullifying the job

// in the job holder, are up to date when this function returns

MemoryBarrier();

return; // job was processed

}

bool processInForeground = false;

{

AutoCriticalSection lock(&criticalSection);

Assert(!IsClosed());

// Get the job again inside the lock to deterministically verify whether the job was already processed, and if not,

// that the job won't be processed until the lock is released

job = manager->GetJob(holder);

if(!job)

return; // job was processed

Assert(job->Manager() == manager);

if(!manager->WasAddedToJobProcessor(job))

{

// The job wasn't added for processing, so ask the manager to prioritize it and add the job to the job processor

// since we need to process it and wait for it

manager->Prioritize(job, true /\* forceAddJobToProcessor \*/, function);

Assert(manager->WasAddedToJobProcessor(job));

processInForeground = manager->ShouldProcessInForeground(true, numJobs);

if(processInForeground)

jobs.Unlink(job);

}

else if (!IsBeingProcessed(job))

{

// If isWaitingForJobs is true, then we have failed a thread service request.

// So we want to force processing in-thread here.

// Otherwise, ask the manager whether to force it in thread or not.

bool forcedInThread = (threadService->HasCallback() && this->parallelThreadData[0]->isWaitingForJobs);

if (forcedInThread || manager->ShouldProcessInForeground(true, numJobs))

{

jobs.Unlink(job);

processInForeground = true;

}

}

if(processInForeground)

{

Assert(numJobs != 0);

--numJobs;

Assert(!jobs.Contains(job));

}

else

{

if (!IsBeingProcessed(job))

{

jobs.MoveToBeginning(job);

}

Assert(!manager->jobBeingWaitedUpon);

manager->jobBeingWaitedUpon = job;

manager->jobBeingWaitedUponProcessed.Reset();

}

}

if(processInForeground)

{

manager->BeforeWaitForJob(holder);

const bool succeeded = ForegroundJobProcessor::Process(job);

{

AutoCriticalSection lock(&criticalSection);

JobProcessed(manager, job, succeeded); // the job may be deleted during this and should not be used afterwards

Assert(manager->numJobsAddedToProcessor != 0);

if(--manager->numJobsAddedToProcessor == 0)

LastJobProcessed(manager);

}

manager->AfterWaitForJob(holder);

return;

}

manager->BeforeWaitForJob(holder);

//WaitWithThread(GetThreadDataFromCurrentJob(manager->jobBeingWaitedUpon), manager->jobBeingWaitedUponProcessed);

manager->jobBeingWaitedUponProcessed.Wait();

manager->jobBeingWaitedUpon = 0;

// No need for memory barrier to indicate to the background thread immediately that this job manager is no longer

// waiting for the job because the wait is indefinite, and a job is processed only once

manager->AfterWaitForJob(holder);

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include "math.h"

bool

Math::FitsInDWord(size\_t value)

{

return ((size\_t)(signed int)(value & 0xFFFFFFFF) == value);

}

uint32

Math::NextPowerOf2(uint32 n)

{

n = n - 1;

n = n | (n >> 1);

n = n | (n >> 2);

n = n | (n >> 4);

n = n | (n >> 8);

n = n | (n >> 16);

n++;

return n;

}

UINT\_PTR

Math::Rand()

{

unsigned int rand;

rand\_s(&rand);

UINT\_PTR newRand = static\_cast<UINT\_PTR>(rand);

#if TARGET\_64

rand\_s(&rand);

newRand |= static\_cast<UINT\_PTR>(rand) << 32;

#endif

return newRand;

}

\_\_declspec(noreturn) void Math::DefaultOverflowPolicy()

{

Js::Throw::OutOfMemory();

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

///---------------------------------------------------------------------------

///

/// class Math

///

///---------------------------------------------------------------------------

class Math

{

public:

// Explicit cast to integral (may truncate). Avoids warning C4302 'type cast': truncation

template <typename T>

static T PointerCastToIntegralTruncate(void \* pointer)

{

return (T)(uintptr)pointer;

}

// Explicit cast to integral. Assert that it doesn't truncate. Avoids warning C4302 'type cast': truncation

template <typename T>

static T PointerCastToIntegral(void \* pointer)

{

T value = PointerCastToIntegralTruncate<T>(pointer);

Assert((uintptr)value == (uintptr)pointer);

return value;

}

static bool FitsInDWord(size\_t value);

static UINT\_PTR Rand();

static bool IsPow2(int32 val) { return (val > 0 && ((val-1) & val) == 0); }

static uint32 NextPowerOf2(uint32 n);

// Use for compile-time evaluation of powers of 2

template<uint32 val> struct Is

{

static const bool Pow2 = ((val-1) & val) == 0;

};

// Defined in the header so that the Recycler static lib doesn't

// need to pull in jscript.common.common.lib

static uint32 Log2(uint32 value)

{

int i;

for (i = 0; value >>= 1; i++);

return i;

}

// Define a couple of overflow policies for the UInt32Math routines.

// The default policy for overflow is to throw an OutOfMemory exception

\_\_declspec(noreturn) static void DefaultOverflowPolicy();

// A functor (class with operator()) which records whether a the calculation

// encountered an overflow condition.

class RecordOverflowPolicy

{

bool fOverflow;

public:

RecordOverflowPolicy() : fOverflow(false)

{

}

// Called when an overflow is detected

void operator()()

{

fOverflow = true;

}

bool HasOverflowed() const

{

return fOverflow;

}

};

template <typename T>

static T Align(T size, T alignment)

{

return ((size + (alignment-1)) & ~(alignment-1));

}

template <typename T, class Func>

static T AlignOverflowCheck(T size, T alignment, \_\_inout Func& overflowFn)

{

Assert(size >= 0);

T alignSize = Align(size, alignment);

if (alignSize < size)

{

overflowFn();

}

return alignSize;

}

template <typename T>

static T AlignOverflowCheck(T size, T alignment)

{

return AlignOverflowCheck(size, alignment, DefaultOverflowPolicy);

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include "Common\UInt32Math.h"

#include "common\NumberUtilities.inl"

namespace Js

{

const double NumberConstants::MAX\_VALUE = \*(double\*)(&NumberConstants::k\_PosMax);

const double NumberConstants::MIN\_VALUE = \*(double\*)(&NumberConstants::k\_PosMin);

const double NumberConstants::NaN = \*(double\*)(&NumberConstants::k\_Nan);

const double NumberConstants::NEGATIVE\_INFINITY= \*(double\*)(&NumberConstants::k\_NegInf);

const double NumberConstants::POSITIVE\_INFINITY= \*(double\*)(&NumberConstants::k\_PosInf );

const double NumberConstants::NEG\_ZERO= \*(double\*)(&NumberConstants::k\_NegZero );

// These are used in 128-bit operations in the JIT and inline asm

\_\_declspec(align(16)) const BYTE NumberConstants::AbsDoubleCst[] =

{ 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F,

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F };

\_\_declspec(align(16)) const BYTE NumberConstants::AbsFloatCst[] =

{ 0xFF, 0xFF, 0xFF, 0x7F,

0xFF, 0xFF, 0xFF, 0x7F,

0xFF, 0xFF, 0xFF, 0x7F,

0xFF, 0xFF, 0xFF, 0x7F };

\_\_declspec(align(16)) double const NumberConstants::UIntConvertConst[2] = { 0, 4294967296.000000 };

\_\_declspec(align(16)) float const NumberConstants::MaskNegFloat[] = { -0.0f, -0.0f, -0.0f, -0.0f };

\_\_declspec(align(16)) double const NumberConstants::MaskNegDouble[] = { -0.0, -0.0 };

bool NumberUtilities::IsDigit(int ch)

{

return ch >= '0' && ch <= '9';

}

BOOL NumberUtilities::FHexDigit(wchar\_t ch, int \*pw)

{

if ((ch -= '0') <= 9)

{

\*pw = ch;

return TRUE;

}

if ((ch -= 'A' - '0') <= 5 || (ch -= 'a' - 'A') <= 5)

{

\*pw = 10 + ch;

return TRUE;

}

return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Multiply two unsigned longs. Return the low ulong and fill \*pluHi with

the high ulong.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma warning(push)

#pragma warning(disable:4035) // Turn off warning that there is no return value

ulong NumberUtilities::MulLu(ulong lu1, ulong lu2, ulong \*pluHi)

{

#if \_WIN32 || \_WIN64

#if I386\_ASM

\_\_asm

{

mov eax, lu1

mul lu2

mov ebx, pluHi

mov DWORD PTR[ebx], edx

}

#else //!I386\_ASM

DWORDLONG llu = UInt32x32To64(lu1, lu2);

\*pluHi = (ulong)(llu >> 32);

return (ulong)llu;

#endif //!I386\_ASM

#else

#error Neither \_WIN32, nor \_WIN64 is defined

#endif

}

#pragma warning(pop)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Add two unsigned longs and return the carry bit.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int NumberUtilities::AddLu(ulong \*plu1, ulong lu2)

{

\*plu1 += lu2;

return \*plu1 < lu2;

}

bool NumberUtilities::IsFinite(double value)

{

#if defined(\_M\_X64\_OR\_ARM64)

return 0 != (~(ToSpecial(value)) & 0x7FF0000000000000ull);

#else

return 0 != (~Js::NumberUtilities::LuHiDbl(value) & 0x7FF00000);

#endif

}

int NumberUtilities::CbitZeroLeft(ulong lu)

{

int cbit = 0;

if (0 == (lu & 0xFFFF0000))

{

cbit += 16;

lu <<= 16;

}

if (0 == (lu & 0xFF000000))

{

cbit += 8;

lu <<= 8;

}

if (0 == (lu & 0xF0000000))

{

cbit += 4;

lu <<= 4;

}

if (0 == (lu & 0xC0000000))

{

cbit += 2;

lu <<= 2;

}

if (0 == (lu & 0x80000000))

{

cbit += 1;

lu <<= 1;

}

Assert(lu & 0x80000000);

return cbit;

}

charcount\_t NumberUtilities::UInt16ToString(uint16 integer, \_\_out \_\_ecount(outBufferSize) WCHAR\* outBuffer, charcount\_t outBufferSize, char widthForPaddingZerosInsteadSpaces)

{

// inlined here

WORD digit;

charcount\_t cchWritten = 0;

Assert(cchWritten < outBufferSize);

// word is 0 to 65,535 -- 5 digits max

if (cchWritten < outBufferSize)

{

if (integer >= 10000)

{

digit = integer / 10000;

integer %= 10000;

\*outBuffer = digit + L'0';

outBuffer++;

cchWritten++;

}

else if( widthForPaddingZerosInsteadSpaces > 4 )

{

\*outBuffer = L'0';

outBuffer++;

cchWritten++;

}

}

Assert(cchWritten < outBufferSize);

if (cchWritten < outBufferSize)

{

if (integer >= 1000)

{

digit = integer / 1000;

integer %= 1000;

\*outBuffer = digit + L'0';

outBuffer++;

cchWritten++;

}

else if( widthForPaddingZerosInsteadSpaces > 3 )

{

\*outBuffer = L'0';

outBuffer++;

cchWritten++;

}

}

Assert(cchWritten < outBufferSize);

if (cchWritten < outBufferSize)

{

if (integer >= 100)

{

digit = integer / 100;

integer %= 100;

\*outBuffer = digit + L'0';

outBuffer++;

cchWritten++;

}

else if( widthForPaddingZerosInsteadSpaces > 2 )

{

\*outBuffer = L'0';

outBuffer++;

cchWritten++;

}

}

Assert(cchWritten < outBufferSize);

if (cchWritten < outBufferSize)

{

if (integer >= 10)

{

digit = integer / 10;

integer %= 10;

\*outBuffer = digit + L'0';

outBuffer++;

cchWritten++;

}

else if( widthForPaddingZerosInsteadSpaces > 1 )

{

\*outBuffer = L'0';

outBuffer++;

cchWritten++;

}

}

Assert(cchWritten < outBufferSize);

if (cchWritten < outBufferSize)

{

\*outBuffer = integer + L'0';

outBuffer++;

cchWritten++;

}

Assert(cchWritten < outBufferSize);

if (cchWritten < outBufferSize)

{

// cchWritten doesn't include the terminating char, like swprintf\_s

\*outBuffer = 0;

}

return cchWritten;

}

BOOL NumberUtilities::TryConvertToUInt32(const wchar\_t\* str, int length, uint32\* intVal)

{

if (length <= 0 || length > 10)

{

return false;

}

if (length == 1)

{

if (str[0] >= L'0' && str[0] <= L'9')

{

\*intVal = (uint32)(str[0] - L'0');

return true;

}

else

{

return false;

}

}

if (str[0] < L'1' || str[0] > L'9')

{

return false;

}

uint32 val = (uint32)(str[0] - L'0');

int calcLen = min(length, 9);

for (int i = 1; i < calcLen; i++)

{

if ((str[i] < L'0')|| (str[i] > L'9'))

{

return false;

}

val = (val \* 10) + (uint32)(str[i] - L'0');

}

if (length == 10)

{

// check for overflow 4294967295

if (str[9] < L'0' || str[9] > L'9' ||

UInt32Math::Mul(val, 10, &val) ||

UInt32Math::Add(val, (uint32)(str[9] - L'0'), &val))

{

return false;

}

}

\*intVal = val;

return true;

}

double NumberUtilities::Modulus(double dblLeft, double dblRight)

{

double value = 0;

if (!Js::NumberUtilities::IsFinite(dblRight))

{

if (NumberUtilities::IsNan(dblRight) || !Js::NumberUtilities::IsFinite(dblLeft))

{

value = NumberConstants::NaN;

}

else

{

value = dblLeft;

}

}

else if (0 == dblRight || NumberUtilities::IsNan(dblLeft))

{

value = NumberConstants::NaN;

}

else if (0 == dblLeft)

{

value = dblLeft;

}

else

{

value = fmod(dblLeft, dblRight);

}

return value;

}

long NumberUtilities::LwFromDblNearest(double dbl)

{

if (Js::NumberUtilities::IsNan(dbl))

return 0;

if (dbl > 0x7FFFFFFFL)

return 0x7FFFFFFFL;

if (dbl < (long)0x80000000L)

return (long)0x80000000L;

return (long)dbl;

}

ulong NumberUtilities::LuFromDblNearest(double dbl)

{

if (Js::NumberUtilities::IsNan(dbl))

return 0;

if (dbl >(ulong)0xFFFFFFFFUL)

return (ulong)0xFFFFFFFFUL;

if (dbl < 0)

return 0;

return (ulong)dbl;

}

BOOL NumberUtilities::FDblIsLong(double dbl, long \*plw)

{

AssertMem(plw);

double dblT;

\*plw = (long)dbl;

dblT = (double)\*plw;

return Js::NumberUtilities::LuHiDbl(dblT) == Js::NumberUtilities::LuHiDbl(dbl) && Js::NumberUtilities::LuLoDbl(dblT) == Js::NumberUtilities::LuLoDbl(dbl);

}

template<typename EncodedChar>

double NumberUtilities::DblFromHex(const EncodedChar \*psz, const EncodedChar \*\*ppchLim)

{

double dbl;

uint uT;

byte bExtra;

int cbit;

// Skip leading zeros.

while (\*psz == '0')

psz++;

dbl = 0;

Assert(Js::NumberUtilities::LuHiDbl(dbl) == 0);

Assert(Js::NumberUtilities::LuLoDbl(dbl) == 0);

// Get the first digit.

if ((uT = \*psz - '0') > 9)

{

if ((uT -= 'A' - '0') <= 5 || (uT -= 'a' - 'A') <= 5)

uT += 10;

else

{

\*ppchLim = psz;

return dbl;

}

}

psz++;

if (uT & 0x08)

{

cbit = 4;

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)(uT & 0x07) << 17;

}

else if (uT & 0x04)

{

cbit = 3;

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)(uT & 0x03) << 18;

}

else if (uT & 0x02)

{

cbit = 2;

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)(uT & 0x01) << 19;

}

else

{

Assert(uT & 0x01);

cbit = 1;

}

bExtra = 0;

for (; ; psz++)

{

if ((uT = (\*psz - '0')) > 9)

{

if ((uT -= 'A' - '0') <= 5 || (uT -= 'a' - 'A') <= 5)

uT += 10;

else

break;

}

if (cbit <= 17)

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)uT << (17 - cbit);

else if (cbit < 21)

{

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)uT >> (cbit - 17);

Js::NumberUtilities::LuLoDbl(dbl) |= (ulong)uT << (49 - cbit);

}

else if (cbit <= 49)

Js::NumberUtilities::LuLoDbl(dbl) |= (ulong)uT << (49 - cbit);

else if (cbit <= 53)

{

Js::NumberUtilities::LuLoDbl(dbl) |= (ulong)uT >> (cbit - 49);

bExtra = (byte)(uT << (57 - cbit));

}

else if (0 != uT)

bExtra |= 1;

cbit += 4;

}

// Set the lim.

\*ppchLim = psz;

// Set the exponent.

cbit += 1022;

if (cbit > 2046)

{

// Overflow to Infinity

Js::NumberUtilities::LuHiDbl(dbl) = 0x7FF00000;

Js::NumberUtilities::LuLoDbl(dbl) = 0;

return dbl;

}

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)cbit << 20;

// Use bExtra to round.

if ((bExtra & 0x80) && ((bExtra & 0x7F) || (Js::NumberUtilities::LuLoDbl(dbl) & 1)))

{

// Round up. Note that this overflows the mantissa correctly,

// even to Infinity.

if (0 == ++Js::NumberUtilities::LuLoDbl(dbl))

++Js::NumberUtilities::LuHiDbl(dbl);

}

return dbl;

}

template <typename EncodedChar>

double NumberUtilities::DblFromBinary(const EncodedChar \*psz, const EncodedChar \*\*ppchLim)

{

double dbl = 0;

Assert(Js::NumberUtilities::LuHiDbl(dbl) == 0);

Assert(Js::NumberUtilities::LuLoDbl(dbl) == 0);

uint uT;

byte bExtra = 0;

int cbit = 0;

// Skip leading zeros.

while (\*psz == '0')

psz++;

// Get the first digit.

uT = \*psz - '0';

if (uT > 1)

{

\*ppchLim = psz;

return dbl;

}

//Now that leading zeros are skipped first bit should be one so lets

//go ahead and count it and increment psz

cbit = 1;

psz++;

// According to the existing implementations these numbers

// should n bits away from 21 and 53. The n bits are determined by the

// numerical type. for example since 4 bits are necessary to represent a

// hexadecimal number and 3 bits to represent an octal you will see that

// the hex case is represented by 21-4 = 17 and the octal case is represented

// by 21-3 = 18, thus for binary where 1 bit is need to represent 2 numbers 21-1 = 20

const uint rightShiftValue = 20;

// Why 52? 52 is the last explicit bit and 1 bit away from 53 (max bits of precision

// for double precision floating point)

const uint leftShiftValue = 52;

for (; (uT = (\*psz - '0')) <= 1; psz++)

{

if (cbit <= rightShiftValue)

{

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)uT << (rightShiftValue - cbit);

}

else if (cbit <= leftShiftValue)

{

Js::NumberUtilities::LuLoDbl(dbl) |= (ulong)uT << (leftShiftValue - cbit);

}

else if (cbit == leftShiftValue + 1)//53 bits

{

Js::NumberUtilities::LuLoDbl(dbl) |= (ulong)uT >> (cbit - leftShiftValue);

bExtra = (byte)(uT << (60 - cbit));

}

else if (0 != uT)

{

bExtra |= 1;

}

cbit++;

}

// Set the lim.

\*ppchLim = psz;

// Set the exponent.

cbit += 1022;

if (cbit > 2046)

{

// Overflow to Infinity

Js::NumberUtilities::LuHiDbl(dbl) = 0x7FF00000;

Js::NumberUtilities::LuLoDbl(dbl) = 0;

return dbl;

}

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)cbit << 20;

// Use bExtra to round.

if ((bExtra & 0x80) && ((bExtra & 0x7F) || (Js::NumberUtilities::LuLoDbl(dbl) & 1)))

{

// Round up. Note that this overflows the mantissa correctly,

// even to Infinity.

if (0 == ++Js::NumberUtilities::LuLoDbl(dbl))

++Js::NumberUtilities::LuHiDbl(dbl);

}

return dbl;

}

template <typename EncodedChar>

double NumberUtilities::DblFromOctal(const EncodedChar \*psz, const EncodedChar \*\*ppchLim)

{

double dbl;

uint uT;

byte bExtra;

int cbit;

// Skip leading zeros.

while (\*psz == '0')

psz++;

dbl = 0;

Assert(Js::NumberUtilities::LuHiDbl(dbl) == 0);

Assert(Js::NumberUtilities::LuLoDbl(dbl) == 0);

// Get the first digit.

uT = \*psz - '0';

if (uT > 7)

{

\*ppchLim = psz;

return dbl;

}

psz++;

if (uT & 0x04)//is the 3rd bit set

{

cbit = 3;

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)(uT & 0x03) << 18;

}

else if (uT & 0x02)//is the 2nd bit set

{

cbit = 2;

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)(uT & 0x01) << 19;

}

else// then is the first bit set

{

Assert(uT & 0x01);

cbit = 1;

}

bExtra = 0;

for (; (uT = (\*psz - '0')) <= 7; psz++)

{

if (cbit <= 18)

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)uT << (18 - cbit);

else if (cbit < 21)

{

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)uT >> (cbit - 18);

Js::NumberUtilities::LuLoDbl(dbl) |= (ulong)uT << (50 - cbit);

}

else if (cbit <= 50)

Js::NumberUtilities::LuLoDbl(dbl) |= (ulong)uT << (50 - cbit);

else if (cbit <= 53)

{

Js::NumberUtilities::LuLoDbl(dbl) |= (ulong)uT >> (cbit - 50);

bExtra = (byte)(uT << (58 - cbit));

}

else if (0 != uT)

bExtra |= 1;

cbit += 3;

}

// Set the lim.

\*ppchLim = psz;

// Set the exponent.

cbit += 1022;

if (cbit > 2046)

{

// Overflow to Infinity

Js::NumberUtilities::LuHiDbl(dbl) = 0x7FF00000;

Js::NumberUtilities::LuLoDbl(dbl) = 0;

return dbl;

}

Js::NumberUtilities::LuHiDbl(dbl) |= (ulong)cbit << 20;

// Use bExtra to round.

if ((bExtra & 0x80) && ((bExtra & 0x7F) || (Js::NumberUtilities::LuLoDbl(dbl) & 1)))

{

// Round up. Note that this overflows the mantissa correctly,

// even to Infinity.

if (0 == ++Js::NumberUtilities::LuLoDbl(dbl))

++Js::NumberUtilities::LuHiDbl(dbl);

}

return dbl;

}

template <typename EncodedChar>

double NumberUtilities::StrToDbl(const EncodedChar \* psz, const EncodedChar \*\*ppchLim, Js::ScriptContext \*const scriptContext)

{

Assert(scriptContext);

bool likelyInt = true;

return Js::NumberUtilities::StrToDbl<EncodedChar>(psz, ppchLim, likelyInt);

}

template double NumberUtilities::StrToDbl<wchar\_t>(const wchar\_t \* psz, const wchar\_t \*\*ppchLim, Js::ScriptContext \*const scriptContext);

template double NumberUtilities::StrToDbl<utf8char\_t>(const utf8char\_t \* psz, const utf8char\_t \*\*ppchLim, Js::ScriptContext \*const scriptContext);

template double NumberUtilities::DblFromHex<wchar\_t>(const wchar\_t \*psz, const wchar\_t \*\*ppchLim);

template double NumberUtilities::DblFromHex<utf8char\_t>(const utf8char\_t \*psz, const utf8char\_t \*\*ppchLim);

template double NumberUtilities::DblFromBinary<wchar\_t>(const wchar\_t \*psz, const wchar\_t \*\*ppchLim);

template double NumberUtilities::DblFromBinary<utf8char\_t>(const utf8char\_t \*psz, const utf8char\_t \*\*ppchLim);

template double NumberUtilities::DblFromOctal<wchar\_t>(const wchar\_t \*psz, const wchar\_t \*\*ppchLim);

template double NumberUtilities::DblFromOctal<utf8char\_t>(const utf8char\_t \*psz, const utf8char\_t \*\*ppchLim);

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

class NumberConstants : public NumberConstantsBase

{

public:

// Our float tagging scheme relies on NaNs to be of this value - changing the NaN value

// will break float tagging for x64.

static const uint64 k\_PosInf = 0x7FF0000000000000ull;

static const uint64 k\_NegInf = 0xFFF0000000000000ull;

static const uint64 k\_PosMin = 0x0000000000000001ull;

static const uint64 k\_PosMax = 0x7FEFFFFFFFFFFFFFull;

static const uint64 k\_NegZero = 0x8000000000000000ull;

static const uint64 k\_Zero = 0x0000000000000000ull;

static const uint64 k\_PointFive = 0x3FE0000000000000ull;

static const uint64 k\_NegPointFive = 0xBFE0000000000000ull;

static const uint64 k\_NegOne = 0xBFF0000000000000ull;

static const uint32 k\_Float32Zero = 0x00000000ul;

static const uint32 k\_Float32PointFive = 0x3F000000ul;

static const uint32 k\_Float32NegPointFive = 0xBF000000ul;

static const double MAX\_VALUE;

static const double MIN\_VALUE;

static const double NaN;

static const double NEGATIVE\_INFINITY;

static const double POSITIVE\_INFINITY;

static const double NEG\_ZERO;

static const BYTE AbsDoubleCst[];

static const BYTE AbsFloatCst[];

static double const UIntConvertConst[];

static double const MaskNegDouble[];

static float const MaskNegFloat[];

};

class NumberUtilities : public NumberUtilitiesBase

{

public:

static bool IsDigit(int ch);

static BOOL NumberUtilities::FHexDigit(wchar\_t ch, int \*pw);

static ulong MulLu(ulong lu1, ulong lu2, ulong \*pluHi);

static int AddLu(ulong \*plu1, ulong lu2);

static ulong &LuHiDbl(double &dbl);

static ulong &LuLoDbl(double &dbl);

static INT64 TryToInt64(double d);

static bool IsValidTryToInt64(\_\_int64 value); // Whether TryToInt64 resulted in a valid value.

static int CbitZeroLeft(ulong lu);

static bool IsFinite(double value);

static bool IsNan(double value);

static bool IsSpecial(double value, uint64 nSpecial);

static uint64 ToSpecial(double value);

// Convert a given UINT16 into its corresponding string.

// outBufferSize is in WCHAR elements (and used only for ASSERTs)

// Returns the number of characters written to outBuffer (not including the \0)

static charcount\_t UInt16ToString(uint16 integer, \_\_out \_\_ecount(outBufferSize) WCHAR\* outBuffer, charcount\_t outBufferSize, char widthForPaddingZerosInsteadSpaces);

// Try to parse an integer string to find out if the string contains an index property name.

static BOOL TryConvertToUInt32(const wchar\_t\* str, int length, uint32\* intVal);

static double Modulus(double dblLeft, double dblRight);

enum FormatType

{

FormatFixed,

FormatExponential,

FormatPrecision

};

// Implemented in lib\parser\common. Should move to lib\common

template<typename EncodedChar>

static double StrToDbl(const EncodedChar \*psz, const EncodedChar \*\*ppchLim, bool& likelyInt);

static BOOL FDblToStr(double dbl, \_\_out\_ecount(nDstBufSize) wchar\_t \*psz, int nDstBufSize);

static int FDblToStr(double dbl, NumberUtilities::FormatType ft, int nDigits, \_\_out\_ecount(cchDst) wchar\_t \*pchDst, int cchDst);

static BOOL FNonZeroFiniteDblToStr(double dbl, \_Out\_writes\_(nDstBufSize) WCHAR\* psz, int nDstBufSize);

\_Success\_(return) static BOOL FNonZeroFiniteDblToStr(double dbl, \_In\_range\_(2, 36) int radix, \_Out\_writes\_(nDstBufSize) WCHAR\* psz, int nDstBufSize);

static double DblFromDecimal(DECIMAL \* pdecIn);

static void CodePointAsSurrogatePair(codepoint\_t codePointValue, \_\_out wchar\_t\* first, \_\_out wchar\_t\* second);

static codepoint\_t SurrogatePairAsCodePoint(codepoint\_t first, codepoint\_t second);

static bool IsSurrogateUpperPart(codepoint\_t codePointValue);

static bool IsSurrogateLowerPart(codepoint\_t codePointValue);

static bool IsInSupplementaryPlane(codepoint\_t codePointValue);

static long LwFromDblNearest(double dbl);

static ulong LuFromDblNearest(double dbl);

static BOOL FDblIsLong(double dbl, long \*plw);

template<typename EncodedChar>

static double DblFromHex(const EncodedChar \*psz, const EncodedChar \*\*ppchLim);

template <typename EncodedChar>

static double DblFromBinary(const EncodedChar \*psz, const EncodedChar \*\*ppchLim);

template<typename EncodedChar>

static double DblFromOctal(const EncodedChar \*psz, const EncodedChar \*\*ppchLim);

template<typename EncodedChar>

static double StrToDbl(const EncodedChar \*psz, const EncodedChar \*\*ppchLim, Js::ScriptContext \*const scriptContext);

const NumberUtilitiesBase\* GetNumberUtilitiesBase() const { return static\_cast<const NumberUtilitiesBase\*>(this); }

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#ifndef NUMBER\_UTIL\_INLINE

#define NUMBER\_UTIL\_INLINE

#endif

namespace Js

{

NUMBER\_UTIL\_INLINE ulong &NumberUtilities::LuHiDbl(double &dbl)

{

#ifdef BIG\_ENDIAN

return ((ulong \*)&dbl)[0];

#else //!BIG\_ENDIAN

return ((ulong \*)&dbl)[1];

#endif //!BIG\_ENDIAN

}

NUMBER\_UTIL\_INLINE ulong &NumberUtilities::LuLoDbl(double &dbl)

{

#ifdef BIG\_ENDIAN

return ((ulong \*)&dbl)[1];

#else //!BIG\_ENDIAN

return ((ulong \*)&dbl)[0];

#endif //!BIG\_ENDIAN

}

#if defined(\_M\_X64)

NUMBER\_UTIL\_INLINE INT64 NumberUtilities::TryToInt64(double T1)

{

// \_mm\_cvttsd\_si64x will result in 0x8000000000000000 if the value is NaN Inf or Zero, or overflows int64

\_\_m128d a;

a = \_mm\_load\_sd(&T1);

return \_mm\_cvttsd\_si64x(a);

}

#else

NUMBER\_UTIL\_INLINE INT64 NumberUtilities::TryToInt64(double T1)

{

INT64 T4\_64;

#if defined(\_M\_IX86)

// If SSE3 is available use FISTPP. VC (dev10) generates a FISTP, but needs to

// first change the FPU rounding, which is very slow...

if (AutoSystemInfo::Data.SSE3Available())

{

// FISTTP will result in 0x8000000000000000 in T4\_64 if the value is NaN Inf or Zero, or overflows int64

\_asm {

FLD T1

FISTTP T4\_64

}

}

else

#endif

#if defined(\_M\_ARM32\_OR\_ARM64)

// Win8 286065: ARM: casts to int64 from double for NaNs, infinity, overflow:

// - non-infinity NaNs -> 0

// - infinity NaNs: -1.#INF -> 0x8000000000000000, 1.#INF -> 0x7FFFFFFFFFFFFFFF.

// - overflow: negative -> 0x8000000000000000, positive-> 0x7FFFFFFFFFFFFFFF.

// We have to take care of non-infinite NaNs to make sure the result is not a valid int64 rather than 0.

if (IsNan(T1))

{

return Pos\_InvalidInt64;

}

else if (T1 < -9223372036854775808.0) // -9223372036854775808 is double value corresponsing to Neg\_InvalidInt64.

{

// TODO: Remove this temp workaround.

// This is to walk around CRT issue (Win8 404170): there is a band of values near/less than negative overflow

// for which cast to int64 results in positive number (bug), then going further down in negative direction it turns

// back to negative overflow value (as it should).

return Pos\_InvalidInt64;

}

else

#endif

{

// The cast will result in 0x8000000000000000 in T4\_64 if the value is NaN Inf or Zero, or overflows int64

T4\_64 = static\_cast<INT64>(T1);

}

#if defined(\_M\_ARM32\_OR\_ARM64)

if (T4\_64 == Neg\_InvalidInt64)

{

// Win8 391983: what happens in 64bit overflow is not spec'ed. On ARM T4\_64 would be 0x7F..FF but if we extend

// ToInt32 to 64bit, because of ES5\_9.5.5 the result would be 0x80..00. On Intel all overflows result in 0x80..00.

// So, be consistent with Intel.

return Pos\_InvalidInt64;

}

#endif

return T4\_64;

}

#endif

// Returns true <=> TryToInt64() call resulted in a valid value.

NUMBER\_UTIL\_INLINE bool NumberUtilities::IsValidTryToInt64(\_\_int64 value)

{

#if defined(\_M\_ARM32\_OR\_ARM64)

return value != Pos\_InvalidInt64 && value != Neg\_InvalidInt64;

#else

return value != Pos\_InvalidInt64;

#endif

}

NUMBER\_UTIL\_INLINE bool NumberUtilities::IsNan(double value)

{

#if defined(\_M\_X64\_OR\_ARM64)

// NaN is a range of values; all bits on the exponent are 1's and some nonzero significant.

// no distinction on signed NaN's

uint64 nCompare = ToSpecial(value);

bool isNan = (0 == (~nCompare & 0x7FF0000000000000ull) &&

0 != (nCompare & 0x000FFFFFFFFFFFFFull));

return isNan;

#else

return 0 == (~Js::NumberUtilities::LuHiDbl(value) & 0x7FF00000) &&

(0 != Js::NumberUtilities::LuLoDbl(value) || 0 != (Js::NumberUtilities::LuHiDbl(value) & 0x000FFFFF));

#endif

}

NUMBER\_UTIL\_INLINE bool NumberUtilities::IsSpecial(double value, uint64 nSpecial)

{

// Perform a bitwise comparison using uint64 instead of a double comparison, since that

// would trigger FPU exceptions, etc.

uint64 nCompare = ToSpecial(value);

return nCompare == nSpecial;

}

NUMBER\_UTIL\_INLINE uint64 NumberUtilities::ToSpecial(double value)

{

return \*(reinterpret\_cast<uint64 \*>(&value));

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

//////////////////////////////////////////////////////////

// NumberUtilitiesBase.h is used by static lib shared between trident and chakra. We need to keep the size

// consistent and try not to change its size. We need to have matching host if the size is changed here.

/////////////////////////////////////////////////////////

#pragma once

namespace Js

{

class NumberConstantsBase

{

public:

static const UINT64 k\_Nan = 0xFFF8000000000000ull;

};

class NumberUtilitiesBase

{

protected:

static const INT64 Pos\_InvalidInt64 = 0x8000000000000000ll; // Used for positive infinity/overflow.

static const INT64 Neg\_InvalidInt64 = 0x7fffffffffffffffll; // Used for negative infinity/overflow.

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include <CommonCommonPch.h>

#include "DataStructures\BigInt.h"

namespace Js

{

static inline BOOL FNzDigit(int ch)

{ return ch >= '1' && ch <= '9'; }

static const long klwMaxExp10 = 310; // Upper bound on base 10 exponent

static const long klwMinExp10 = -325; // Lower bound on base 10 exponent

static const int kcbMaxRgb = 50;

static const int kcchMaxSig = 20; // 20 significant digits. ECMA allows this.

// Small powers of ten. These are all the powers of ten that have an exact

// representation in IEEE double precision format.

static const double g\_rgdblTens[] =

{

1e00, 1e01, 1e02, 1e03, 1e04, 1e05, 1e06, 1e07, 1e08, 1e09,

1e10, 1e11, 1e12, 1e13, 1e14, 1e15, 1e16, 1e17, 1e18, 1e19,

1e20, 1e21, 1e22, 1e23, 1e24, 1e25, 1e26, 1e27, 1e28

};

static inline wchar\_t ToDigit(long wVal)

{

//return reinterpret\_cast<wchar\_t>((wVal < 10) ? '0' + (ushort) wVal : 'a' - 10 + (ushort) wVal);

return (ushort)((wVal < 10) ? '0' + (ushort) wVal : 'a' - 10 + (ushort) wVal);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Big floating point number.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct BIGNUM

{

// WARNING: some asm code below assumes the ordering of these fields.

ulong m\_lu0;

ulong m\_lu1;

ulong m\_lu2;

int m\_wExp;

// This is a bound on the absolute value of the error. It is based at

// one bit before the least significant bit of m\_lu0.

ulong m\_luError;

// Test to see if the num is zero. This works even if we're not normalized.

BOOL FZero(void)

{ return m\_lu2 == 0 && m\_lu1 == 0 && m\_lu0 == 0; }

// Normalize the big number - make sure the high bit is 1 or

// everything is zero (including the exponent).

void Normalize(void);

// Round based on the given extra data using IEEE round to nearest rule.

void Round(ulong luExtra)

{

if (0 == (luExtra & 0x80000000) ||

0 == (luExtra & 0x7FFFFFFF) && 0 == (m\_lu0 & 1))

{

if (luExtra)

m\_luError++;

return;

}

m\_luError++;

// Round up.

if (Js::NumberUtilities::AddLu(&m\_lu0, 1) &&

Js::NumberUtilities::AddLu(&m\_lu1, 1) &&

Js::NumberUtilities::AddLu(&m\_lu2, 1))

{

Assert(0 == m\_lu0);

Assert(0 == m\_lu1);

Assert(0 == m\_lu2);

m\_lu2 = 0x80000000;

m\_wExp++;

}

}

// Multiply by ten and add a base 10 digit.

void MulTenAdd(byte bAdd, ulong \*pluExtra);

// Set the value from decimal digits and decimal exponent

template <typename EncodedChar>

void SetFromRgchExp(const EncodedChar \*pch, long cch, long lwExp);

// Multiply by a BIGNUM

void Mul(const BIGNUM \*pnumOp);

// Get the double value.

double GetDbl(void);

// Lop off the integer part and return it.

ulong LuMod1(void)

{

if (m\_wExp <= 0)

return 0;

Assert(m\_wExp <= 32);

ulong luT = m\_lu2 >> (32 - m\_wExp);

m\_lu2 &= 0x7FFFFFFF >> (m\_wExp - 1);

Normalize();

return luT;

}

// If m\_luError is not zero, add it and set m\_luError to zero.

void MakeUpperBound(void)

{

Assert(m\_luError < 0xFFFFFFFF);

ulong luT = (m\_luError + 1) >> 1;

if (luT &&

Js::NumberUtilities::AddLu(&m\_lu0, luT) &&

Js::NumberUtilities::AddLu(&m\_lu1, 1) &&

Js::NumberUtilities::AddLu(&m\_lu2, 1))

{

Assert(m\_lu2 == 0);

Assert(m\_lu1 == 0);

m\_lu2 = 0x80000000;

m\_lu0 = (m\_lu0 >> 1) + (m\_lu0 & 1);

m\_wExp++;

}

m\_luError = 0;

}

// If m\_luError is not zero, subtract it and set m\_luError to zero.

void MakeLowerBound(void)

{

Assert(m\_luError < 0xFFFFFFFF);

ulong luT = (m\_luError + 1) >> 1;

if (luT &&

!Js::NumberUtilities::AddLu(&m\_lu0, (ulong)-(long)luT) &&

!Js::NumberUtilities::AddLu(&m\_lu1, 0xFFFFFFFF))

{

Js::NumberUtilities::AddLu(&m\_lu2, 0xFFFFFFFF);

if (0 == (0x80000000 & m\_lu2))

Normalize();

}

m\_luError = 0;

}

};

// Positive powers of 10 to 96 bits precision.

static const BIGNUM g\_rgnumPos[46] =

{

{ 0x00000000, 0x00000000, 0xA0000000, 4, 0 }, // 10^1

{ 0x00000000, 0x00000000, 0xC8000000, 7, 0 }, // 10^2

{ 0x00000000, 0x00000000, 0xFA000000, 10, 0 }, // 10^3

{ 0x00000000, 0x00000000, 0x9C400000, 14, 0 }, // 10^4

{ 0x00000000, 0x00000000, 0xC3500000, 17, 0 }, // 10^5

{ 0x00000000, 0x00000000, 0xF4240000, 20, 0 }, // 10^6

{ 0x00000000, 0x00000000, 0x98968000, 24, 0 }, // 10^7

{ 0x00000000, 0x00000000, 0xBEBC2000, 27, 0 }, // 10^8

{ 0x00000000, 0x00000000, 0xEE6B2800, 30, 0 }, // 10^9

{ 0x00000000, 0x00000000, 0x9502F900, 34, 0 }, // 10^10

{ 0x00000000, 0x00000000, 0xBA43B740, 37, 0 }, // 10^11

{ 0x00000000, 0x00000000, 0xE8D4A510, 40, 0 }, // 10^12

{ 0x00000000, 0x00000000, 0x9184E72A, 44, 0 }, // 10^13

{ 0x00000000, 0x80000000, 0xB5E620F4, 47, 0 }, // 10^14

{ 0x00000000, 0xA0000000, 0xE35FA931, 50, 0 }, // 10^15

{ 0x00000000, 0x04000000, 0x8E1BC9BF, 54, 0 }, // 10^16

{ 0x00000000, 0xC5000000, 0xB1A2BC2E, 57, 0 }, // 10^17

{ 0x00000000, 0x76400000, 0xDE0B6B3A, 60, 0 }, // 10^18

{ 0x00000000, 0x89E80000, 0x8AC72304, 64, 0 }, // 10^19

{ 0x00000000, 0xAC620000, 0xAD78EBC5, 67, 0 }, // 10^20

{ 0x00000000, 0x177A8000, 0xD8D726B7, 70, 0 }, // 10^21

{ 0x00000000, 0x6EAC9000, 0x87867832, 74, 0 }, // 10^22

{ 0x00000000, 0x0A57B400, 0xA968163F, 77, 0 }, // 10^23

{ 0x00000000, 0xCCEDA100, 0xD3C21BCE, 80, 0 }, // 10^24

{ 0x00000000, 0x401484A0, 0x84595161, 84, 0 }, // 10^25

{ 0x00000000, 0x9019A5C8, 0xA56FA5B9, 87, 0 }, // 10^26

{ 0x00000000, 0xF4200F3A, 0xCECB8F27, 90, 0 }, // 10^27

{ 0x40000000, 0xF8940984, 0x813F3978, 94, 0 }, // 10^28

{ 0x50000000, 0x36B90BE5, 0xA18F07D7, 97, 0 }, // 10^29

{ 0xA4000000, 0x04674EDE, 0xC9F2C9CD, 100, 0 }, // 10^30

{ 0x4D000000, 0x45812296, 0xFC6F7C40, 103, 0 }, // 10^31

{ 0xF0200000, 0x2B70B59D, 0x9DC5ADA8, 107, 0 }, // 10^32

{ 0x3CBF6B72, 0xFFCFA6D5, 0xC2781F49, 213, 1 }, // 10^64 (rounded up)

{ 0xC5CFE94F, 0xC59B14A2, 0xEFB3AB16, 319, 1 }, // 10^96 (rounded up)

{ 0xC66F336C, 0x80E98CDF, 0x93BA47C9, 426, 1 }, // 10^128

{ 0x577B986B, 0x7FE617AA, 0xB616A12B, 532, 1 }, // 10^160

{ 0x85BBE254, 0x3927556A, 0xE070F78D, 638, 1 }, // 10^192 (rounded up)

{ 0x82BD6B71, 0xE33CC92F, 0x8A5296FF, 745, 1 }, // 10^224 (rounded up)

{ 0xDDBB901C, 0x9DF9DE8D, 0xAA7EEBFB, 851, 1 }, // 10^256 (rounded up)

{ 0x73832EEC, 0x5C6A2F8C, 0xD226FC19, 957, 1 }, // 10^288

{ 0xE6A11583, 0xF2CCE375, 0x81842F29, 1064, 1 }, // 10^320

{ 0x5EBF18B7, 0xDB900AD2, 0x9FA42700, 1170, 1 }, // 10^352 (rounded up)

{ 0x1027FFF5, 0xAEF8AA17, 0xC4C5E310, 1276, 1 }, // 10^384

{ 0xB5E54F71, 0xE9B09C58, 0xF28A9C07, 1382, 1 }, // 10^416

{ 0xA7EA9C88, 0xEBF7F3D3, 0x957A4AE1, 1489, 1 }, // 10^448

{ 0x7DF40A74, 0x0795A262, 0xB83ED8DC, 1595, 1 }, // 10^480

};

// Negative powers of 10 to 96 bits precision.

static const BIGNUM g\_rgnumNeg[46] =

{

{ 0xCCCCCCCD, 0xCCCCCCCC, 0xCCCCCCCC, -3, 1 }, // 10^-1 (rounded up)

{ 0x3D70A3D7, 0x70A3D70A, 0xA3D70A3D, -6, 1 }, // 10^-2

{ 0x645A1CAC, 0x8D4FDF3B, 0x83126E97, -9, 1 }, // 10^-3

{ 0xD3C36113, 0xE219652B, 0xD1B71758, -13, 1 }, // 10^-4

{ 0x0FCF80DC, 0x1B478423, 0xA7C5AC47, -16, 1 }, // 10^-5

{ 0xA63F9A4A, 0xAF6C69B5, 0x8637BD05, -19, 1 }, // 10^-6 (rounded up)

{ 0x3D329076, 0xE57A42BC, 0xD6BF94D5, -23, 1 }, // 10^-7

{ 0xFDC20D2B, 0x8461CEFC, 0xABCC7711, -26, 1 }, // 10^-8

{ 0x31680A89, 0x36B4A597, 0x89705F41, -29, 1 }, // 10^-9 (rounded up)

{ 0xB573440E, 0xBDEDD5BE, 0xDBE6FECE, -33, 1 }, // 10^-10

{ 0xF78F69A5, 0xCB24AAFE, 0xAFEBFF0B, -36, 1 }, // 10^-11

{ 0xF93F87B7, 0x6F5088CB, 0x8CBCCC09, -39, 1 }, // 10^-12

{ 0x2865A5F2, 0x4BB40E13, 0xE12E1342, -43, 1 }, // 10^-13

{ 0x538484C2, 0x095CD80F, 0xB424DC35, -46, 1 }, // 10^-14 (rounded up)

{ 0x0F9D3701, 0x3AB0ACD9, 0x901D7CF7, -49, 1 }, // 10^-15

{ 0x4C2EBE68, 0xC44DE15B, 0xE69594BE, -53, 1 }, // 10^-16

{ 0x09BEFEBA, 0x36A4B449, 0xB877AA32, -56, 1 }, // 10^-17 (rounded up)

{ 0x3AFF322E, 0x921D5D07, 0x9392EE8E, -59, 1 }, // 10^-18

{ 0x2B31E9E4, 0xB69561A5, 0xEC1E4A7D, -63, 1 }, // 10^-19 (rounded up)

{ 0x88F4BB1D, 0x92111AEA, 0xBCE50864, -66, 1 }, // 10^-20 (rounded up)

{ 0xD3F6FC17, 0x74DA7BEE, 0x971DA050, -69, 1 }, // 10^-21 (rounded up)

{ 0x5324C68B, 0xBAF72CB1, 0xF1C90080, -73, 1 }, // 10^-22

{ 0x75B7053C, 0x95928A27, 0xC16D9A00, -76, 1 }, // 10^-23

{ 0xC4926A96, 0x44753B52, 0x9ABE14CD, -79, 1 }, // 10^-24

{ 0x3A83DDBE, 0xD3EEC551, 0xF79687AE, -83, 1 }, // 10^-25 (rounded up)

{ 0x95364AFE, 0x76589DDA, 0xC6120625, -86, 1 }, // 10^-26

{ 0x775EA265, 0x91E07E48, 0x9E74D1B7, -89, 1 }, // 10^-27 (rounded up)

{ 0x8BCA9D6E, 0x8300CA0D, 0xFD87B5F2, -93, 1 }, // 10^-28

{ 0x096EE458, 0x359A3B3E, 0xCAD2F7F5, -96, 1 }, // 10^-29

{ 0xA125837A, 0x5E14FC31, 0xA2425FF7, -99, 1 }, // 10^-30 (rounded up)

{ 0x80EACF95, 0x4B43FCF4, 0x81CEB32C, -102, 1 }, // 10^-31 (rounded up)

{ 0x67DE18EE, 0x453994BA, 0xCFB11EAD, -106, 1 }, // 10^-32 (rounded up)

{ 0x3F2398D7, 0xA539E9A5, 0xA87FEA27, -212, 1 }, // 10^-64

{ 0x11DBCB02, 0xFD75539B, 0x88B402F7, -318, 1 }, // 10^-96

{ 0xAC7CB3F7, 0x64BCE4A0, 0xDDD0467C, -425, 1 }, // 10^-128 (rounded up)

{ 0x59ED2167, 0xDB73A093, 0xB3F4E093, -531, 1 }, // 10^-160

{ 0x7B6306A3, 0x5423CC06, 0x91FF8377, -637, 1 }, // 10^-192

{ 0xA4F8BF56, 0x4A314EBD, 0xECE53CEC, -744, 1 }, // 10^-224

{ 0xFA911156, 0x637A1939, 0xC0314325, -850, 1 }, // 10^-256 (rounded up)

{ 0x4EE367F9, 0x836AC577, 0x9BECCE62, -956, 1 }, // 10^-288

{ 0x8920B099, 0x478238D0, 0xFD00B897, -1063, 1 }, // 10^-320 (rounded up)

{ 0x0092757C, 0x46F34F7D, 0xCD42A113, -1169, 1 }, // 10^-352 (rounded up)

{ 0x88DBA000, 0xB11B0857, 0xA686E3E8, -1275, 1 }, // 10^-384 (rounded up)

{ 0x1A4EB007, 0x3FFC68A6, 0x871A4981, -1381, 1 }, // 10^-416 (rounded up)

{ 0x84C663CF, 0xB6074244, 0xDB377599, -1488, 1 }, // 10^-448 (rounded up)

{ 0x61EB52E2, 0x79007736, 0xB1D983B4, -1594, 1 }, // 10^-480

};

void BIGNUM::Normalize(void)

{

int w1, w2;

// Normalize mantissa

if (m\_lu2 == 0)

{

if (m\_lu1 == 0)

{

if (m\_lu0 == 0)

{

m\_wExp = 0;

return;

}

m\_lu2 = m\_lu0;

m\_lu0 = 0;

m\_wExp -= 64;

}

else

{

m\_lu2 = m\_lu1;

m\_lu1 = m\_lu0;

m\_lu0 = 0;

m\_wExp -= 32;

}

}

if (0 != (w1 = Js::NumberUtilities::CbitZeroLeft(m\_lu2)))

{

w2 = 32 - w1;

m\_lu2 = (m\_lu2 << w1) | (m\_lu1 >> w2);

m\_lu1 = (m\_lu1 << w1) | (m\_lu0 >> w2);

m\_lu0 = (m\_lu0 << w1);

m\_wExp -= w1;

}

}

void BIGNUM::MulTenAdd(byte bAdd, ulong \*pluExtra)

{

Assert(bAdd <= 9);

Assert(m\_lu2 & 0x80000000);

ulong rglu[5];

// First "multiply" by eight

m\_wExp += 3;

Assert(m\_wExp >= 4);

// Initialize the carry values based on bAdd and m\_wExp.

memset(rglu, 0, sizeof(rglu));

if (0 != bAdd)

{

int ilu = 3 - (m\_wExp >> 5);

if (ilu < 0)

rglu[0] = 1;

else

{

int ibit = m\_wExp & 0x1F;

if (ibit < 4)

{

Assert(ilu < 4);

rglu[ilu + 1] = bAdd >> ibit;

if (ibit > 0)

rglu[ilu] = (ulong)bAdd << (32 - ibit);

}

else

{

Assert(ilu < 5);

rglu[ilu] = (ulong)bAdd << (32 - ibit);

}

}

}

// Shift and add to multiply by ten.

rglu[1] += Js::NumberUtilities::AddLu(&rglu[0], m\_lu0 << 30);

rglu[2] += Js::NumberUtilities::AddLu(&m\_lu0, (m\_lu0 >> 2) + (m\_lu1 << 30));

if (rglu[1])

rglu[2] += Js::NumberUtilities::AddLu(&m\_lu0, rglu[1]);

rglu[3] += Js::NumberUtilities::AddLu(&m\_lu1, (m\_lu1 >> 2) + (m\_lu2 << 30));

if (rglu[2])

rglu[3] += Js::NumberUtilities::AddLu(&m\_lu1, rglu[2]);

rglu[4] = Js::NumberUtilities::AddLu(&m\_lu2, (m\_lu2 >> 2) + rglu[3]);

// Handle the final carry.

if (rglu[4])

{

Assert(rglu[4] == 1);

rglu[0] = (rglu[0] >> 1) | (rglu[0] & 1) | (m\_lu0 << 31);

m\_lu0 = (m\_lu0 >> 1) | (m\_lu1 << 31);

m\_lu1 = (m\_lu1 >> 1) | (m\_lu2 << 31);

m\_lu2 = (m\_lu2 >> 1) | 0x80000000;

m\_wExp++;

}

\*pluExtra = rglu[0];

}

template<typename EncodedChar>

void BIGNUM::SetFromRgchExp(const EncodedChar \*prgch, long cch, long lwExp)

{

Assert(cch > 0);

AssertArrMemR(prgch, cch);

const BIGNUM \*prgnum;

int wT;

ulong luExtra;

const EncodedChar \*pchLim = prgch + cch;

// Record the first digit

Assert(FNzDigit(prgch[0]));

m\_lu2 = (ulong)(prgch[0] - '0') << 28;

m\_lu1 = 0;

m\_lu0 = 0;

m\_wExp = 4;

m\_luError = 0;

lwExp--;

Normalize();

while (++prgch < pchLim)

{

if (\*prgch == '.')

continue;

Assert(Js::NumberUtilities::IsDigit(\*prgch));

MulTenAdd((byte) (\*prgch - '0'), &luExtra);

lwExp--;

if (0 != luExtra)

{

// We've filled up our precision.

Round(luExtra);

if (prgch < pchLim + 1)

{

// There are more digits, so add another error bit just for

// safety's sake.

m\_luError++;

}

break;

}

}

// Now multiply by 10^lwExp

if (0 == lwExp)

return;

if (lwExp < 0)

{

prgnum = g\_rgnumNeg;

lwExp = -lwExp;

}

else

prgnum = g\_rgnumPos;

Assert(lwExp > 0 && lwExp < 512);

wT = (int)lwExp & 0x1F;

if (wT > 0)

Mul(&prgnum[wT - 1]);

wT = ((int)lwExp >> 5) & 0x0F;

if (wT > 0)

Mul(&prgnum[wT + 30]);

}

void BIGNUM::Mul(const BIGNUM \*pnumOp)

{

ulong rglu[6];

Assert(m\_lu2 & 0x80000000);

Assert(pnumOp->m\_lu2 & 0x80000000);

memset(rglu, 0, sizeof(rglu));

#if I386\_ASM

\_\_asm

{

mov edi,this

mov esi,pnumOp

lea ebx,rglu

// first "digit" of pnumOp

mov ecx,DWORD PTR [esi]

cmp ecx,0

jz LDigit2

mov eax,DWORD PTR [edi]

mul ecx

mov DWORD PTR [ebx],eax

mov DWORD PTR [ebx+4],edx

mov eax,DWORD PTR [edi+4]

mul ecx

add DWORD PTR [ebx+4],eax

adc DWORD PTR [ebx+8],edx

mov eax,DWORD PTR [edi+8]

mul ecx

add DWORD PTR [ebx+8],eax

adc DWORD PTR [ebx+12],edx

// second "digit" of pnumOp

LDigit2:

mov ecx,DWORD PTR [esi+4]

cmp ecx,0

jz LDigit3

mov eax,DWORD PTR [edi]

mul ecx

add DWORD PTR [ebx+4],eax

adc DWORD PTR [ebx+8],edx

adc DWORD PTR [ebx+12],0

adc DWORD PTR [ebx+16],0

mov eax,DWORD PTR [edi+4]

mul ecx

add DWORD PTR [ebx+8],eax

adc DWORD PTR [ebx+12],edx

adc DWORD PTR [ebx+16],0

mov eax,DWORD PTR [edi+8]

mul ecx

add DWORD PTR [ebx+12],eax

adc DWORD PTR [ebx+16],edx

// third "digit" of pnumOp

LDigit3:

mov ecx,DWORD PTR [esi+8]

mov eax,DWORD PTR [edi]

mul ecx

add DWORD PTR [ebx+8],eax

adc DWORD PTR [ebx+12],edx

adc DWORD PTR [ebx+16],0

adc DWORD PTR [ebx+20],0

mov eax,DWORD PTR [edi+4]

mul ecx

add DWORD PTR [ebx+12],eax

adc DWORD PTR [ebx+16],edx

adc DWORD PTR [ebx+20],0

mov eax,DWORD PTR [edi+8]

mul ecx

add DWORD PTR [ebx+16],eax

adc DWORD PTR [ebx+20],edx

}

#else //!I386\_ASM

ulong luLo, luHi, luT;

int wCarry;

if (0 != (luT = m\_lu0))

{

luLo = Js::NumberUtilities::MulLu(luT, pnumOp->m\_lu0, &luHi);

rglu[0] = luLo;

rglu[1] = luHi;

luLo = Js::NumberUtilities::MulLu(luT, pnumOp->m\_lu1, &luHi);

Assert(luHi < 0xFFFFFFFF);

wCarry = Js::NumberUtilities::AddLu(&rglu[1], luLo);

Js::NumberUtilities::AddLu(&rglu[2], luHi + wCarry);

luLo = Js::NumberUtilities::MulLu(luT, pnumOp->m\_lu2, &luHi);

Assert(luHi < 0xFFFFFFFF);

wCarry = Js::NumberUtilities::AddLu(&rglu[2], luLo);

Js::NumberUtilities::AddLu(&rglu[3], luHi + wCarry);

}

if (0 != (luT = m\_lu1))

{

luLo = Js::NumberUtilities::MulLu(luT, pnumOp->m\_lu0, &luHi);

Assert(luHi < 0xFFFFFFFF);

wCarry = Js::NumberUtilities::AddLu(&rglu[1], luLo);

wCarry = Js::NumberUtilities::AddLu(&rglu[2], luHi + wCarry);

if (wCarry && Js::NumberUtilities::AddLu(&rglu[3], 1))

Js::NumberUtilities::AddLu(&rglu[4], 1);

luLo = Js::NumberUtilities::MulLu(luT, pnumOp->m\_lu1, &luHi);

Assert(luHi < 0xFFFFFFFF);

wCarry = Js::NumberUtilities::AddLu(&rglu[2], luLo);

wCarry = Js::NumberUtilities::AddLu(&rglu[3], luHi + wCarry);

if (wCarry)

Js::NumberUtilities::AddLu(&rglu[4], 1);

luLo = Js::NumberUtilities::MulLu(luT, pnumOp->m\_lu2, &luHi);

Assert(luHi < 0xFFFFFFFF);

wCarry = Js::NumberUtilities::AddLu(&rglu[3], luLo);

Js::NumberUtilities::AddLu(&rglu[4], luHi + wCarry);

}

luT = m\_lu2;

Assert(0 != luT);

luLo = Js::NumberUtilities::MulLu(luT, pnumOp->m\_lu0, &luHi);

Assert(luHi < 0xFFFFFFFF);

wCarry = Js::NumberUtilities::AddLu(&rglu[2], luLo);

wCarry = Js::NumberUtilities::AddLu(&rglu[3], luHi + wCarry);

if (wCarry && Js::NumberUtilities::AddLu(&rglu[4], 1))

Js::NumberUtilities::AddLu(&rglu[5], 1);

luLo = Js::NumberUtilities::MulLu(luT, pnumOp->m\_lu1, &luHi);

Assert(luHi < 0xFFFFFFFF);

wCarry = Js::NumberUtilities::AddLu(&rglu[3], luLo);

wCarry = Js::NumberUtilities::AddLu(&rglu[4], luHi + wCarry);

if (wCarry)

Js::NumberUtilities::AddLu(&rglu[5], 1);

luLo = Js::NumberUtilities::MulLu(luT, pnumOp->m\_lu2, &luHi);

Assert(luHi < 0xFFFFFFFF);

wCarry = Js::NumberUtilities::AddLu(&rglu[4], luLo);

Js::NumberUtilities::AddLu(&rglu[5], luHi + wCarry);

#endif //!I386\_ASM

// Compute the new exponent

m\_wExp += pnumOp->m\_wExp;

// Accumulate the error. Adding doesn't necessarily give an accurate

// bound if both of the errors are bigger than 2.

Assert(m\_luError <= 2 || pnumOp->m\_luError <= 2);

m\_luError += pnumOp->m\_luError;

// Handle rounding and normalize.

if (0 == (rglu[5] & 0x80000000))

{

if (0 != (rglu[2] & 0x40000000) &&

(0 != (rglu[2] & 0xBFFFFFFF) || 0 != rglu[1] || 0 != rglu[0]))

{

// Round up by 1

if (Js::NumberUtilities::AddLu(&rglu[2], 0x40000000) &&

Js::NumberUtilities::AddLu(&rglu[3], 1) &&

Js::NumberUtilities::AddLu(&rglu[4], 1))

{

Js::NumberUtilities::AddLu(&rglu[5], 1);

if (rglu[5] & 0x80000000)

goto LNormalized;

}

}

// have to shift by one

Assert(0 != (rglu[5] & 0x40000000));

m\_lu2 = (rglu[5] << 1) | (rglu[4] >> 31);

m\_lu1 = (rglu[4] << 1) | (rglu[3] >> 31);

m\_lu0 = (rglu[3] << 1) | (rglu[2] >> 31);

m\_wExp--;

m\_luError <<= 1;

// Add one for the error.

if ((rglu[2] & 0x7FFFFFFF) || rglu[1] || rglu[0])

m\_luError++;

}

else

{

if (0 != (rglu[2] & 0x80000000) &&

(0 != (rglu[3] & 1) || 0 != (rglu[2] & 0x7FFFFFFF) ||

0 != rglu[1] || 0 != rglu[0]))

{

// Round up by 1

if (Js::NumberUtilities::AddLu(&rglu[3], 1) &&

Js::NumberUtilities::AddLu(&rglu[4], 1) &&

Js::NumberUtilities::AddLu(&rglu[5], 1))

{

Assert(0 == rglu[3]);

Assert(0 == rglu[4]);

Assert(0 == rglu[5]);

rglu[5] = 0x80000000;

m\_wExp++;

}

}

LNormalized:

m\_lu2 = rglu[5];

m\_lu1 = rglu[4];

m\_lu0 = rglu[3];

// Add one for the error.

if (rglu[2] || rglu[1] || rglu[0])

m\_luError++;

}

}

double BIGNUM::GetDbl(void)

{

double dbl;

ulong luEx;

int wExp;

Assert(m\_lu2 & 0x80000000);

wExp = m\_wExp + 1022;

if (wExp >= 2047)

{

Js::NumberUtilities::LuHiDbl(dbl) = 0x7FF00000;

Js::NumberUtilities::LuLoDbl(dbl) = 0;

return dbl;

}

// Round after filling in the bits. In the extra ulong, we set the low bit

// if there are any extra non-zero bits. This is for breaking the tie when

// deciding whether to round up or down.

if (wExp > 0)

{

// Normalized.

Js::NumberUtilities::LuHiDbl(dbl) = ((ulong)wExp << 20) | ((m\_lu2 & 0x7FFFFFFF) >> 11);

Js::NumberUtilities::LuLoDbl(dbl) = m\_lu2 << 21 | m\_lu1 >> 11;

luEx = m\_lu1 << 21 | (m\_lu0 != 0);

}

else if (wExp > -20)

{

// Denormal with some high bits.

int wT = 12 - wExp;

Assert(wT >= 12 && wT < 32);

Js::NumberUtilities::LuHiDbl(dbl) = m\_lu2 >> wT;

Js::NumberUtilities::LuLoDbl(dbl) = (m\_lu2 << (32 - wT)) | (m\_lu1 >> wT);

luEx = (m\_lu1 << (32 - wT)) | (m\_lu0 != 0);

}

else if (wExp == -20)

{

// Denormal with no high bits.

Js::NumberUtilities::LuHiDbl(dbl) = 0;

Js::NumberUtilities::LuLoDbl(dbl) = m\_lu2;

luEx = m\_lu1 | (m\_lu0 != 0);

}

else if (wExp > -52)

{

// Denormal with no high bits.

int wT = -wExp - 20;

Assert(wT > 0 && wT < 32);

Js::NumberUtilities::LuHiDbl(dbl) = 0;

Js::NumberUtilities::LuLoDbl(dbl) = m\_lu2 >> wT;

luEx = m\_lu2 << (32 - wT) | (m\_lu1 != 0) |

(m\_lu0 != 0);

}

else if (wExp == -52)

{

// Zero unless we round up below.

Js::NumberUtilities::LuHiDbl(dbl) = 0;

Js::NumberUtilities::LuLoDbl(dbl) = 0;

luEx = m\_lu2 | (m\_lu1 != 0) | (m\_lu0 != 0);

}

else

return (double)0;

// Handle rounding

if ((luEx & 0x80000000) && ((luEx & 0x7FFFFFFF) || (Js::NumberUtilities::LuLoDbl(dbl) & 1)))

{

// Round up. Note that this works even when we overflow into the

// exponent.

if (Js::NumberUtilities::AddLu(&Js::NumberUtilities::LuLoDbl(dbl), 1))

Js::NumberUtilities::AddLu(&Js::NumberUtilities::LuHiDbl(dbl), 1);

}

return dbl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The double contains a binary value, M \* 2^n, which is off by at most 1

in the least significant bit; (prgch, cch, lwExp) represents a decimal

value, D \* 10^e. Note that (prgch, cch) may contain a decimal point and

lwExp is as if there is an implied decimal point immediately preceding

the digits.

The general scheme is to find an integer N (the smaller the better) such

that N \* M \* 2^n and N \* D \* 10^e are both integers. We then compare

N \* M \* 2^n to N \* D \* 10^e (at full precision). If the binary value is

greater, we adjust it to be exactly half way to the next value that can

come from a double. We then compare again to decided whether to bump the

double up to the next value. Similarly if the binary value is smaller,

we adjust it to be exactly half way to the previous representable value

and re-compare.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename EncodedChar>

static double AdjustDbl(double dbl, const EncodedChar \*prgch, long cch, long lwExp)

{

Js::BigInt biDec, biDbl;

long c2Dec, c2Dbl;

long c5Dec, c5Dbl;

int wAddHi, wT;

long iT;

long wExp2;

ulong rglu[2];

ulong luT;

BOOL f;

long clu;

if (!biDec.FInitFromDigits(prgch, cch, &cch))

goto LFail;

lwExp -= cch;

// lwExp is a base 10 exponent.

if (lwExp >= 0)

{

c5Dec = c2Dec = lwExp;

c5Dbl = c2Dbl = 0;

}

else

{

c5Dec = c2Dec = 0;

c5Dbl = c2Dbl = -lwExp;

}

rglu[1] = Js::NumberUtilities::LuHiDbl(dbl);

wExp2 = (rglu[1] >> 20) & 0x07FF;

rglu[1] &= 0x000FFFFF;

rglu[0] = Js::NumberUtilities::LuLoDbl(dbl);

wAddHi = 1;

if (0 != wExp2)

{

// Normal, so add implicit bit.

if (0 == rglu[1] && 0 == rglu[0] && 1 != wExp2)

{

// Power of 2 (and not adjacent to the first denormal), so the

// adjacent low value is closer than the high value.

wAddHi = 2;

rglu[1] = 0x00200000;

wExp2--;

}

else

rglu[1] |= 0x00100000;

wExp2 -= 1076;

}

else

wExp2 = -1075;

// Shift left by 1 bit : the adjustment values need the next lower bit.

rglu[1] = (rglu[1] << 1) | (rglu[0] >> 31);

rglu[0] <<= 1;

// We must determine how many words of significant digits this requires.

if (0 == rglu[0] && 0 == rglu[1])

clu = 0;

else if (0 == rglu[1])

clu = 1;

else

clu = 2;

f = biDbl.FInitFromRglu(rglu, clu);

Assert(f);

if (wExp2 >= 0)

c2Dbl += wExp2;

else

c2Dec += -wExp2;

// Eliminate common powers of 2.

if (c2Dbl > c2Dec)

{

c2Dbl -= c2Dec;

c2Dec = 0;

// See if biDec has some powers of 2 that we can get rid of.

for (iT = 0; c2Dbl >= 32 && 0 == biDec.Lu(iT); iT++)

c2Dbl -= 32;

if (iT > 0)

biDec.ShiftLusRight(iT);

Assert(c2Dbl < 32 || biDec.Lu(0) != 0);

luT = biDec.Lu(0);

for (iT = 0; iT < c2Dbl && 0 == (luT & (1L << iT)); iT++)

;

if (iT > 0)

{

c2Dbl -= iT;

biDec.ShiftRight(iT);

}

}

else

{

c2Dec -= c2Dbl;

c2Dbl = 0;

}

// There are no common powers of 2 or common powers of 5.

Assert(0 == c2Dbl || 0 == c2Dec);

Assert(0 == c5Dbl || 0 == c5Dec);

// Fold in the powers of 5.

if (c5Dbl > 0)

{

if (!biDbl.FMulPow5(c5Dbl))

goto LFail;

}

else if (c5Dec > 0 && !biDec.FMulPow5(c5Dec))

goto LFail;

// Fold in the powers of 2.

if (c2Dbl > 0)

{

if (!biDbl.FShiftLeft(c2Dbl))

goto LFail;

}

else if (c2Dec > 0 && !biDec.FShiftLeft(c2Dec))

goto LFail;

// Now determine whether biDbl is above or below biDec.

wT = biDbl.Compare(&biDec);

if (0 == wT)

return dbl;

if (wT > 0)

{

// biDbl is greater. Recompute with the dbl minus half the distance

// to the next smaller double.

if (!Js::NumberUtilities::AddLu(&rglu[0], 0xFFFFFFFF))

Js::NumberUtilities::AddLu(&rglu[1], 0xFFFFFFFF);

AssertVerify(biDbl.FInitFromRglu(rglu, 1 + (0 != rglu[1])));

if (c5Dbl > 0 && !biDbl.FMulPow5(c5Dbl))

goto LFail;

if (c2Dbl > 0 && !biDbl.FShiftLeft(c2Dbl))

goto LFail;

wT = biDbl.Compare(&biDec);

if (wT > 0 || 0 == wT && 0 != (Js::NumberUtilities::LuLoDbl(dbl) & 1))

{

// Return the next lower value.

if (!Js::NumberUtilities::AddLu(&Js::NumberUtilities::LuLoDbl(dbl), 0xFFFFFFFF))

Js::NumberUtilities::AddLu(&Js::NumberUtilities::LuHiDbl(dbl), 0xFFFFFFFF);

}

}

else

{

// biDbl is smaller. Recompute with the dbl plus half the distance

// to the next larger double.

if (Js::NumberUtilities::AddLu(&rglu[0], wAddHi))

Js::NumberUtilities::AddLu(&rglu[1], 1);

AssertVerify(biDbl.FInitFromRglu(rglu, 1 + (0 != rglu[1])));

if (c5Dbl > 0 && !biDbl.FMulPow5(c5Dbl))

goto LFail;

if (c2Dbl > 0 && !biDbl.FShiftLeft(c2Dbl))

goto LFail;

wT = biDbl.Compare(&biDec);

if (wT < 0 || 0 == wT && 0 != (Js::NumberUtilities::LuLoDbl(dbl) & 1))

{

// Return the next higher value.

if (Js::NumberUtilities::AddLu(&Js::NumberUtilities::LuLoDbl(dbl), 1))

Js::NumberUtilities::AddLu(&Js::NumberUtilities::LuHiDbl(dbl), 1);

}

}

LFail:

return dbl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

String to Double.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename EncodedChar>

double Js::NumberUtilities::StrToDbl( const EncodedChar \*psz, const EncodedChar \*\*ppchLim, bool& likelyInt )

{

ulong lu;

BIGNUM num;

BIGNUM numHi;

BIGNUM numLo;

double dbl;

double dblLo;

#if DBG

bool canUseLowPrec = false;

double dblLowPrec;

Js::NumberUtilities::LuHiDbl(dblLowPrec) = 0x7FFFFFFF;

Js::NumberUtilities::LuLoDbl(dblLowPrec) = 0xFFFFFFFF;

Assert(Js::NumberUtilities::IsNan(dblLowPrec));

#endif //DBG

// For the mantissa digits. After leaving the state machine, pchMinDig

// points to the first digit and pchLimDig points just past the last

// digit. cchDig is the number of digits. pchLimDig - pchMinDig may be

// cchDig + 1 (if there is a decimal point).

long cchDig = 0;

const EncodedChar \*pchMinDig = NULL;

const EncodedChar \*pchLimDig = NULL;

int signExp = 1; // sign of the exponent

int signMan = 0; // sign of the mantissa

long lwAdj = 0; // exponent adjustment

long lwExp = 0; // the exponent

const EncodedChar \*pchSave;

const EncodedChar \*pch = psz;

// Enter the state machine

// Initialization

LRestart:

switch (\*pch)

{

default:

if (FNzDigit(\*pch))

goto LGetLeftDig;

break;

case 'I':

// Check for the special case of [+|-]Infinity.

if (pch[1] == 'n' && pch[2] == 'f' && pch[3] == 'i' && pch[4] == 'n' && pch[5] == 'i' && pch[6] == 't' && pch[7] == 'y')

{

\*ppchLim = pch + 8;

Js::NumberUtilities::LuHiDbl(dbl) = 0x7FF00000;

Js::NumberUtilities::LuLoDbl(dbl) = 0;

goto LDone;

}

break;

case '-':

if (signMan)

break;

pch++;

signMan = -1;

goto LRestart;

case '+':

if (signMan)

break;

pch++;

signMan = +1;

goto LRestart;

case '0':

while ('0' == \*++pch)

;

goto LGetLeft;

case '.':

if (Js::NumberUtilities::IsDigit(pch[1]))

goto LGetRight;

break;

}

// Nothing digested - set the result to NaN and exit.

// We cannot use any other NaN value because of our tagged float encoding.

\*ppchLim = psz;

Js::NumberUtilities::LuHiDbl(dbl) = 0xFFF80000;

Js::NumberUtilities::LuLoDbl(dbl) = 0x00000000;

goto LDone;

LGetLeft:

// Get digits to the left of the decimal point

if (Js::NumberUtilities::IsDigit(\*pch))

{

LGetLeftDig:

pchMinDig = pch;

for (cchDig = 1; Js::NumberUtilities::IsDigit(\*++pch); cchDig++)

;

}

switch (\*pch)

{

case '.':

goto LGetRight;

case 'E':

case 'e':

goto LGetExp;

}

goto LEnd;

LGetRight:

Assert(\*pch == '.');

likelyInt = false;

pch++;

if (NULL == pchMinDig)

{

for ( ; \*pch == '0'; pch++)

lwAdj--;

pchMinDig = pch;

}

for( ; Js::NumberUtilities::IsDigit(\*pch); pch++)

{

cchDig++;

lwAdj--;

}

switch (\*pch)

{

case 'E':

case 'e':

goto LGetExp;

}

goto LEnd;

LGetExp:

pchLimDig = pch;

pchSave = pch++; // points to 'E'

if (Js::NumberUtilities::IsDigit(\*pch))

goto LGetExpDigits;

switch (\*pch)

{

case '-':

signExp = -1;

// fall-through

case '+':

pch++;

if (Js::NumberUtilities::IsDigit(\*pch))

goto LGetExpDigits;

break;

}

// back up to the 'E'

pch = pchSave;

goto LEnd;

LGetExpDigits:

for( ; Js::NumberUtilities::IsDigit(\*pch); pch++)

{

lwExp = lwExp \* 10 + (\*pch - '0');

if (lwExp > 100000000)

lwExp = 100000000;

}

LEnd:

\*ppchLim = pch;

if (cchDig == 0)

{

dbl = 0;

goto LDone;

}

if (NULL == pchLimDig)

pchLimDig = pch;

Assert(pchMinDig != NULL);

Assert(pchLimDig - pchMinDig == cchDig ||

pchLimDig - pchMinDig == cchDig + 1);

// Limit to kcchMaxSig digits.

if (cchDig > kcchMaxSig)

{

// cchDig - number of digits from the first nonzero digit in the input

// pchLimDig - at this point, this points to the character after the last digit in the input, and after the decimal

// point if the number ends with a decimal point

// lwAdj - considering that the decimal point is initially after the last digit in the input, this contains the number

// of digits the decimal point should be moved to the right to put the decimal point in the correct place,

// excluding the exponent component. Since this excludes the exponent component, at this point lwAdj is either

// zero or negative.

// Here, we are going to consider that there are only kcchMaxSig digits (effectively the same as replacing excessive

// digits with a '0' to make them insignificant, as the spec requests). So, the following need to be done:

const long numExcessiveDigits = cchDig - kcchMaxSig;

// Move pchLimDig to the left over numExcessiveDigits digits. Note that it needs to move over digits; the decimal point

// does not count as a digit. We determine if pchLimDig would jump over the decimal point by using the lwAdj value and

// if so, jump over one more character. Note also that if lwAdj is zero, there may or may not be a decimal point, and

// pchLimDig would only need to jump over a decimal point if it exists.

if (-lwAdj <= numExcessiveDigits &&

(lwAdj != 0 || pchLimDig[-1] == L'.'))

{

// Need to jump over the decimal point

--pchLimDig;

}

pchLimDig -= numExcessiveDigits;

// We previously considered that the decimal point is initially after the last digit in the input, and is moved to the

// right by lwAdj digits. Now, we're going to consider that the decimal point is initially after the kcchMaxSig'th

// digit. So, the decimal point needs to be moved to the right by numExcessiveDigits digits, so add that to lwAdj.

lwAdj += numExcessiveDigits;

cchDig = kcchMaxSig;

}

// Remove trailing zero's from mantissa

Assert(FNzDigit(\*pchMinDig));

for (;;)

{

if (\*--pchLimDig == '0')

{

cchDig--;

lwAdj++;

}

else if (\*pchLimDig != '.')

{

Assert(FNzDigit(\*pchLimDig));

pchLimDig++;

break;

}

}

Assert(pchLimDig - pchMinDig == cchDig ||

pchLimDig - pchMinDig == cchDig + 1);

if (signExp < 0)

lwExp = -lwExp;

lwExp += lwAdj;

// See if we can just use IEEE double arithmetic.

if (cchDig <= 15 && lwExp >= -22 && lwExp + cchDig <= 37)

{

// These calculations are all exact since cchDig <= 15.

if (cchDig <= 9)

{

// Can use the ALU.

for (lu = 0, pch = pchMinDig; pch < pchLimDig; pch++)

{

if (\*pch != '.')

{

Assert(Js::NumberUtilities::IsDigit(\*pch));

lu = lu \* 10 + (\*pch - '0');

}

}

dbl = lu;

}

else

{

for (dbl = 0, pch = pchMinDig; pch < pchLimDig; pch++)

{

if (\*pch != '.')

{

Assert(Js::NumberUtilities::IsDigit(\*pch));

dbl = dbl \* 10 + (\*pch - '0');

}

}

}

// This is the only (potential) rounding operation and we assume

// the compiler does the correct IEEE rounding.

if (lwExp > 0)

{

if (lwExp > 22)

{

// This one is exact. We're using the fact that cchDig < 15

// to handle exponents bigger than 22.

dbl \*= g\_rgdblTens[15 - cchDig];

Assert(lwExp - (15 - cchDig) <= 22);

dbl \*= g\_rgdblTens[lwExp - (15 - cchDig)];

}

else

dbl \*= g\_rgdblTens[lwExp];

}

else if (lwExp < 0)

dbl /= g\_rgdblTens[-lwExp];

#if DBG

// In the debug version, execute the high precision code also and

// verify that the results are the same.

canUseLowPrec = true;

dblLowPrec = dbl;

#else //!DBG

goto LDone;

#endif //!DBG

}

lwExp += cchDig;

if (lwExp >= klwMaxExp10)

{

// Overflow to infinity.

Js::NumberUtilities::LuHiDbl(dbl) = 0x7FF00000;

Js::NumberUtilities::LuLoDbl(dbl) = 0;

goto LDone;

}

if (lwExp <= klwMinExp10)

{

// Underflow to 0.

dbl = 0;

goto LDone;

}

// Convert to a big number.

Assert(pchLimDig - pchMinDig >= 0 && pchLimDig - pchMinDig <= LONG\_MAX);

num.SetFromRgchExp(pchMinDig, (long)(pchLimDig - pchMinDig), lwExp);

// If there is no error in the big number, just convert it to a double.

if (0 == num.m\_luError)

{

dbl = num.GetDbl();

#if DBG

Assert(pchLimDig - pchMinDig >= 0 && pchLimDig - pchMinDig <= LONG\_MAX);

dblLo = AdjustDbl(dbl, pchMinDig, (long)(pchLimDig - pchMinDig), lwExp);

Assert(dbl == dblLo);

#endif //DBG

goto LDone;

}

// The big number has error in it, so see if the error matters.

// Get the upper bound and lower bound. If they convert to the same

// double we're done.

numHi = num;

numHi.MakeUpperBound();

numLo = num;

numLo.MakeLowerBound();

dbl = numHi.GetDbl();

dblLo = numLo.GetDbl();

if (dbl == dblLo)

{

#if DBG

Assert(dbl == num.GetDbl());

Assert(pchLimDig - pchMinDig >= 0 && pchLimDig - pchMinDig <= LONG\_MAX);

dblLo = AdjustDbl(dbl, pchMinDig, (long)(pchLimDig - pchMinDig), lwExp);

Assert(dbl == dblLo || Js::NumberUtilities::IsNan(dblLo));

#endif //DBG

goto LDone;

}

// Need to use big integer arithmetic. There's too much error in

// our result and it's close to a boundary value. This is rare,

// but does happen, e.g.:

// x = 1.2345678901234568347913049445e+200;

//

Assert(pchLimDig - pchMinDig >= 0 && pchLimDig - pchMinDig <= LONG\_MAX);

dbl = AdjustDbl(num.GetDbl(), pchMinDig, (long)(pchLimDig - pchMinDig), lwExp);

LDone:

// This assert was removed because it would fire on VERY rare occasions. Not

// repro on all machines and very hard to repro even on machines that could repro it.

// The numbers (dblLowPrec and dbl) were different in their two least sig bits only

// which is \_probably\_ within expected error. I did not take the time to fully

// investigate whether this really does meet the ECMA spec...

//

// Assert(Js::NumberUtilities::IsNan(dblLowPrec) || dblLowPrec == dbl);

#if DBG

if(canUseLowPrec)

{

// Use the same final behavior in debug builds as for non-debug builds by using the low-precision value

dbl = dblLowPrec;

}

#endif

if (signMan < 0)

dbl = -dbl;

return dbl;

}

template double Js::NumberUtilities::StrToDbl<wchar\_t>( const wchar\_t \* psz, const wchar\_t \*\*ppchLim, bool& likelyInt );

template double Js::NumberUtilities::StrToDbl<utf8char\_t>(const utf8char\_t \* psz, const utf8char\_t \*\*ppchLim, bool& likelyInt);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Uses big integer arithmetic to get the sequence of digits.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_Success\_(return)

static BOOL FDblToRgbPrecise(double dbl, \_\_out\_ecount(kcbMaxRgb) byte \*prgb, int \*pwExp10, byte \*\*ppbLim)

{

byte bT;

BOOL fPow2;

int ib;

int clu;

int wExp10, wExp2, w1, w2;

int c2Num, c2Den, c5Num, c5Den;

double dblT;

Js::BigInt biNum, biDen, biHi, biLo;

Js::BigInt \*pbiLo;

Js::BigInt biT;

ulong rglu[2];

// Caller should take care of 0, negative and non-finite values.

Assert(Js::NumberUtilities::IsFinite(dbl));

Assert(0 < dbl);

// Init the Denominator, Hi error and Lo error bigints.

rglu[0] = 1;

AssertVerify(biDen.FInitFromRglu(rglu, 1));

AssertVerify(biHi.FInitFromRglu(rglu, 1));

wExp2 = (int)(((Js::NumberUtilities::LuHiDbl(dbl) & 0x7FF00000) >> 20) - 1075);

rglu[1] = Js::NumberUtilities::LuHiDbl(dbl) & 0x000FFFFF;

rglu[0] = Js::NumberUtilities::LuLoDbl(dbl);

clu = 2;

fPow2 = FALSE;

if (wExp2 == -1075)

{

// dbl is denormalized.

Assert(0 == (Js::NumberUtilities::LuHiDbl(dbl) & 0x7FF00000));

if (0 == rglu[1])

clu = 1;

// Get dblT such that dbl / dblT is a power of 2 and 1 <= dblT < 2.

// First multiply by a power of 2 to get a normalized value.

Js::NumberUtilities::LuHiDbl(dblT) = 0x4FF00000;

Js::NumberUtilities::LuLoDbl(dblT) = 0;

dblT \*= dbl;

Assert(0 != (Js::NumberUtilities::LuHiDbl(dblT) & 0x7FF00000));

// This is the power of 2.

w1 = (int)((Js::NumberUtilities::LuHiDbl(dblT) & 0x7FF00000) >> 20) - (256 + 1023);

Js::NumberUtilities::LuHiDbl(dblT) &= 0x000FFFFF;

Js::NumberUtilities::LuHiDbl(dblT) |= 0x3FF00000;

// Adjust wExp2 because we don't have the implicit bit.

wExp2++;

}

else

{

// Get dblT such that dbl / dblT is a power of 2 and 1 <= dblT < 2.

// First multiply by a power of 2 to get a normalized value.

dblT = dbl;

Js::NumberUtilities::LuHiDbl(dblT) &= 0x000FFFFF;

Js::NumberUtilities::LuHiDbl(dblT) |= 0x3FF00000;

// This is the power of 2.

w1 = wExp2 + 52;

if (0 == rglu[0] && 0 == rglu[1] && wExp2 > -1074)

{

// Power of 2 bigger than smallest normal. The next smaller

// representable value is closer than the next larger value.

rglu[1] = 0x00200000;

wExp2--;

fPow2 = TRUE;

}

else

{

// Normalized and not a power of 2 or the smallest normal. The

// representable values on either side are the same distance away.

rglu[1] |= 0x00100000;

}

}

// Compute an approximation to the base 10 log. This is borrowed from

// David Gay's paper.

Assert(1 <= dblT && dblT < 2);

dblT = (dblT - 1.5) \* 0.289529654602168 + 0.1760912590558 +

w1 \* 0.301029995663981;

wExp10 = (int)dblT;

if (dblT < 0 && dblT != wExp10)

wExp10--;

if (wExp2 >= 0)

{

c2Num = wExp2;

c2Den = 0;

}

else

{

c2Num = 0;

c2Den = -wExp2;

}

if (wExp10 >= 0)

{

c5Num = 0;

c5Den = wExp10;

c2Den += wExp10;

}

else

{

c2Num -= wExp10;

c5Num = -wExp10;

c5Den = 0;

}

if (c2Num > 0 && c2Den > 0)

{

w1 = c2Num < c2Den ? c2Num : c2Den;

c2Num -= w1;

c2Den -= w1;

}

// We need a bit for the Hi and Lo values.

c2Num++;

c2Den++;

// Initialize biNum and multiply by powers of 5.

if (c5Num > 0)

{

Assert(0 == c5Den);

if (!biHi.FMulPow5(c5Num))

goto LFail;

if (!biNum.FInitFromBigint(&biHi))

goto LFail;

if (clu == 1)

{

if (!biNum.FMulAdd(rglu[0], 0))

goto LFail;

}

else

{

if (!biNum.FMulAdd(rglu[1], 0))

goto LFail;

if (!biNum.FShiftLeft(32))

goto LFail;

if (rglu[0] != 0)

{

if (!biT.FInitFromBigint(&biHi))

goto LFail;

if (!biT.FMulAdd(rglu[0], 0))

goto LFail;

if (!biNum.FAdd(&biT))

goto LFail;

}

}

}

else

{

Assert(clu <= 2);

AssertVerify(biNum.FInitFromRglu(rglu, clu));

if (c5Den > 0 && !biDen.FMulPow5(c5Den))

goto LFail;

}

// BIGINT::DivRem only works if the 4 high bits of the divisor are 0.

// It works most efficiently if there are exactly 4 zero high bits.

// Adjust c2Den and c2Num to guarantee this.

w1 = Js::NumberUtilities::CbitZeroLeft(biDen.Lu(biDen.Clu() - 1));

w1 = (w1 + 28 - c2Den) & 0x1F;

c2Num += w1;

c2Den += w1;

// Multiply by powers of 2.

Assert(c2Num > 0 && c2Den > 0);

if (!biNum.FShiftLeft(c2Num))

goto LFail;

if (c2Num > 1 && !biHi.FShiftLeft(c2Num - 1))

goto LFail;

if (!biDen.FShiftLeft(c2Den))

goto LFail;

Assert(0 == (biDen.Lu(biDen.Clu() - 1) & 0xF0000000));

Assert(0 != (biDen.Lu(biDen.Clu() - 1) & 0x08000000));

// Get pbiLo and handle the power of 2 case where biHi needs to be doubled.

if (fPow2)

{

pbiLo = &biLo;

if (!pbiLo->FInitFromBigint(&biHi))

goto LFail;

if (!biHi.FShiftLeft(1))

goto LFail;

}

else

pbiLo = &biHi;

for (ib = 0; ib < kcbMaxRgb; )

{

bT = (byte)biNum.DivRem(&biDen);

if (ib == 0 && bT == 0)

{

// Our estimate of wExp10 was too big. Oh well.

wExp10--;

goto LSkip;

}

// w1 = sign(biNum - \*pbiLo).

w1 = biNum.Compare(pbiLo);

// w2 = sign(biNum + biHi - biDen).

if (biDen.Compare(&biHi) < 0)

w2 = 1;

else

{

// REVIEW : is there a faster way to do this?

biT.FInitFromBigint(&biDen);

biT.Subtract(&biHi);

w2 = biNum.Compare(&biT);

}

// if (biNum + biHi == biDen && even)

if (0 == w2 && 0 == (Js::NumberUtilities::LuLoDbl(dbl) & 1))

{

// Rounding up this digit produces exactly (biNum + biHi) which

// StrToDbl will round down to dbl.

if (bT == 9)

goto LRoundUp9;

if (w1 > 0)

bT++;

Assert(ib < kcbMaxRgb);

prgb[ib++] = bT;

break;

}

// if (biNum < \*pbiLo || biNum == \*pbiLo && even)

if (w1 < 0 || 0 == w1 && 0 == (Js::NumberUtilities::LuLoDbl(dbl) & 1))

{

// if (biNum + biHi > biDen)

if (w2 > 0)

{

// Decide whether to round up.

if (!biNum.FShiftLeft(1))

goto LFail;

w2 = biNum.Compare(&biDen);

if ((w2 > 0 || w2 == 0 && (bT & 1)) && bT++ == 9)

goto LRoundUp9;

}

Assert(ib < kcbMaxRgb);

prgb[ib++] = bT;

break;

}

// if (biNum + biHi > biDen)

if (w2 > 0)

{

// Round up and be done with it.

if (bT != 9)

{

Assert(ib < kcbMaxRgb);

prgb[ib++] = bT + 1;

break;

}

LRoundUp9:

while (ib > 0)

{

if (prgb[--ib] != 9)

{

prgb[ib++]++;

goto LReturn;

}

}

wExp10++;

Assert(ib < kcbMaxRgb);

prgb[ib++] = 1;

break;

}

// Save the digit.

Assert(ib < kcbMaxRgb);

prgb[ib++] = bT;

LSkip:

if (!biNum.FMulAdd(10, 0))

goto LFail;

if (!biHi.FMulAdd(10, 0))

goto LFail;

if (pbiLo != &biHi && !pbiLo->FMulAdd(10, 0))

goto LFail;

}

LReturn:

\*pwExp10 = wExp10 + 1;

AnalysisAssert(ib <= kcbMaxRgb);

\*ppbLim = &prgb[ib];

return TRUE;

LFail:

return FALSE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Get mantissa bytes (BCD).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_Success\_(return)

static BOOL FDblToRgbFast(double dbl, \_Out\_writes\_to\_(kcbMaxRgb, (\*ppbLim - prgb)) byte \*prgb, int \*pwExp10, byte \*\*ppbLim)

{

int ib;

int iT;

ulong luT;

ulong luScale;

const BIGNUM \*pnum;

byte bHH = 0, bHL = 0, bLH = 0, bLL = 0;

ulong luHH, luHL, luLH, luLL;

BIGNUM numHH, numHL, numLH, numLL, numBase;

int wExp2;

int wExp10 = 0;

// Caller should take care of 0, negative and non-finite values.

Assert(Js::NumberUtilities::IsFinite(dbl));

Assert(0 < dbl);

// Get numHH and numLL such that numLL < dbl < numHH and the

// difference between adjacent values is half the distance to the next

// representable value (in a double).

wExp2 = (int)((Js::NumberUtilities::LuHiDbl(dbl) >> 20) & 0x07FF);

if (wExp2 > 0)

{

// See if dbl is a small integer.

if (wExp2 >= 1023 && wExp2 <= 1075 && dbl == floor(dbl))

goto LSmallInt;

// Normalized

numBase.m\_lu2 = 0x80000000 | ((Js::NumberUtilities::LuHiDbl(dbl) & 0x000FFFFFF) << 11) |

(Js::NumberUtilities::LuLoDbl(dbl) >> 21);

numBase.m\_lu1 = Js::NumberUtilities::LuLoDbl(dbl) << 11;

numBase.m\_lu0 = 0;

numBase.m\_wExp = wExp2 - 1022;

numBase.m\_luError = 0;

// Get the upper bound

numHH = numBase;

numHH.m\_lu1 |= (1 << 10);

// Get the lower bound. A power of 2 must be special cased.

numLL = numBase;

if (0x80000000 == numLL.m\_lu2 && 0 == numLL.m\_lu1)

{

// Subtract (0x00000000, 0x00000200, 0x00000000). Same as adding

// (0xFFFFFFFF, 0xFFFFFE00, 0x00000000)

luT = 0xFFFFFE00;

}

else

{

// Subtract (0x00000000, 0x00000400, 0x00000000). Same as adding

// (0xFFFFFFFF, 0xFFFFFC00, 0x00000000)

luT = 0xFFFFFC00;

}

if (!Js::NumberUtilities::AddLu(&numLL.m\_lu1, luT))

{

Js::NumberUtilities::AddLu(&numLL.m\_lu2, 0xFFFFFFFF);

if (0 == (0x80000000 & numLL.m\_lu2))

numLL.Normalize();

}

}

else

{

// Denormal

numBase.m\_lu2 = Js::NumberUtilities::LuHiDbl(dbl) & 0x000FFFFF;

numBase.m\_lu1 = Js::NumberUtilities::LuLoDbl(dbl);

numBase.m\_lu0 = 0;

numBase.m\_wExp = -1010;

numBase.m\_luError = 0;

// Get the upper bound

numHH = numBase;

numHH.m\_lu0 = 0x80000000;

// Get the lower bound

numLL = numHH;

if (!Js::NumberUtilities::AddLu(&numLL.m\_lu1, 0xFFFFFFFF))

Js::NumberUtilities::AddLu(&numLL.m\_lu2, 0xFFFFFFFF);

numBase.Normalize();

numHH.Normalize();

numLL.Normalize();

}

// Multiply by powers of ten until 0 < numHH.m\_wExp < 32.

if (numHH.m\_wExp >= 32)

{

iT = (numHH.m\_wExp - 25) \* 15 / -g\_rgnumNeg[45].m\_wExp;

Assert(iT >= 0 && iT < 16);

\_\_analysis\_assume(iT >= 0 && iT < 16);

if (iT > 0)

{

pnum = &g\_rgnumNeg[30 + iT];

Assert(numHH.m\_wExp + pnum->m\_wExp > 1);

numHH.Mul(pnum);

numLL.Mul(pnum);

wExp10 += iT \* 32;

}

if (numHH.m\_wExp >= 32)

{

iT = (numHH.m\_wExp - 25) \* 32 / -g\_rgnumNeg[31].m\_wExp;

Assert(iT > 0 && iT <= 32);

pnum = &g\_rgnumNeg[iT - 1];

Assert(numHH.m\_wExp + pnum->m\_wExp > 1);

numHH.Mul(pnum);

numLL.Mul(pnum);

wExp10 += iT;

}

}

else if (numHH.m\_wExp < 1)

{

iT = (25 - numHH.m\_wExp) \* 15 / g\_rgnumPos[45].m\_wExp;

Assert(iT >= 0 && iT < 16);

\_\_analysis\_assume(iT >= 0 && iT < 16);

if (iT > 0)

{

pnum = &g\_rgnumPos[30 + iT];

Assert(numHH.m\_wExp + pnum->m\_wExp <= 32);

numHH.Mul(pnum);

numLL.Mul(pnum);

wExp10 -= iT \* 32;

}

if (numHH.m\_wExp < 1)

{

iT = (25 - numHH.m\_wExp) \* 32 / g\_rgnumPos[31].m\_wExp;

Assert(iT > 0 && iT <= 32);

pnum = &g\_rgnumPos[iT - 1];

Assert(numHH.m\_wExp + pnum->m\_wExp <= 32);

numHH.Mul(pnum);

numLL.Mul(pnum);

wExp10 -= iT;

}

}

Assert(numHH.m\_wExp > 0 && numHH.m\_wExp < 32);

// Get the upper and lower bounds for these.

numHL = numHH;

numHH.MakeUpperBound();

numHL.MakeLowerBound();

luHH = numHH.LuMod1();

luHL = numHL.LuMod1();

numLH = numLL;

numLH.MakeUpperBound();

numLL.MakeLowerBound();

luLH = numLH.LuMod1();

luLL = numLL.LuMod1();

Assert(luLL <= luLH && luLH <= luHL && luHL <= luHH);

// Find the starting scale

luScale = 1;

if (luHH >= 100000000)

{

luScale = 100000000;

wExp10 += 8;

}

else

{

if (luHH >= 10000)

{

luScale = 10000;

wExp10 += 4;

}

if (luHH >= 100 \* luScale)

{

luScale \*= 100;

wExp10 += 2;

}

}

if (luHH >= 10 \* luScale)

{

luScale \*= 10;

wExp10++;

}

wExp10++;

Assert(luHH >= luScale && luHH / luScale < 10);

for (ib = 0; ib < kcbMaxRgb; )

{

Assert(luLL <= luHH);

bHH = (byte)(luHH / luScale);

luHH %= luScale;

bLL = (byte)(luLL / luScale);

luLL %= luScale;

if (bHH != bLL)

break;

Assert(luHH != 0 || !numHH.FZero());

Assert(ib < kcbMaxRgb);

prgb[ib++] = bHH;

if (1 == luScale)

{

// Multiply by 10^8.

luScale = 10000000;

numHH.Mul(&g\_rgnumPos[7]);

numHH.MakeUpperBound();

luHH = numHH.LuMod1();

if (luHH >= 100000000)

goto LFail;

numHL.Mul(&g\_rgnumPos[7]);

numHL.MakeLowerBound();

luHL = numHL.LuMod1();

numLH.Mul(&g\_rgnumPos[7]);

numLH.MakeUpperBound();

luLH = numLH.LuMod1();

numLL.Mul(&g\_rgnumPos[7]);

numLL.MakeLowerBound();

luLL = numLL.LuMod1();

}

else

luScale /= 10;

}

// LL and HH diverged. Get the digit values for LH and HL.

Assert(0 <= bLL && bLL < bHH && bHH <= 9);

bLH = (byte)((luLH / luScale) % 10);

luLH %= luScale;

bHL = (byte)((luHL / luScale) % 10);

luHL %= luScale;

if (bLH >= bHL)

goto LFail;

// LH and HL also diverged.

// We can get by with one fewer digit if: LL == LH and bLH is zero

// and the current value of LH is zero and the least significant bit of

// the double is zero. In this case, we have exactly the digit sequence

// for the original numLL and IEEE and will rounds numLL up to the double.

if (0 == bLH && 0 == luLH && numLH.FZero() && 0 == (Js::NumberUtilities::LuLoDbl(dbl) & 1))

;

else if (bHL - bLH > 1)

{

Assert(ib < kcbMaxRgb);

if(!(ib < kcbMaxRgb))

goto LFail;

// HL and LH differ by at least two in this digit, so split

// the difference.

prgb[ib++] = (bHL + bLH + 1) / 2;

}

else if (0 != luHL || !numHL.FZero() || 0 == (Js::NumberUtilities::LuLoDbl(dbl) & 1))

{

Assert(ib < kcbMaxRgb);

if(!(ib < kcbMaxRgb))

goto LFail;

// We can just use bHL because this guarantees that we're bigger than

// LH and less than HL, so must convert to the double.

prgb[ib++] = bHL;

}

else

goto LFail;

Assert(ib <= kcbMaxRgb);

\*pwExp10 = wExp10;

\*ppbLim = &prgb[ib];

return TRUE;

LSmallInt:

// dbl should be an integer from 1 to (2^53 - 1).

Assert(dbl == floor(dbl) && 1 <= dbl && dbl <= 9007199254740991.0L);

iT = 0;

if (dbl >= g\_rgdblTens[iT + 8])

iT += 8;

if (dbl >= g\_rgdblTens[iT + 4])

iT += 4;

if (dbl >= g\_rgdblTens[iT + 2])

iT += 2;

if (dbl >= g\_rgdblTens[iT + 1])

iT += 1;

Assert(iT >= 0 && iT <= 15);

Assert(dbl >= g\_rgdblTens[iT] && dbl < g\_rgdblTens[iT + 1]);

\*pwExp10 = iT + 1;

for (ib = 0; 0 != dbl && ib < kcbMaxRgb && 0 <= iT; iT--)

{

Assert(iT >= 0);

bHH = (byte)(dbl / g\_rgdblTens[iT]);

dbl -= bHH \* g\_rgdblTens[iT];

Assert(dbl == floor(dbl) && 0 <= dbl && dbl < g\_rgdblTens[iT]);

prgb[ib++] = bHH;

}

\*ppbLim = &prgb[ib];

return TRUE;

LFail:

return FALSE;

}

static BOOL FormatDigits(\_In\_reads\_(pbLim - pbSrc) byte \*pbSrc, byte \*pbLim, int wExp10, \_Out\_writes\_(cchDst) OLECHAR \*pchDst, int cchDst)

{

AssertArrMem(pbSrc, pbLim - pbSrc);

AnalysisAssert(pbLim > pbSrc);

if (pbLim <= pbSrc)

{

Assert(0);

return FALSE;

}

// check the expected size of the resulting string...

size\_t nCount;

if ((wExp10 <= -6) ||(wExp10 > 21))

{

nCount = (pbLim - pbSrc) + 6;

if (wExp10 >= 100)

nCount += 2;

else if (wExp10 >= 10)

nCount += 1;

}

else if (wExp10 <= 0)

nCount = (pbLim - pbSrc) + 3 + abs(wExp10);

else

nCount = (pbLim - pbSrc) + 1 + wExp10;

if ((int)nCount >= cchDst)

{

Assert(0);

return FALSE;

}

if (wExp10 <= -6 || wExp10 > 21)

{

// Exponential notation - first digit

\*pchDst++ = \*pbSrc++ + '0';

if (pbSrc < pbLim)

{

// Decimal point and remaining digits

\*pchDst++ = '.';

while (pbSrc < pbLim)

\*pchDst++ = \*pbSrc++ + '0';

}

// 'e' and exponent sign

\*pchDst++ = 'e';

if (--wExp10 < 0)

{

\*pchDst++ = '-';

wExp10 = -wExp10;

}

else

\*pchDst++ = '+';

// Exponent Digits

Assert(wExp10 < 1000);

if (wExp10 >= 100)

{

\*pchDst++ = (wchar\_t)('0' + wExp10 / 100);

wExp10 %= 100;

\*pchDst++ = (wchar\_t)('0' + wExp10 / 10);

wExp10 %= 10;

}

else if (wExp10 >= 10)

{

\*pchDst++ = (wchar\_t)('0' + wExp10 / 10);

wExp10 %= 10;

}

#pragma prefast(suppress:26014, "We have calculate the check the buffer size above already")

\*pchDst++ = (wchar\_t)('0' + wExp10);

\*pchDst = 0;

}

else if (wExp10 <= 0)

{

// Just fractional stuff

\*pchDst++ = '0';

#pragma prefast(suppress:26014, "We have calculate the check the buffer size above already")

\*pchDst++ = '.';

for( ; wExp10 < 0; wExp10++)

\*pchDst++ = '0';

while (pbSrc < pbLim)

\*pchDst++ = \*pbSrc++ + '0';

\*pchDst = 0;

}

else

{

// Stuff to the left of the decimal point

while (pbSrc < pbLim)

{

\*pchDst++ = \*pbSrc++ + '0';

if (--wExp10 == 0 && pbSrc < pbLim)

\*pchDst++ = '.';

}

for( ; wExp10 > 0; wExp10--)

\*pchDst++ = '0';

\*pchDst = 0;

}

return TRUE;

}

\_\_success(return <= nDstBufSize)

#pragma prefast(suppress:6101, "when return value is > nDstBufSize, the pchDst is not initialized. Prefast doesn't seems to pick that up in the annotation")

static int FormatDigitsFixed(byte \*pbSrc, byte \*pbLim, int wExp10, int nFractionDigits, \_\_out\_ecount\_part(nDstBufSize, return) wchar\_t \*pchDst, int nDstBufSize)

{

AnalysisAssert(pbLim > pbSrc);

AssertArrMem(pbSrc, pbLim - pbSrc);

AnalysisAssert(nFractionDigits >= -1);

// nFractionDigits == -1 => print exactly as many fractional digits as necessary : no trailing 0's.

int n = 1; // the no. of chars. in the result.

if (wExp10 <= 0)

{

// Just fractional stuff

if( nFractionDigits < 0 )

{

// Set nFractionDigits such that we get all the significant digits and no trailing zeros

AnalysisAssert(pbLim - pbSrc < INT\_MAX);

nFractionDigits = -wExp10 + (int)(pbLim - pbSrc);

}

n++; // for '0'

if( nFractionDigits > 0 )

{

n += nFractionDigits + 1;

}

if( nDstBufSize >= n )

{

\*pchDst++ = '0';

if( nFractionDigits > 0 )

{

\*pchDst++ = '.';

for( ; wExp10 < 0 && nFractionDigits > 0; wExp10++, nFractionDigits--)

\*pchDst++ = '0';

for( ;pbSrc < pbLim && nFractionDigits > 0; nFractionDigits--)

\*pchDst++ = \*pbSrc++ + '0';

while(nFractionDigits-- > 0)

\*pchDst++ = '0';

}

\*pchDst = 0;

}

}

else

{

n += wExp10; // chars to the left of the decimal point.

if( nFractionDigits < 0 )

{

// Set nFractionDigits such that we get all the significant digits and no trailing zeros

nFractionDigits = (pbLim - pbSrc <= wExp10) ? 0 : (int)(pbLim - pbSrc) - wExp10;

}

if( nFractionDigits > 0)

n += nFractionDigits + 1;

if( nDstBufSize >= n )

{

// Stuff to the left of the decimal point

for (;pbSrc < pbLim && wExp10 > 0; wExp10-- )

{

\*pchDst++ = \*pbSrc++ + '0';

}

if(wExp10 > 0)

{

for( ; wExp10 > 0; wExp10--)

\*pchDst++ = '0';

}

//Stuff to the right of the decimal point

if (nFractionDigits > 0)

{

\*pchDst++ = '.';

for (;pbSrc < pbLim && nFractionDigits > 0; nFractionDigits-- )

{

\*pchDst++ = \*pbSrc++ + '0';

}

// Pad with 0's at the end to get the required number of fractional digits

while( nFractionDigits-- > 0)

\*pchDst++ = '0';

}

\*pchDst = 0;

}

}

return n;

}

\_\_success(return <= cchDst)

static int FormatDigitsExponential(

byte \* pbSrc,

byte \* pbLim,

int wExp10,

int nFractionDigits,

\_\_out\_ecount\_part(cchDst,return) wchar\_t \* pchDst,

int cchDst

)

{

AnalysisAssert(pbLim > pbSrc);

Assert(pbLim - pbSrc <= kcbMaxRgb);

AssertArrMem(pbSrc, pbLim - pbSrc);

AssertArrMem(pchDst, cchDst);

AnalysisAssert(wExp10 < 1000);

\_\_analysis\_assume(pbLim > pbSrc);

\_\_analysis\_assume(pbLim - pbSrc <= kcbMaxRgb);

\_\_analysis\_assume(wExp10 < 1000);

int n = 1; // first digit

if (nFractionDigits < 0) // output as many fractional digits as we can

{

int cch = (int)(pbLim - (1 + pbSrc)); // 1 == first digit

if (cch > 0)

{

n += (1 + cch); // 1 == '.'

}

}

else if (nFractionDigits > 0)

{

n += (1 + nFractionDigits); // 1 == '.'

}

// 'e' and exponent sign

n += 2;

// Exponent Digits

int wExp10Abs = ((wExp10-1) >= 0) ? (wExp10-1) : -(wExp10-1);

if (wExp10Abs >= 100)

{

n += 3;

}

else if (wExp10Abs >= 10)

{

n += 2;

}

else

{

n += 1;

}

n++; // null terminator

if (cchDst < n) return n;

#if DBG // save pchDst to validate n

wchar\_t \* pchDstStart = pchDst;

#endif

// First digit

\*pchDst++ = '0' + \*pbSrc++;

if (nFractionDigits < 0) // output as many fractional digits as we can

{

if (pbSrc < pbLim)

{

// Decimal point and remaining digits

\*pchDst++ = '.';

do

{

\*pchDst++ = '0' + \*pbSrc++;

} while (pbSrc < pbLim);

}

}

else if (nFractionDigits > 0)

{

// Decimal point and remaining digits

\*pchDst++ = '.';

for ( ; pbSrc < pbLim && nFractionDigits > 0; nFractionDigits--)

\*pchDst++ = '0' + \*pbSrc++;

while (nFractionDigits-- > 0)

\*pchDst++ = '0';

}

// 'e' and exponent sign

\*pchDst++ = 'e';

if (--wExp10 < 0)

{

#pragma prefast(suppress:26014, "We have calculate the check the buffer size above already")

\*pchDst++ = '-';

wExp10 = -wExp10;

}

else

\*pchDst++ = '+';

// Exponent Digits

if (wExp10 >= 100)

{

\*pchDst++ = (wchar\_t)('0' + wExp10 / 100);

wExp10 %= 100;

\*pchDst++ = (wchar\_t)('0' + wExp10 / 10);

wExp10 %= 10;

}

else if (wExp10 >= 10)

{

\*pchDst++ = (wchar\_t)('0' + wExp10 / 10);

wExp10 %= 10;

}

\*pchDst++ = (wchar\_t)('0' + wExp10);

\*pchDst = 0;

Assert(1 + pchDst - pchDstStart == n);

Assert(1 + pchDst <= pchDstStart + cchDst);

return n;

}

/\*

\*RoundTo:Rounds off the BCD representation of a number to a specified number of digits

\* The input number is contained in [pbSrc .. pbSrc + pbLim -1] and the output number

\* will be contained in [pbDst .. \*ppbLimRes - 1]. 'nDigits' is the number of digits

\* to which the number should be rounded up.

\*

\* Return value: 1 if an extra leading 1 needed to be added, 0 otherwise.

\*/

#pragma prefast(suppress:6101)

static int RoundTo(byte \*pbSrc, byte \*pbLim, int nDigits, \_\_out\_bcount(nDigits+1) byte \*pbDst, byte \*\*ppbLimRes )

{

AnalysisAssert(pbLim > pbSrc);

AssertArrMem(pbSrc, pbLim - pbSrc);

AnalysisAssert(nDigits >= 0);

int retVal = 0;

if ((pbLim - pbSrc) < 0)

{

AnalysisAssert(FALSE);

return 0;

}

if( pbLim - pbSrc <= nDigits )

{

// no change required

js\_memcpy\_s( pbDst, nDigits + 1, pbSrc, pbLim - pbSrc );

\*ppbLimRes = pbDst + (pbLim - pbSrc);

}

else

{

int i = nDigits;

if( pbSrc[i] >= 5 )

{

// Add 1 to the BCD representation.

for( i = nDigits - 1; i >= 0; i-- )

{

if( pbSrc[i] + 1 > 9 )

pbDst[i] = 0;

else

{

pbDst[i] = pbSrc[i] + 1;

break;

}

}

if( i < 0 && pbDst[0] == 0 )

{

// An extra leading '1' is required. Move the number in pbDst to the right

// and tack it on.

memmove(pbDst + 1, pbDst, nDigits);

pbDst[0] = 1;

retVal = 1;

}

}

if( i > 0 )

js\_memcpy\_s( pbDst, nDigits + 1, pbSrc, i );

\*ppbLimRes = pbDst + nDigits;

}

return retVal;

}

/\*

\* Format a number according to the given 'FormatType'. Used by

\* the toFixed, toExponential and toPrecision methods.

\* If 'ft' is FormatFixed or FormatExponential then nDigits is the

\* number of fractional digits. If ft is FormatPrecision then nDigits

\* is the precision.

\*

\* Returns the number of chars. in the result. If 'nDstBufSize'

\* is less than this number, no data is written to the buffer 'pchDst'.

\*/

int Js::NumberUtilities::FDblToStr(double dbl, Js::NumberUtilities::FormatType ft, int nDigits, \_\_out\_ecount(cchDst) wchar\_t \*pchDst, int cchDst)

{

int n = 0; // the no. of chars in the result.

int wExp10;

byte rgb[kcbMaxRgb];

byte \*pbLim;

if (!Js::NumberUtilities::IsFinite(dbl))

{

if (Js::NumberUtilities::IsNan(dbl))

{

n = 4; //(int)wcslen(OLESTR("NaN")) + 1;

if( cchDst >= n )

wcscpy\_s(pchDst, cchDst, L"NaN");

}

else

{

n = 9; //(int)wcslen(OLESTR("Infinity")) + 1;

int neg = 0;

if (dbl < 0 )

{

neg = 1;

n++;

}

if( cchDst >= n )

{

if (neg)

\*pchDst++ = '-';

wcscpy\_s(pchDst, cchDst - neg, L"Infinity");

}

}

return n;

}

if (0 == dbl)

{

rgb[0] = 0;

pbLim = &rgb[1];

wExp10 = 1;

}

else

{

// Handle the sign.

if (Js::NumberUtilities::LuHiDbl(dbl) & 0x80000000)

{

n++;

if( cchDst >= n)

{

\*pchDst++ = '-';

cchDst--;

}

Js::NumberUtilities::LuHiDbl(dbl) &= 0x7FFFFFFF;

}

if (!FDblToRgbFast(dbl, rgb, &wExp10, &pbLim) &&

!FDblToRgbPrecise(dbl, rgb, &wExp10, &pbLim))

{

AssertMsg(FALSE, "Failure in FDblToRgbPrecise");

return FALSE;

}

}

// We have to round up and truncate the BCD representation of the mantissa

// to the length required by the format.

byte rgbAdj[kcbMaxRgb];

byte \*pbLimAdj = NULL;

switch(ft)

{

case Js::NumberUtilities::FormatFixed:

if( nDigits >= 0 )

{

//Either session pointer is null or session is in compat mode switch to compat handling

if ((wExp10 + nDigits) > 0)

{

Assert(wExp10 + nDigits + 1 <= kcbMaxRgb);

wExp10 += RoundTo(rgb, pbLim, wExp10 + nDigits, rgbAdj, &pbLimAdj);

}

else

{

//Special case: When negative power of 10 is more than most significant digit.

if( rgb[0] >= 5 )

{

rgbAdj[0] = 1;

wExp10 += 1;

}

else

rgbAdj[0] = 0;

pbLimAdj = rgbAdj + 1;

}

}

else

RoundTo( rgb, pbLim, kcbMaxRgb-1, rgbAdj, &pbLimAdj );

n += FormatDigitsFixed(rgbAdj, pbLimAdj, wExp10, nDigits, pchDst, cchDst);

break;

case Js::NumberUtilities::FormatExponential:

if (nDigits >= 0)

{

Assert(nDigits + 2 <= kcbMaxRgb);

wExp10 += RoundTo(rgb, pbLim, nDigits + 1, rgbAdj, &pbLimAdj);

}

else

RoundTo( rgb, pbLim, kcbMaxRgb-1, rgbAdj, &pbLimAdj );

n += FormatDigitsExponential(rgbAdj, pbLimAdj, wExp10 , nDigits, pchDst, cchDst);

break;

case Js::NumberUtilities::FormatPrecision:

Assert(nDigits + 1 <= kcbMaxRgb);

wExp10 += RoundTo( rgb, pbLim, nDigits, rgbAdj, &pbLimAdj );

// NOTE: the 'e' in the toPrecision algorithm in the ECMA standard is equal to wExp - 1.

if( wExp10 - 1 < -6 || wExp10 - 1 >= nDigits )

n += FormatDigitsExponential(rgbAdj, pbLimAdj, wExp10, nDigits - 1, pchDst, cchDst);

else

n += FormatDigitsFixed(rgbAdj, pbLimAdj, wExp10, nDigits - wExp10, pchDst, cchDst);

break;

}

return n;

}

BOOL Js::NumberUtilities::FDblToStr(double dbl, \_\_out\_ecount(cchDst) wchar\_t \*pchDst, int cchDst)

{

if (!Js::NumberUtilities::IsFinite(dbl))

{

if (Js::NumberUtilities::IsNan(dbl))

return 0 == wcscpy\_s(pchDst, cchDst, L"NaN");

else

{

if (dbl < 0)

{

if (cchDst < 10) return FALSE;

\*pchDst++ = '-';

cchDst--;

}

return 0 == wcscpy\_s(pchDst, cchDst, L"Infinity");

}

}

if (0 == dbl)

{

if (cchDst < 2) return FALSE;

\*pchDst++ = '0';

\*pchDst = 0;

return TRUE;

}

return FNonZeroFiniteDblToStr(dbl, pchDst, cchDst);

}

BOOL Js::NumberUtilities::FNonZeroFiniteDblToStr(double dbl, \_\_out\_ecount(cchDst) OLECHAR \*pchDst, int cchDst)

{

int wExp10;

byte rgb[kcbMaxRgb];

byte \*pbLim;

Assert(Js::NumberUtilities::IsFinite(dbl));

Assert(dbl != 0);

// Handle the sign.

if (Js::NumberUtilities::LuHiDbl(dbl) & 0x80000000)

{

if (cchDst < 2) return FALSE;

\*pchDst++ = '-';

cchDst--;

Js::NumberUtilities::LuHiDbl(dbl) &= 0x7FFFFFFF;

}

#if DBG

double dblT;

const wchar\_t \*pch;

// In Debug, always call FDblToRgbPrecise and verify that it converts back.

if (FDblToRgbPrecise(dbl, rgb, &wExp10, &pbLim))

{

if (FormatDigits(rgb, pbLim, wExp10, pchDst, cchDst))

{

bool likelyInt = true;

dblT = StrToDbl<wchar\_t>(pchDst, &pch,likelyInt);

Assert(0 == \*pch);

Assert(dblT == dbl);

}

else

AssertMsg(FALSE, "Failure in FormatDigits");

}

else

AssertMsg(FALSE, "Failure in FDblToRgbPrecise");

#endif //DBG

if (!FDblToRgbFast(dbl, rgb, &wExp10, &pbLim) &&

!FDblToRgbPrecise(dbl, rgb, &wExp10, &pbLim))

{

AssertMsg(FALSE, "Failure in FDblToRgbPrecise");

return FALSE;

}

if (!FormatDigits(rgb, pbLim, wExp10, pchDst, cchDst))

{

AssertMsg(FALSE, "Failure in FormatDigits");

return FALSE;

}

#if DBG

bool likelyInt = true;

dblT = StrToDbl<wchar\_t>(pchDst, &pch, likelyInt);

Assert(0 == \*pch);

Assert(dblT == dbl);

#endif //DBG

return TRUE;

}

// Maximum number of digits to show for each base.

static const int g\_rgcchSig[] =

{

00,00,53,34,27,24,22,20,19,18,

17,17,16,16,15,15,14,14,14,14,

14,13,13,13,13,13,13,12,12,12,

12,12,12,12,12,12,12

};

//

// Convert a non-Nan, non-Zero, non-Infinite double value to string. (Moved from JavascriptNumber.cpp).

//

\_Success\_(return)

BOOL Js::NumberUtilities::FNonZeroFiniteDblToStr(double dbl, \_In\_range\_(2, 36) int radix, \_Out\_writes\_(nDstBufSize) WCHAR\* psz, int nDstBufSize)

{

Assert(!Js::NumberUtilities::IsNan(dbl));

Assert(dbl != 0);

Assert(Js::NumberUtilities::IsFinite(dbl));

Assert(radix != 10);

Assert(radix >= 2 && radix <= 36);

// convert to string with radix

//( back compat port of FDblToStrRadix())

int cbitDigit;

double valueDen, valueT;

int wExp2, wExp, wDig;

int maxOutDigits, cchSig, cch;

int len = nDstBufSize;

wchar\_t \* ppsz = psz;

if (0x80000000 & Js::NumberUtilities::LuHiDbl(dbl))

{

\*ppsz++ = '-';

len--;

Js::NumberUtilities::LuHiDbl(dbl) &= 0x7FFFFFFF;

}

switch (radix)

{

// We special case log computations for powers of 2.

case 2: cbitDigit = 1; break;

case 4: cbitDigit = 2; break;

case 8: cbitDigit = 3; break;

case 16: cbitDigit = 4; break;

case 32: cbitDigit = 5; break;

default: cbitDigit = 0; break;

}

// REVIEW : fix this to do more accurate conversions? This is exact for

// powers of 2, but not for other radixes.

// REVIEW : round?

wExp2 = (int)((Js::NumberUtilities::LuHiDbl(dbl) & 0x7FF00000) >> 20) - 0x03FF;

maxOutDigits = g\_rgcchSig[radix];

\_\_analysis\_assume(maxOutDigits > 0);

if (wExp2 < -60 || wExp2 > 60)

{

// Use exponential notation. Get the exponent and normalize.

if (cbitDigit != 0)

{

// Power of 2. These computations are exact.

wExp = wExp2 / cbitDigit;

wExp2 = wExp \* cbitDigit;

// Avoid overflow and underflow.

if (wExp2 > 0)

{

wExp2 -= cbitDigit;

dbl /= radix;

}

else

{

wExp2 += cbitDigit;

dbl \*= radix;

}

Js::NumberUtilities::LuHiDbl(valueT) = (ulong)(0x03FF + wExp2) << 20;

Js::NumberUtilities::LuLoDbl(valueT) = 0;

}

else

{

wExp = (int)floor(log(dbl) / log((double)radix) + 1.0);

valueT = pow((double)radix, wExp);

if (!Js::NumberUtilities::IsFinite(valueT))

{

valueT = pow((double)radix, --wExp);

}

else if (0 == valueT)

{

valueT = pow((double)radix, ++wExp);

}

}

dbl = dbl / valueT;

while (dbl < 1)

{

dbl \*= radix;

wExp--;

}

AssertMsg(1 <= dbl && dbl < radix, "malformed computation in radix toString()");

// First digit.

wDig = (int)dbl;

if (len < 2)

{

return FALSE; //We run out of buffer size.

}

len--;

\*ppsz++ = ToDigit(wDig);

maxOutDigits--;

dbl -= wDig;

// Radix point and remaining digits.

if (0 != dbl)

{

if (len < maxOutDigits + 2)

{

return FALSE; //We run out of buffer size.

}

len -= maxOutDigits + 1;

\*ppsz++ = '.';

while (dbl != 0 && maxOutDigits-- > 0)

{

dbl \*= radix;

wDig = (int)dbl;

if (wDig >= radix)

{

wDig = radix - 1;

}

\*ppsz++ = ToDigit(wDig);

dbl -= wDig;

}

}

// Exponent.

if (len < 9) // NOTE: may actually need less room

{

return FALSE; //We run out of buffer size.

}

\*ppsz++ = '(';

\*ppsz++ = 'e';

if (wExp < 0)

{

\*ppsz++ = '-';

wExp = -wExp;

}

else

{

\*ppsz++ = '+';

}

if (wExp >= 10)

{

if (wExp >= 100)

{

if (wExp >= 1000)

{

\*ppsz++ = (wchar\_t)('0' + wExp / 1000);

wExp %= 1000;

}

\*ppsz++ = (wchar\_t)('0' + wExp / 100);

wExp %= 100;

}

\*ppsz++ = (wchar\_t)('0' + wExp / 10);

wExp %= 10;

}

\*ppsz++ = (wchar\_t)('0' + wExp);

\*ppsz++ = ')';

\*ppsz = 0;

return TRUE;

}

// Output the integer portion.

if (1 <= dbl)

{

if (0 != cbitDigit)

{

wExp = wExp2 / cbitDigit;

wExp2 = wExp \* cbitDigit;

Js::NumberUtilities::LuHiDbl(valueDen) = (ulong)(0x03FF + wExp2) << 20;

Js::NumberUtilities::LuLoDbl(valueDen) = 0;

cchSig = abs(wExp) + 1;

}

else

{

cchSig = 1;

for (valueDen = 1; (valueT = valueDen \* radix) <= dbl; valueDen = valueT)

{

cchSig++;

}

}

AssertMsg(valueDen <= dbl && dbl < valueDen \* radix, "Bad floating point format");

\_\_analysis\_assume(cchSig >= 0);

if (len < cchSig + 1)

{

return FALSE; //We run out of buffer size.

}

len -= cchSig;

for (cch = 0; cch < cchSig; cch++)

{

wDig = (int)(dbl / valueDen);

if (wDig >= radix)

{

wDig = radix - 1;

}

\*ppsz++ = ToDigit(wDig);

dbl -= wDig \* valueDen;

valueDen /= radix;

}

}

else

{

if (len < 2)

{

return FALSE; //We run out of buffer size.

}

len--;

\*ppsz++ = '0';

cchSig = 0;

}

// Output the fractional portion.

if (0 != dbl && cchSig < maxOutDigits)

{

// Output the radix point.

if (len < 3)

{

return FALSE; //We run out of buffer size.

}

len--;

\*ppsz++ = '.';

do

{

dbl \*= radix;

wDig = (int)dbl;

if (wDig >= radix)

{

wDig = radix - 1;

}

if (len < 2)

{

return FALSE; //We run out of buffer size.

}

len--;

\*ppsz++ = ToDigit(wDig);

dbl -= wDig;

if (0 != wDig || 0 != cchSig)

{

cchSig++;

}

} while (0 != dbl && cchSig < maxOutDigits);

}

if (len < 1)

{

return FALSE; //We run out of buffer size.

}

\*ppsz = 0;

return TRUE;

}

static const int64 ci64\_2to64 = 0x43F0000000000000;

static const double cdbl\_2to64 = \*(double\*)&ci64\_2to64;

double Js::NumberUtilities::DblFromDecimal(DECIMAL \* pdecIn)

{

double dblRet;

Assert(pdecIn->scale >= 0 && pdecIn->scale < 29);

\_\_analysis\_assume(pdecIn->scale >= 0 && pdecIn->scale < 29);

if ((LONG)pdecIn->Mid32 < 0)

{

dblRet = (cdbl\_2to64 + (double)(LONGLONG)pdecIn->Lo64 +

(double)pdecIn->Hi32 \* cdbl\_2to64) / g\_rgdblTens[pdecIn->scale];

}

else

{

dblRet = ((double)(LONGLONG)pdecIn->Lo64 +

(double)pdecIn->Hi32 \* cdbl\_2to64) / g\_rgdblTens[pdecIn->scale];

}

if (pdecIn->sign != 0)

dblRet = -dblRet;

return dblRet;

}

void Js::NumberUtilities::CodePointAsSurrogatePair(codepoint\_t codePointValue, \_\_out wchar\_t\* first, \_\_out wchar\_t\* second)

{

AssertMsg(first != nullptr && second != nullptr, "Null ptr's passed in for out.");

AssertMsg(IsInSupplementaryPlane(codePointValue), "Code point is not a surrogate pair.");

codePointValue -= 0x10000;

\*first = (wchar\_t)(codePointValue >> 10) + 0xD800;

\*second = (wchar\_t)(codePointValue & 0x3FF /\* This is same as cpv % 0x400 \*/) + 0xDC00;

}

codepoint\_t Js::NumberUtilities::SurrogatePairAsCodePoint(codepoint\_t first, codepoint\_t second)

{

AssertMsg(IsSurrogateLowerPart(first) && IsSurrogateUpperPart(second), "Characters don't form a surrogate pair.");

return ((first - 0xD800) << 10) + (second - 0xDC00) + 0x10000;

}

bool Js::NumberUtilities::IsSurrogateUpperPart(codepoint\_t codePointValue)

{

return codePointValue >= 0xDC00 && codePointValue <= 0xDFFF;

}

bool Js::NumberUtilities::IsSurrogateLowerPart(codepoint\_t codePointValue)

{

return codePointValue >= 0xD800 && codePointValue <= 0xDBFF;

}

bool Js::NumberUtilities::IsInSupplementaryPlane(codepoint\_t codePointValue)

{

Assert(codePointValue <= 0x10FFFF);

return codePointValue >= 0x10000;

}

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include "RejitReason.h"

const char \*const RejitReasonNames[] =

{

#define REJIT\_REASON(n) "" STRINGIZE(n) "",

#include "RejitReasons.h"

#undef REJIT\_REASON

};

const uint NumRejitReasons = \_countof(RejitReasonNames);

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

BEGIN\_ENUM\_BYTE(RejitReason)

#define REJIT\_REASON(n) n,

#include "RejitReasons.h"

#undef REJIT\_REASON

END\_ENUM\_BYTE();

extern const char \*const RejitReasonNames[];

extern const uint NumRejitReasons;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#if !defined(REJIT\_REASON)

#error REJIT\_REASON must be defined before including this file.

#endif

REJIT\_REASON(None)

REJIT\_REASON(Forced)

REJIT\_REASON(RuntimeStatsEnabled)

REJIT\_REASON(AggressiveIntTypeSpecDisabled)

REJIT\_REASON(AggressiveMulIntTypeSpecDisabled)

REJIT\_REASON(DivIntTypeSpecDisabled)

REJIT\_REASON(TrackIntOverflowDisabled)

REJIT\_REASON(FloatTypeSpecDisabled)

REJIT\_REASON(ImplicitCallFlagsChanged)

REJIT\_REASON(FailedPolymorphicInlineeTypeCheck)

REJIT\_REASON(InlineeChanged)

REJIT\_REASON(CheckThisDisabled)

REJIT\_REASON(LossyIntTypeSpecDisabled)

REJIT\_REASON(MemOpDisabled)

REJIT\_REASON(FailedTypeCheck)

REJIT\_REASON(FailedFixedFieldTypeCheck)

REJIT\_REASON(FailedFixedFieldCheck)

REJIT\_REASON(FailedEquivalentTypeCheck)

REJIT\_REASON(FailedEquivalentFixedFieldTypeCheck)

REJIT\_REASON(CtorGuardInvalidated)

REJIT\_REASON(ArrayCheckHoistDisabled)

REJIT\_REASON(ArrayMissingValueCheckHoistDisabled)

REJIT\_REASON(ExpectingNativeArray)

REJIT\_REASON(ConvertedNativeArray)

REJIT\_REASON(ArrayAccessNeededHelperCall)

REJIT\_REASON(JsArraySegmentHoistDisabled)

REJIT\_REASON(TypedArrayTypeSpecDisabled)

REJIT\_REASON(ExpectingConventionalNativeArrayAccess)

REJIT\_REASON(LdLenIntSpecDisabled)

REJIT\_REASON(FailedTagCheck)

REJIT\_REASON(BoundCheckHoistDisabled)

REJIT\_REASON(LoopCountBasedBoundCheckHoistDisabled)

REJIT\_REASON(AfterLoopBodyRejit)

REJIT\_REASON(DisableSwitchOptExpectingInteger)

REJIT\_REASON(DisableSwitchOptExpectingString)

REJIT\_REASON(InlineApplyDisabled)

REJIT\_REASON(InlineSpreadDisabled)

REJIT\_REASON(FloorInliningDisabled)

REJIT\_REASON(ModByPowerOf2)

REJIT\_REASON(NoProfile)

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include <float.h>

#include "Common\SmartFPUControl.h"

//

// Floating point unit utility functions

//

static errno\_t SetFPUControlDefault(void)

{

#if \_M\_AMD64 || \_M\_ARM

return \_controlfp\_s(0, \_RC\_NEAR + \_DN\_SAVE + \_EM\_INVALID + \_EM\_ZERODIVIDE +

\_EM\_OVERFLOW + \_EM\_UNDERFLOW + \_EM\_INEXACT,

\_MCW\_EM | \_MCW\_DN | \_MCW\_RC);

#elif \_M\_IX86

\_control87(\_CW\_DEFAULT, \_MCW\_EM | \_MCW\_DN | \_MCW\_PC | \_MCW\_RC | \_MCW\_IC);

return 0;

#else

return \_controlfp\_s(0, \_CW\_DEFAULT, \_MCW\_EM | \_MCW\_DN | \_MCW\_PC | \_MCW\_RC | \_MCW\_IC);

#endif

}

static errno\_t GetFPUControl(unsigned int \*pctrl)

{

Assert(pctrl != nullptr);

#if \_M\_IX86

\*pctrl = \_control87(0, 0);

return 0;

#else

return \_controlfp\_s(pctrl, 0, 0);

#endif

}

static errno\_t SetFPUControl(unsigned int fpctrl)

{

#if \_M\_AMD64 || \_M\_ARM

return \_controlfp\_s(0, fpctrl, \_MCW\_EM | \_MCW\_DN | \_MCW\_RC);

#elif \_M\_IX86

\_control87(fpctrl, (unsigned int)(-1));

return 0;

#else

return \_controlfp\_s(0, fpctrl, (unsigned int)(-1));

#endif

}

static void ClearFPUStatus(void)

{

// WinSE 187789

// \_clearfp gives up the thread's time slice, so clear only if flags are set

if (\_statusfp())

\_clearfp();

}

template <bool enabled>

SmartFPUControlT<enabled>::SmartFPUControlT()

{

if (enabled)

{

m\_oldFpuControl = INVALID\_FPUCONTROL;

ClearFPUStatus(); // Clear pending exception status first (blue 555235)

m\_err = GetFPUControl(&m\_oldFpuControl);

if (m\_err == 0)

{

m\_err = SetFPUControlDefault();

}

}

#if DBG

else

{

m\_oldFpuControl = INVALID\_FPUCONTROL;

m\_err = GetFPUControl(&m\_oldFpuControl);

m\_oldFpuControlForConsistencyCheck = m\_oldFpuControl;

}

#endif

}

template <bool enabled>

SmartFPUControlT<enabled>::~SmartFPUControlT()

{

if (enabled)

{

RestoreFPUControl();

}

#if DBG

else

{

uint currentFpuControl;

m\_err = GetFPUControl(&currentFpuControl);

if (m\_err == 0 && m\_oldFpuControlForConsistencyCheck != INVALID\_FPUCONTROL)

{

Assert(m\_oldFpuControlForConsistencyCheck == currentFpuControl);

}

}

#endif

}

template <bool enabled>

void

SmartFPUControlT<enabled>::RestoreFPUControl()

{

if (enabled)

{

if (m\_oldFpuControl != INVALID\_FPUCONTROL)

{

m\_err = SetFPUControl(m\_oldFpuControl);

m\_oldFpuControl = INVALID\_FPUCONTROL; // Only restore once

}

}

else

{

// Shouldn't restore if this is not enabled

Assert(false);

}

}

// Explicit instantiation

template class SmartFPUControlT<true>;

template class SmartFPUControlT<false>;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// This class saves the FPU control word, and sets it to the default value.

// The default value will prevent all floating point exceptions other than

// those because of denormal operand from being generated.

// When the instance goes out of scope, the control word will be restored to the original value.

template <bool enabled>

class SmartFPUControlT

{

static const uint INVALID\_FPUCONTROL = (uint)-1;

public:

SmartFPUControlT();

~SmartFPUControlT();

bool HasErr() const

{

return m\_err != 0;

}

HRESULT GetErr() const

{

Assert(HasErr());

return HRESULT\_FROM\_WIN32(m\_err);

}

void RestoreFPUControl();

private:

#if DBG

uint m\_oldFpuControlForConsistencyCheck = INVALID\_FPUCONTROL;

#endif

uint m\_oldFpuControl;

errno\_t m\_err;

};

// Set and Restore FPU control value by default

typedef SmartFPUControlT<true> SmartFPUControl;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// Note, these definitions must match the definitions in jsrt.h.

// Callback for a background work item.

namespace JsUtil

{

class ThreadService

{

public:

typedef void (CALLBACK \*BackgroundWorkItemCallback)(void \* callbackData);

// Callback for the thread service itself, to which we can submit background work items.

typedef bool (CALLBACK \*ThreadServiceCallback)(BackgroundWorkItemCallback callback, void \* callbackData);

private:

ThreadServiceCallback threadService;

bool isInCallback;

public:

ThreadService(ThreadServiceCallback threadService) :

threadService(threadService),

isInCallback(false)

{

}

bool Invoke(BackgroundWorkItemCallback callback, void \* callbackData)

{

isInCallback = true;

bool result = threadService(callback, callbackData);

isInCallback = false;

return result;

}

bool HasCallback() const

{

return this != nullptr && threadService != nullptr;

}

bool IsInCallback() const

{

return isInCallback;

}

};

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include "Common\Tick.h"

namespace Js {

uint64 Tick::s\_luFreq;

uint64 Tick::s\_luBegin;

#if DBG

uint64 Tick::s\_DEBUG\_luStart = 0;

uint64 Tick::s\_DEBUG\_luSkip = 0;

#endif

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// struct Tick

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// Tick::Tick

///

/// Tick() initializes a new Tick instance to an "empty" time. This instance

/// must be assigned to another Tick instance or Now() to have value.

///

///----------------------------------------------------------------------------

Tick::Tick()

{

m\_luTick = 0;

}

///----------------------------------------------------------------------------

///

/// Tick::Tick

///

/// Tick() initializes a new Tick instance to a specific time, in native

/// time units.

///

///----------------------------------------------------------------------------

Tick::Tick(

uint64 luTick) // Tick, in internal units

{

m\_luTick = luTick;

}

///----------------------------------------------------------------------------

///

/// Tick::FromMicroseconds

///

/// FromMicroseconds() returns a Tick instance from a given time in

/// microseconds.

///

///----------------------------------------------------------------------------

Tick

Tick::FromMicroseconds(

uint64 luTime) // Time, in microseconds

{

//

// Ensure we can convert losslessly.

//

#if DBG

const uint64 luMaxTick = \_UI64\_MAX / s\_luFreq;

AssertMsg(luTime <= luMaxTick, "Ensure time can be converted losslessly");

#endif // DBG

//

// Create the Tick

//

uint64 luTick = luTime \* s\_luFreq / ((uint64) 1000000);

return Tick(luTick);

}

///----------------------------------------------------------------------------

///

/// Tick::FromQPC

///

/// FromQPC() returns a Tick instance from a given QPC time.

///

///----------------------------------------------------------------------------

Tick

Tick::FromQPC(

uint64 luTime) // Time, in QPC units

{

return Tick(luTime - s\_luBegin);

}

///----------------------------------------------------------------------------

///

/// Tick::ToQPC

///

/// ToQPC() returns the QPC time for this time instance

///

///----------------------------------------------------------------------------

uint64

Tick::ToQPC()

{

return (m\_luTick + s\_luBegin);

}

///----------------------------------------------------------------------------

///

/// Tick::operator +

///

/// operator +()

///

///----------------------------------------------------------------------------

Tick

Tick::operator +(

TickDelta tdChange // RHS TickDelta

) const

{

return Tick(m\_luTick + tdChange.m\_lnDelta);

}

///----------------------------------------------------------------------------

///

/// Tick::operator -

///

/// operator -()

///

///----------------------------------------------------------------------------

Tick

Tick::operator -(

TickDelta tdChange // RHS TickDelta

) const

{

return Tick(m\_luTick - tdChange.m\_lnDelta);

}

///----------------------------------------------------------------------------

///

/// Tick::operator -

///

/// operator -()

///

///----------------------------------------------------------------------------

TickDelta

Tick::operator -(

Tick timeOther // RHS Tick

) const

{

return TickDelta(m\_luTick - timeOther.m\_luTick);

}

///----------------------------------------------------------------------------

///

/// Tick::operator ==

///

/// operator ==()

///

///----------------------------------------------------------------------------

bool

Tick::operator ==(

Tick timeOther // RHS Tick

) const

{

return m\_luTick == timeOther.m\_luTick;

}

///----------------------------------------------------------------------------

///

/// Tick::operator !=

///

/// operator !=()

///

///----------------------------------------------------------------------------

bool

Tick::operator !=(

Tick timeOther // RHS Tick

) const

{

return m\_luTick != timeOther.m\_luTick;

}

///----------------------------------------------------------------------------

///

/// Tick::operator <

///

/// operator <()

///

///----------------------------------------------------------------------------

bool

Tick::operator <(

Tick timeOther // RHS Tick

) const

{

return m\_luTick < timeOther.m\_luTick;

}

///----------------------------------------------------------------------------

///

/// Tick::operator <=

///

/// operator <=()

///

///----------------------------------------------------------------------------

bool

Tick::operator <=(

Tick timeOther // RHS Tick

) const

{

return m\_luTick <= timeOther.m\_luTick;

}

///----------------------------------------------------------------------------

///

/// Tick::operator >

///

/// operator >()

///

///----------------------------------------------------------------------------

bool

Tick::operator >(

Tick timeOther // RHS Tick

) const

{

return m\_luTick > timeOther.m\_luTick;

}

///----------------------------------------------------------------------------

///

/// Tick::operator >=

///

/// operator >=()

///

///----------------------------------------------------------------------------

bool

Tick::operator >=(

Tick timeOther // RHS Tick

) const

{

return m\_luTick >= timeOther.m\_luTick;

}

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// struct TickDelta

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// TickDelta::TickDelta

///

/// TickDelta() initializes a new TickDelta instance to "zero" delta.

///

///----------------------------------------------------------------------------

TickDelta::TickDelta()

{

m\_lnDelta = 0;

}

///----------------------------------------------------------------------------

///

/// TickDelta::TickDelta

///

/// TickDelta() initializes a new TickDelta instance to a specific time delta,

/// in native time units.

///

///----------------------------------------------------------------------------

TickDelta::TickDelta(

int64 lnDelta)

{

m\_lnDelta = lnDelta;

}

///----------------------------------------------------------------------------

///

/// TickDelta::ToMicroseconds

///

/// ToMicroseconds() returns the time delta, in microseconds. The time is

/// rounded to the nearest available whole units.

///

///----------------------------------------------------------------------------

int64

TickDelta::ToMicroseconds() const

{

if (\*this == Infinite())

{

return \_I64\_MAX;

}

//

// Ensure we can convert losslessly.

//

const int64 lnMinTimeDelta = \_I64\_MIN / ((int64) 1000000);

const int64 lnMaxTimeDelta = \_I64\_MAX / ((int64) 1000000);

AssertMsg((m\_lnDelta <= lnMaxTimeDelta) && (m\_lnDelta >= lnMinTimeDelta),

"Ensure delta can be converted to microseconds losslessly");

//

// Compute the microseconds.

//

int64 lnFreq = (int64) Tick::s\_luFreq;

int64 lnTickDelta = (m\_lnDelta \* ((int64) 1000000)) / lnFreq;

return lnTickDelta;

}

///----------------------------------------------------------------------------

///

/// TickDelta::FromMicroseconds

///

/// FromMicroseconds() returns a TickDelta instance from a given delta in

/// microseconds.

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::FromMicroseconds(

int64 lnTimeDelta) // Time delta, in 1/1000^2 sec

{

AssertMsg(lnTimeDelta != \_I64\_MAX, "Use Infinite() to create an infinite TickDelta");

//

// Ensure that we can convert losslessly.

//

int64 lnFreq = (int64) Tick::s\_luFreq;

#if DBG

const int64 lnMinTimeDelta = \_I64\_MIN / lnFreq;

const int64 lnMaxTimeDelta = \_I64\_MAX / lnFreq;

AssertMsg((lnTimeDelta <= lnMaxTimeDelta) && (lnTimeDelta >= lnMinTimeDelta),

"Ensure delta can be converted to native format losslessly");

#endif // DBG

//

// Create the TickDelta

//

int64 lnTickDelta = (lnTimeDelta \* lnFreq) / ((int64) 1000000);

TickDelta td(lnTickDelta);

AssertMsg(td != Infinite(), "Can not create infinite TickDelta");

return td;

}

///----------------------------------------------------------------------------

///

/// TickDelta::FromMicroseconds

///

/// FromMicroseconds() returns a TickDelta instance from a given delta in

/// microseconds.

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::FromMicroseconds(

int nTimeDelta) // Tick delta, in 1/1000^2 sec

{

AssertMsg(nTimeDelta != \_I32\_MAX, "Use Infinite() to create an infinite TickDelta");

return FromMicroseconds((int64) nTimeDelta);

}

///----------------------------------------------------------------------------

///

/// TickDelta::FromMilliseconds

///

/// FromMilliseconds() returns a TickDelta instance from a given delta in

/// milliseconds.

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::FromMilliseconds(

int nTimeDelta) // Tick delta, in 1/1000^1 sec

{

AssertMsg(nTimeDelta != \_I32\_MAX, "Use Infinite() to create an infinite TickDelta");

return FromMicroseconds(((int64) nTimeDelta) \* ((int64) 1000));

}

///----------------------------------------------------------------------------

///

/// TickDelta::Infinite

///

/// Infinite() returns a time-delta infinitely far away.

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::Infinite()

{

return TickDelta(\_I64\_MAX);

}

///----------------------------------------------------------------------------

///

/// TickDelta::IsForward

///

/// IsForward() returns whether adding this TickDelta to a given Tick will

/// not move the time backwards.

///

///----------------------------------------------------------------------------

bool

TickDelta::IsForward() const

{

return m\_lnDelta >= 0;

}

///----------------------------------------------------------------------------

///

/// TickDelta::IsBackward

///

/// IsBackward() returns whether adding this TickDelta to a given Tick will

/// not move the time forwards.

///

///----------------------------------------------------------------------------

bool

TickDelta::IsBackward() const

{

return m\_lnDelta <= 0;

}

///----------------------------------------------------------------------------

///

/// TickDelta::Abs

///

/// Abs() returns the absolute value of the TickDelta.

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::Abs(TickDelta tdOther)

{

return TickDelta(tdOther.m\_lnDelta < 0 ? -tdOther.m\_lnDelta : tdOther.m\_lnDelta);

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator %

///

/// operator %()

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::operator %(

TickDelta tdOther // RHS TickDelta

) const

{

return TickDelta(m\_lnDelta % tdOther.m\_lnDelta);

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator \

///

/// operator \() - Divides one TickDelta by another, in TickDelta units

///

///----------------------------------------------------------------------------

int64

TickDelta::operator /(

TickDelta tdOther // RHS TickDelta

) const

{

return m\_lnDelta / tdOther.m\_lnDelta;

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator +

///

/// operator +()

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::operator +(

TickDelta tdOther // RHS TickDelta

) const

{

AssertMsg((\*this != Infinite()) && (tdOther != Infinite()),

"Can not combine infinite TickDeltas");

return TickDelta(m\_lnDelta + tdOther.m\_lnDelta);

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator -

///

/// operator -()

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::operator -(

TickDelta tdOther // RHS TickDelta

) const

{

AssertMsg((\*this != Infinite()) && (tdOther != Infinite()),

"Can not combine infinite TickDeltas");

return TickDelta(m\_lnDelta - tdOther.m\_lnDelta);

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator \*

///

/// operator \*()

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::operator \*(

int nScale // RHS scale

) const

{

AssertMsg(\*this != Infinite(), "Can not combine infinite TickDeltas");

return TickDelta(m\_lnDelta \* nScale);

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator \*

///

/// operator \*()

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::operator \*(

float flScale // RHS scale

) const

{

AssertMsg(\*this != Infinite(), "Can not combine infinite TickDeltas");

return TickDelta((int64) (((double) m\_lnDelta) \* ((double) flScale)));

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator /

///

/// operator /()

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::operator /(

int nScale // RHS scale

) const

{

AssertMsg(\*this != Infinite(), "Can not combine infinite TickDeltas");

AssertMsg(nScale != 0, "Can not scale by 0");

return TickDelta(m\_lnDelta / nScale);

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator /

///

/// operator /()

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::operator /(

float flScale // RHS scale

) const

{

AssertMsg(\*this != Infinite(), "Can not combine infinite TickDeltas");

AssertMsg(flScale != 0, "Can not scale by 0");

return TickDelta((int64) (((double) m\_lnDelta) / ((double) flScale)));

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator +=

///

/// operator +=()

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::operator +=(

TickDelta tdOther) // RHS TickDelta

{

AssertMsg((\*this != Infinite()) && (tdOther != Infinite()),

"Can not combine infinite TickDeltas");

m\_lnDelta = m\_lnDelta + tdOther.m\_lnDelta;

return \*this;

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator -=

///

/// operator -=()

///

///----------------------------------------------------------------------------

TickDelta

TickDelta::operator -=(

TickDelta tdOther) // RHS TickDelta

{

AssertMsg((\*this != Infinite()) && (tdOther != Infinite()),

"Can not combine infinite TickDeltas");

m\_lnDelta = m\_lnDelta - tdOther.m\_lnDelta;

return \*this;

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator ==

///

/// operator ==()

///

///----------------------------------------------------------------------------

bool

TickDelta::operator ==(

TickDelta tdOther // RHS TickDelta

) const

{

return m\_lnDelta == tdOther.m\_lnDelta;

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator !=

///

/// operator !=()

///

///----------------------------------------------------------------------------

bool

TickDelta::operator !=(

TickDelta tdOther // RHS TickDelta

) const

{

return m\_lnDelta != tdOther.m\_lnDelta;

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator <

///

/// operator <()

///

///----------------------------------------------------------------------------

bool

TickDelta::operator <(

TickDelta tdOther // RHS TickDelta

) const

{

return m\_lnDelta < tdOther.m\_lnDelta;

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator <=

///

/// operator <=()

///

///----------------------------------------------------------------------------

bool

TickDelta::operator <=(

TickDelta tdOther // RHS TickDelta

) const

{

return m\_lnDelta <= tdOther.m\_lnDelta;

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator >

///

/// operator >()

///

///----------------------------------------------------------------------------

bool

TickDelta::operator >(

TickDelta tdOther // RHS TickDelta

) const

{

return m\_lnDelta > tdOther.m\_lnDelta;

}

///----------------------------------------------------------------------------

///

/// TickDelta::operator >=

///

/// operator >=()

///

///----------------------------------------------------------------------------

bool

TickDelta::operator >=(

TickDelta tdOther // RHS TickDelta

) const

{

return m\_lnDelta >= tdOther.m\_lnDelta;

}

void Tick::InitType()

{

/\* CheckWin32( \*/ QueryPerformanceFrequency((LARGE\_INTEGER \*) &s\_luFreq);

/\* CheckWin32( \*/ QueryPerformanceCounter((LARGE\_INTEGER \*) &s\_luBegin);

#if DBG

s\_luBegin += s\_DEBUG\_luStart;

#endif

//

// Ensure that we have a sufficient amount of time so that we can handle useful time operations.

//

uint64 nSec = \_UI64\_MAX / s\_luFreq;

if (nSec < 5 \* 60)

{

#if FIXTHIS

PromptInvalid("QueryPerformanceFrequency() will not provide at least 5 minutes");

return Results::GenericFailure;

#endif

}

}

Tick Tick::Now()

{

// Determine our current time

uint64 luCurrent = s\_luBegin;

/\* Verify( \*/ QueryPerformanceCounter((LARGE\_INTEGER \*) &luCurrent);

#if DBG

luCurrent += s\_DEBUG\_luStart + s\_DEBUG\_luSkip;

#endif

// Create a Tick instance, using our delta since we started tracking time.

uint64 luDelta = luCurrent - s\_luBegin;

return Tick(luDelta);

}

uint64 Tick::ToMicroseconds() const

{

//

// Convert time in microseconds (1 / 1000^2). Because of the finite precision and wrap-around,

// this math depends on where the Tick is.

//

const uint64 luOneSecUs = (uint64) 1000000;

const uint64 luSafeTick = \_UI64\_MAX / luOneSecUs;

if (m\_luTick < luSafeTick)

{

//

// Small enough to convert directly into microseconds.

//

uint64 luTick = (m\_luTick \* luOneSecUs) / s\_luFreq;

return luTick;

}

else

{

//

// Number is too large, so we need to do this is stages.

// 1. Compute the number of seconds

// 2. Convert the remainder

// 3. Add the two parts together

//

uint64 luSec = m\_luTick / s\_luFreq;

uint64 luRemain = m\_luTick - luSec \* s\_luFreq;

uint64 luTick = (luRemain \* luOneSecUs) / s\_luFreq;

luTick += luSec \* luOneSecUs;

return luTick;

}

}

int TickDelta::ToMilliseconds() const

{

if (\*this == Infinite())

{

return \_I32\_MAX;

}

int64 nTickUs = ToMicroseconds();

int64 lnRound = 500;

if (nTickUs < 0)

{

lnRound = -500;

}

int64 lnDelta = (nTickUs + lnRound) / ((int64) 1000);

AssertMsg((lnDelta <= INT\_MAX) && (lnDelta >= INT\_MIN), "Ensure no overflow");

return (int) lnDelta;

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

struct TickDelta;

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// struct Tick

///

/// Tick represents an abstract point in time in some internal units. This

/// class is designed to be very fast and very accurate to use, internally

/// utilizing the system's "high-fidelity" time mechanism. Tick always

/// "moves forward", meaning that time values continually increase.

///

/// This class is carefully designed to use two's complement and wrap-around

/// overflow math. The entire storage needs to be used, and the internal time

/// format can not be converted to a different unit, or there will be problems

/// when the system time wraps around. Since we are using

/// QueryPerformanceCounter(), this is variable, but it can happen every 2.5

/// hours. This also means that we should never have TickDeltas / arithmetic

/// that is for any great period of time.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

struct Tick

{

// Construction

public:

static void InitType();

Tick();

private:

Tick(uint64 luTick);

// Properties

public:

uint64 ToMicroseconds() const;

static Tick FromMicroseconds(uint64 luTick);

static Tick FromQPC(uint64 luQPCTick);

static Tick Now();

Tick operator +(TickDelta tdChange) const;

Tick operator -(TickDelta tdChange) const;

TickDelta operator -(Tick timeOther) const;

bool operator ==(Tick timeOther) const;

bool operator !=(Tick timeOther) const;

bool operator <(Tick timeOther) const;

bool operator <=(Tick timeOther) const;

bool operator >(Tick timeOther) const;

bool operator >=(Tick timeOther) const;

uint64 ToQPC();

// Data

private:

static uint64 s\_luFreq; // Frequency

static uint64 s\_luBegin; // Beginning time

#if DBG

static uint64 s\_DEBUG\_luStart; // Tick start offset for debugging

static uint64 s\_DEBUG\_luSkip; // Tick skip offset for debugging

#endif

uint64 m\_luTick; // Current time sample

friend TickDelta;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* struct TickDelta

\*

\* TickDelta represents the measured period between two Ticks. TickDelta values

\* may be combined with each other, and then added to Ticks to create new Ticks.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct TickDelta

{

// Construction

public:

TickDelta();

private:

TickDelta(int64 lnDelta);

// Properties

public:

int64 ToMicroseconds() const;

int ToMilliseconds() const;

static TickDelta FromMicroseconds(int nTickDelta);

static TickDelta FromMicroseconds(int64 lnTickDelta);

static TickDelta FromMilliseconds(int nTickDelta);

bool IsForward() const;

bool IsBackward() const;

static TickDelta Infinite();

static TickDelta Abs(TickDelta tdOther);

TickDelta operator +(TickDelta tdOther) const;

TickDelta operator -(TickDelta tdOther) const;

TickDelta operator %(TickDelta tdOther) const;

int64 operator /(TickDelta tdOther) const;

TickDelta operator \*(int nScale) const;

TickDelta operator \*(float flScale) const;

TickDelta operator /(int nScale) const;

TickDelta operator /(float flScale) const;

TickDelta operator +=(TickDelta tdOther);

TickDelta operator -=(TickDelta tdOther);

bool operator ==(TickDelta tdOther) const;

bool operator !=(TickDelta tdOther) const;

bool operator <(TickDelta tdOther) const;

bool operator <=(TickDelta tdOther) const;

bool operator >(TickDelta tdOther) const;

bool operator >=(TickDelta tdOther) const;

// Data

private:

int64 m\_lnDelta; // Tick delta

friend Tick;

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

class UInt16Math

{

public:

template< class Func >

static uint16 Add(uint16 lhs, uint16 rhs, \_\_inout Func& overflowFn)

{

uint16 result = lhs + rhs;

// If the result is smaller than the LHS, then we overflowed

if( result < lhs )

{

overflowFn();

}

return result;

}

template< class Func >

static void Inc(uint16& lhs, \_\_inout Func& overflowFn)

{

++lhs;

// If lhs becomes 0, then we overflowed

if(!lhs)

{

overflowFn();

}

}

// Convenience function which uses DefaultOverflowPolicy (throws OOM upon overflow)

static uint16 Add(uint16 lhs, uint16 rhs)

{

return Add(lhs, rhs, ::Math::DefaultOverflowPolicy);

}

// Convenience function which returns a bool indicating overflow

static bool Add(uint16 lhs, uint16 rhs, \_\_out uint16\* result)

{

::Math::RecordOverflowPolicy overflowGuard;

\*result = Add(lhs, rhs, overflowGuard);

return overflowGuard.HasOverflowed();

}

// Convenience function which uses DefaultOverflowPolicy (throws OOM upon overflow)

static void Inc(uint16& lhs)

{

Inc(lhs, ::Math::DefaultOverflowPolicy);

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

class UInt32Math

{

public:

template< class Func >

static uint32 Add(uint32 lhs, uint32 rhs, \_\_inout Func& overflowFn)

{

uint32 result = lhs + rhs;

// If the result is smaller than the LHS, then we overflowed

if( result < lhs )

{

overflowFn();

}

return result;

}

template< class Func >

static void Inc(uint32& lhs, \_\_inout Func& overflowFn)

{

++lhs;

// If lhs becomes 0, then we overflowed

if(!lhs)

{

overflowFn();

}

}

template< class Func >

static uint32 Mul(uint32 lhs, uint32 rhs, \_\_inout Func& overflowFn)

{

// Do the multiplication using 64-bit unsigned math.

uint64 result = static\_cast<uint64>(lhs) \* static\_cast<uint64>(rhs);

// Does the result fit in 32-bits?

if(result >= (1ui64 << 32))

{

overflowFn();

}

return static\_cast<uint32>(result);

}

template<uint32 mul, class Func >

static uint32 Mul(uint32 left, \_\_inout Func& overflowFn)

{

CompileAssert(mul != 0);

if (left > (UINT\_MAX / mul))

{

overflowFn();

}

// If mul is a power of 2, the compiler will convert this to a shift left.

return left \* mul;

}

// Using 0 for mul will result in compile-time divide by zero error (which is desired behavior)

template< uint32 add, uint32 mul, class Func >

static uint32 AddMul(uint32 left, \_\_inout Func& overflowFn)

{

//

// The result will overflow if (left+add)\*mul > UINT\_MAX

// Rearranging, this becomes: left > (UINT\_MAX / mul ) - add

//

// If mul and add are compile-time constants then LTCG will collapse

// this to a simple constant comparison.

//

CompileAssert(UINT\_MAX/mul >= add);

if( left > ((UINT\_MAX / mul) - add) )

{

overflowFn();

}

// When add and mul are small constants, the compiler is

// typically able to use the LEA instruction here.

return (left + add) \* mul;

}

// Using 0 for mul will result in compile-time divide by zero error (which is desired behavior)

template< uint32 mul, uint32 add, class Func >

static uint32 MulAdd(uint32 left, \_\_inout Func& overflowFn)

{

//

// The result will overflow if (left\*mul)+add > UINT\_MAX

// Rearranging, this becomes: left > (UINT\_MAX - add) / mul

//

// If add and mul are compile-time constants then LTCG will collapse

// this to a simple constant comparison.

//

if( left > ((UINT\_MAX - add) / mul) )

{

overflowFn();

}

// When add and mul are small constants, the compiler is

// typically able to use the LEA instruction here.

return (left \* mul) + add;

}

// Convenience functions which use the DefaultOverflowPolicy (throw OOM upon overflow)

template< uint32 add, uint32 mul >

static uint32 AddMul(uint32 left)

{

return AddMul<add,mul>(left, ::Math::DefaultOverflowPolicy);

}

static uint32 Add(uint32 lhs, uint32 rhs)

{

return Add( lhs, rhs, ::Math::DefaultOverflowPolicy );

}

static uint32 Mul(uint32 lhs, uint32 rhs)

{

return Mul(lhs, rhs, ::Math::DefaultOverflowPolicy );

}

template<uint32 mul>

static uint32 Mul(uint32 lhs)

{

return Mul<mul>(lhs, ::Math::DefaultOverflowPolicy);

}

template< uint32 mul, uint32 add >

static uint32 MulAdd(uint32 left)

{

return MulAdd<mul,add>(left, ::Math::DefaultOverflowPolicy);

}

// Convenience functions which return a bool indicating overflow

static bool Add(uint32 lhs, uint32 rhs, \_\_out uint32\* result)

{

::Math::RecordOverflowPolicy overflowGuard;

\*result = Add(lhs, rhs, overflowGuard);

return overflowGuard.HasOverflowed();

}

static bool Mul(uint32 lhs, uint32 rhs, \_\_out uint32\* result)

{

::Math::RecordOverflowPolicy overflowGuard;

\*result = Mul(lhs, rhs, overflowGuard);

return overflowGuard.HasOverflowed();

}

// Convenience function which uses DefaultOverflowPolicy (throws OOM upon overflow)

static void Inc(uint32& lhs)

{

Inc(lhs, ::Math::DefaultOverflowPolicy);

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include "Common\unicode.h"

//

// This file contains tables used for operations on the Unicode character

// set, such as changing case.

//

// See "The Unicode Standard, Version 2.0" for complete documentation.

//

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Mapping to uppercase.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static const wchar kchMinUpper1 = 0x0061;

static const wchar kchLimUpper1 = 0x0293;

static const wchar g\_mpchchUpper1[kchLimUpper1 - kchMinUpper1] =

{

0x0041, 0x0042, 0x0043, 0x0044, 0x0045, 0x0046, 0x0047, 0x0048, 0x0049, 0x004a,

0x004b, 0x004c, 0x004d, 0x004e, 0x004f, 0x0050, 0x0051, 0x0052, 0x0053, 0x0054,

0x0055, 0x0056, 0x0057, 0x0058, 0x0059, 0x005a, 0x007b, 0x007c, 0x007d, 0x007e,

0x007f, 0x0080, 0x0081, 0x0082, 0x0083, 0x0084, 0x0085, 0x0086, 0x0087, 0x0088,

0x0089, 0x008a, 0x008b, 0x008c, 0x008d, 0x008e, 0x008f, 0x0090, 0x0091, 0x0092,

0x0093, 0x0094, 0x0095, 0x0096, 0x0097, 0x0098, 0x0099, 0x009a, 0x009b, 0x009c,

0x009d, 0x009e, 0x009f, 0x00a0, 0x00a1, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6,

0x00a7, 0x00a8, 0x00a9, 0x00aa, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af, 0x00b0,

0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7, 0x00b8, 0x00b9, 0x00ba,

0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf, 0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4,

0x00c5, 0x00c6, 0x00c7, 0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce,

0x00cf, 0x00d0, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7, 0x00d8,

0x00d9, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df, 0x00c0, 0x00c1, 0x00c2,

0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7, 0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc,

0x00cd, 0x00ce, 0x00cf, 0x00d0, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6,

0x00f7, 0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x0178, 0x0100,

0x0100, 0x0102, 0x0102, 0x0104, 0x0104, 0x0106, 0x0106, 0x0108, 0x0108, 0x010a,

0x010a, 0x010c, 0x010c, 0x010e, 0x010e, 0x0110, 0x0110, 0x0112, 0x0112, 0x0114,

0x0114, 0x0116, 0x0116, 0x0118, 0x0118, 0x011a, 0x011a, 0x011c, 0x011c, 0x011e,

0x011e, 0x0120, 0x0120, 0x0122, 0x0122, 0x0124, 0x0124, 0x0126, 0x0126, 0x0128,

0x0128, 0x012a, 0x012a, 0x012c, 0x012c, 0x012e, 0x012e, 0x0130, 0x0049, 0x0132,

0x0132, 0x0134, 0x0134, 0x0136, 0x0136, 0x0138, 0x0139, 0x0139, 0x013b, 0x013b,

0x013d, 0x013d, 0x013f, 0x013f, 0x0141, 0x0141, 0x0143, 0x0143, 0x0145, 0x0145,

0x0147, 0x0147, 0x0149, 0x014a, 0x014a, 0x014c, 0x014c, 0x014e, 0x014e, 0x0150,

0x0150, 0x0152, 0x0152, 0x0154, 0x0154, 0x0156, 0x0156, 0x0158, 0x0158, 0x015a,

0x015a, 0x015c, 0x015c, 0x015e, 0x015e, 0x0160, 0x0160, 0x0162, 0x0162, 0x0164,

0x0164, 0x0166, 0x0166, 0x0168, 0x0168, 0x016a, 0x016a, 0x016c, 0x016c, 0x016e,

0x016e, 0x0170, 0x0170, 0x0172, 0x0172, 0x0174, 0x0174, 0x0176, 0x0176, 0x0178,

0x0179, 0x0179, 0x017b, 0x017b, 0x017d, 0x017d, 0x0053, 0x0180, 0x0181, 0x0182,

0x0182, 0x0184, 0x0184, 0x0186, 0x0187, 0x0187, 0x0189, 0x018a, 0x018b, 0x018b,

0x018d, 0x018e, 0x018f, 0x0190, 0x0191, 0x0191, 0x0193, 0x0194, 0x0195, 0x0196,

0x0197, 0x0198, 0x0198, 0x019a, 0x019b, 0x019c, 0x019d, 0x019e, 0x019f, 0x01a0,

0x01a0, 0x01a2, 0x01a2, 0x01a4, 0x01a4, 0x01a6, 0x01a7, 0x01a7, 0x01a9, 0x01aa,

0x01ab, 0x01ac, 0x01ac, 0x01ae, 0x01af, 0x01af, 0x01b1, 0x01b2, 0x01b3, 0x01b3,

0x01b5, 0x01b5, 0x01b7, 0x01b8, 0x01b8, 0x01ba, 0x01bb, 0x01bc, 0x01bc, 0x01be,

0x01bf, 0x01c0, 0x01c1, 0x01c2, 0x01c3, 0x01c4, 0x01c4, 0x01c4, 0x01c7, 0x01c7,

0x01c7, 0x01ca, 0x01ca, 0x01ca, 0x01cd, 0x01cd, 0x01cf, 0x01cf, 0x01d1, 0x01d1,

0x01d3, 0x01d3, 0x01d5, 0x01d5, 0x01d7, 0x01d7, 0x01d9, 0x01d9, 0x01db, 0x01db,

0x01dd, 0x01de, 0x01de, 0x01e0, 0x01e0, 0x01e2, 0x01e2, 0x01e4, 0x01e4, 0x01e6,

0x01e6, 0x01e8, 0x01e8, 0x01ea, 0x01ea, 0x01ec, 0x01ec, 0x01ee, 0x01ee, 0x01f0,

0x01f1, 0x01f1, 0x01f1, 0x01f4, 0x01f4, 0x01f6, 0x01f7, 0x01f8, 0x01f9, 0x01fa,

0x01fa, 0x01fc, 0x01fc, 0x01fe, 0x01fe, 0x0200, 0x0200, 0x0202, 0x0202, 0x0204,

0x0204, 0x0206, 0x0206, 0x0208, 0x0208, 0x020a, 0x020a, 0x020c, 0x020c, 0x020e,

0x020e, 0x0210, 0x0210, 0x0212, 0x0212, 0x0214, 0x0214, 0x0216, 0x0216, 0x0218,

0x0219, 0x021a, 0x021b, 0x021c, 0x021d, 0x021e, 0x021f, 0x0220, 0x0221, 0x0222,

0x0223, 0x0224, 0x0225, 0x0226, 0x0227, 0x0228, 0x0229, 0x022a, 0x022b, 0x022c,

0x022d, 0x022e, 0x022f, 0x0230, 0x0231, 0x0232, 0x0233, 0x0234, 0x0235, 0x0236,

0x0237, 0x0238, 0x0239, 0x023a, 0x023b, 0x023c, 0x023d, 0x023e, 0x023f, 0x0240,

0x0241, 0x0242, 0x0243, 0x0244, 0x0245, 0x0246, 0x0247, 0x0248, 0x0249, 0x024a,

0x024b, 0x024c, 0x024d, 0x024e, 0x024f, 0x0250, 0x0251, 0x0252, 0x0181, 0x0186,

0x0255, 0x0189, 0x018a, 0x018e, 0x018f, 0x025a, 0x0190, 0x025c, 0x025d, 0x025e,

0x025f, 0x0193, 0x0261, 0x0262, 0x0194, 0x0264, 0x0265, 0x0266, 0x0267, 0x0197,

0x0196, 0x026a, 0x026b, 0x026c, 0x026d, 0x026e, 0x019c, 0x0270, 0x0271, 0x019d,

0x0273, 0x0274, 0x0275, 0x0276, 0x0277, 0x0278, 0x0279, 0x027a, 0x027b, 0x027c,

0x027d, 0x027e, 0x027f, 0x0280, 0x0281, 0x0282, 0x01a9, 0x0284, 0x0285, 0x0286,

0x0287, 0x01ae, 0x0289, 0x01b1, 0x01b2, 0x028c, 0x028d, 0x028e, 0x028f, 0x0290,

0x0291, 0x01b7

};

static const wchar kchMinUpper2 = 0x03ac;

static const wchar kchLimUpper2 = 0x0587;

static const wchar g\_mpchchUpper2[kchLimUpper2 - kchMinUpper2] =

{

0x0386, 0x0388, 0x0389, 0x038a, 0x03b0, 0x0391, 0x0392, 0x0393, 0x0394, 0x0395,

0x0396, 0x0397, 0x0398, 0x0399, 0x039a, 0x039b, 0x039c, 0x039d, 0x039e, 0x039f,

0x03a0, 0x03a1, 0x03c2, 0x03a3, 0x03a4, 0x03a5, 0x03a6, 0x03a7, 0x03a8, 0x03a9,

0x03aa, 0x03ab, 0x038c, 0x038e, 0x038f, 0x03cf, 0x0392, 0x0398, 0x03d2, 0x03d3,

0x03d4, 0x03a6, 0x03a0, 0x03d7, 0x03d8, 0x03d9, 0x03da, 0x03db, 0x03dc, 0x03dd,

0x03de, 0x03df, 0x03e0, 0x03e1, 0x03e2, 0x03e2, 0x03e4, 0x03e4, 0x03e6, 0x03e6,

0x03e8, 0x03e8, 0x03ea, 0x03ea, 0x03ec, 0x03ec, 0x03ee, 0x03ee, 0x039a, 0x03a1,

0x03f2, 0x03f3, 0x03f4, 0x03f5, 0x03f6, 0x03f7, 0x03f8, 0x03f9, 0x03fa, 0x03fb,

0x03fc, 0x03fd, 0x03fe, 0x03ff, 0x0400, 0x0401, 0x0402, 0x0403, 0x0404, 0x0405,

0x0406, 0x0407, 0x0408, 0x0409, 0x040a, 0x040b, 0x040c, 0x040d, 0x040e, 0x040f,

0x0410, 0x0411, 0x0412, 0x0413, 0x0414, 0x0415, 0x0416, 0x0417, 0x0418, 0x0419,

0x041a, 0x041b, 0x041c, 0x041d, 0x041e, 0x041f, 0x0420, 0x0421, 0x0422, 0x0423,

0x0424, 0x0425, 0x0426, 0x0427, 0x0428, 0x0429, 0x042a, 0x042b, 0x042c, 0x042d,

0x042e, 0x042f, 0x0410, 0x0411, 0x0412, 0x0413, 0x0414, 0x0415, 0x0416, 0x0417,

0x0418, 0x0419, 0x041a, 0x041b, 0x041c, 0x041d, 0x041e, 0x041f, 0x0420, 0x0421,

0x0422, 0x0423, 0x0424, 0x0425, 0x0426, 0x0427, 0x0428, 0x0429, 0x042a, 0x042b,

0x042c, 0x042d, 0x042e, 0x042f, 0x0450, 0x0401, 0x0402, 0x0403, 0x0404, 0x0405,

0x0406, 0x0407, 0x0408, 0x0409, 0x040a, 0x040b, 0x040c, 0x045d, 0x040e, 0x040f,

0x0460, 0x0460, 0x0462, 0x0462, 0x0464, 0x0464, 0x0466, 0x0466, 0x0468, 0x0468,

0x046a, 0x046a, 0x046c, 0x046c, 0x046e, 0x046e, 0x0470, 0x0470, 0x0472, 0x0472,

0x0474, 0x0474, 0x0476, 0x0476, 0x0478, 0x0478, 0x047a, 0x047a, 0x047c, 0x047c,

0x047e, 0x047e, 0x0480, 0x0480, 0x0482, 0x0483, 0x0484, 0x0485, 0x0486, 0x0487,

0x0488, 0x0489, 0x048a, 0x048b, 0x048c, 0x048d, 0x048e, 0x048f, 0x0490, 0x0490,

0x0492, 0x0492, 0x0494, 0x0494, 0x0496, 0x0496, 0x0498, 0x0498, 0x049a, 0x049a,

0x049c, 0x049c, 0x049e, 0x049e, 0x04a0, 0x04a0, 0x04a2, 0x04a2, 0x04a4, 0x04a4,

0x04a6, 0x04a6, 0x04a8, 0x04a8, 0x04aa, 0x04aa, 0x04ac, 0x04ac, 0x04ae, 0x04ae,

0x04b0, 0x04b0, 0x04b2, 0x04b2, 0x04b4, 0x04b4, 0x04b6, 0x04b6, 0x04b8, 0x04b8,

0x04ba, 0x04ba, 0x04bc, 0x04bc, 0x04be, 0x04be, 0x04c0, 0x04c1, 0x04c1, 0x04c3,

0x04c3, 0x04c5, 0x04c6, 0x04c7, 0x04c7, 0x04c9, 0x04ca, 0x04cb, 0x04cb, 0x04cd,

0x04ce, 0x04cf, 0x04d0, 0x04d0, 0x04d2, 0x04d2, 0x04d4, 0x04d4, 0x04d6, 0x04d6,

0x04d8, 0x04d8, 0x04da, 0x04da, 0x04dc, 0x04dc, 0x04de, 0x04de, 0x04e0, 0x04e0,

0x04e2, 0x04e2, 0x04e4, 0x04e4, 0x04e6, 0x04e6, 0x04e8, 0x04e8, 0x04ea, 0x04ea,

0x04ec, 0x04ed, 0x04ee, 0x04ee, 0x04f0, 0x04f0, 0x04f2, 0x04f2, 0x04f4, 0x04f4,

0x04f6, 0x04f7, 0x04f8, 0x04f8, 0x04fa, 0x04fb, 0x04fc, 0x04fd, 0x04fe, 0x04ff,

0x0500, 0x0501, 0x0502, 0x0503, 0x0504, 0x0505, 0x0506, 0x0507, 0x0508, 0x0509,

0x050a, 0x050b, 0x050c, 0x050d, 0x050e, 0x050f, 0x0510, 0x0511, 0x0512, 0x0513,

0x0514, 0x0515, 0x0516, 0x0517, 0x0518, 0x0519, 0x051a, 0x051b, 0x051c, 0x051d,

0x051e, 0x051f, 0x0520, 0x0521, 0x0522, 0x0523, 0x0524, 0x0525, 0x0526, 0x0527,

0x0528, 0x0529, 0x052a, 0x052b, 0x052c, 0x052d, 0x052e, 0x052f, 0x0530, 0x0531,

0x0532, 0x0533, 0x0534, 0x0535, 0x0536, 0x0537, 0x0538, 0x0539, 0x053a, 0x053b,

0x053c, 0x053d, 0x053e, 0x053f, 0x0540, 0x0541, 0x0542, 0x0543, 0x0544, 0x0545,

0x0546, 0x0547, 0x0548, 0x0549, 0x054a, 0x054b, 0x054c, 0x054d, 0x054e, 0x054f,

0x0550, 0x0551, 0x0552, 0x0553, 0x0554, 0x0555, 0x0556, 0x0557, 0x0558, 0x0559,

0x055a, 0x055b, 0x055c, 0x055d, 0x055e, 0x055f, 0x0560, 0x0531, 0x0532, 0x0533,

0x0534, 0x0535, 0x0536, 0x0537, 0x0538, 0x0539, 0x053a, 0x053b, 0x053c, 0x053d,

0x053e, 0x053f, 0x0540, 0x0541, 0x0542, 0x0543, 0x0544, 0x0545, 0x0546, 0x0547,

0x0548, 0x0549, 0x054a, 0x054b, 0x054c, 0x054d, 0x054e, 0x054f, 0x0550, 0x0551,

0x0552, 0x0553, 0x0554, 0x0555, 0x0556

};

static const wchar kchMinUpper3 = 0x1e01;

static const wchar kchLimUpper3 = 0x1ff4;

static const wchar g\_mpchchUpper3[kchLimUpper3 - kchMinUpper3] =

{

0x1e00, 0x1e02, 0x1e02, 0x1e04, 0x1e04, 0x1e06, 0x1e06, 0x1e08, 0x1e08, 0x1e0a,

0x1e0a, 0x1e0c, 0x1e0c, 0x1e0e, 0x1e0e, 0x1e10, 0x1e10, 0x1e12, 0x1e12, 0x1e14,

0x1e14, 0x1e16, 0x1e16, 0x1e18, 0x1e18, 0x1e1a, 0x1e1a, 0x1e1c, 0x1e1c, 0x1e1e,

0x1e1e, 0x1e20, 0x1e20, 0x1e22, 0x1e22, 0x1e24, 0x1e24, 0x1e26, 0x1e26, 0x1e28,

0x1e28, 0x1e2a, 0x1e2a, 0x1e2c, 0x1e2c, 0x1e2e, 0x1e2e, 0x1e30, 0x1e30, 0x1e32,

0x1e32, 0x1e34, 0x1e34, 0x1e36, 0x1e36, 0x1e38, 0x1e38, 0x1e3a, 0x1e3a, 0x1e3c,

0x1e3c, 0x1e3e, 0x1e3e, 0x1e40, 0x1e40, 0x1e42, 0x1e42, 0x1e44, 0x1e44, 0x1e46,

0x1e46, 0x1e48, 0x1e48, 0x1e4a, 0x1e4a, 0x1e4c, 0x1e4c, 0x1e4e, 0x1e4e, 0x1e50,

0x1e50, 0x1e52, 0x1e52, 0x1e54, 0x1e54, 0x1e56, 0x1e56, 0x1e58, 0x1e58, 0x1e5a,

0x1e5a, 0x1e5c, 0x1e5c, 0x1e5e, 0x1e5e, 0x1e60, 0x1e60, 0x1e62, 0x1e62, 0x1e64,

0x1e64, 0x1e66, 0x1e66, 0x1e68, 0x1e68, 0x1e6a, 0x1e6a, 0x1e6c, 0x1e6c, 0x1e6e,

0x1e6e, 0x1e70, 0x1e70, 0x1e72, 0x1e72, 0x1e74, 0x1e74, 0x1e76, 0x1e76, 0x1e78,

0x1e78, 0x1e7a, 0x1e7a, 0x1e7c, 0x1e7c, 0x1e7e, 0x1e7e, 0x1e80, 0x1e80, 0x1e82,

0x1e82, 0x1e84, 0x1e84, 0x1e86, 0x1e86, 0x1e88, 0x1e88, 0x1e8a, 0x1e8a, 0x1e8c,

0x1e8c, 0x1e8e, 0x1e8e, 0x1e90, 0x1e90, 0x1e92, 0x1e92, 0x1e94, 0x1e94, 0x1e96,

0x1e97, 0x1e98, 0x1e99, 0x1e9a, 0x1e9b, 0x1e9c, 0x1e9d, 0x1e9e, 0x1e9f, 0x1ea0,

0x1ea0, 0x1ea2, 0x1ea2, 0x1ea4, 0x1ea4, 0x1ea6, 0x1ea6, 0x1ea8, 0x1ea8, 0x1eaa,

0x1eaa, 0x1eac, 0x1eac, 0x1eae, 0x1eae, 0x1eb0, 0x1eb0, 0x1eb2, 0x1eb2, 0x1eb4,

0x1eb4, 0x1eb6, 0x1eb6, 0x1eb8, 0x1eb8, 0x1eba, 0x1eba, 0x1ebc, 0x1ebc, 0x1ebe,

0x1ebe, 0x1ec0, 0x1ec0, 0x1ec2, 0x1ec2, 0x1ec4, 0x1ec4, 0x1ec6, 0x1ec6, 0x1ec8,

0x1ec8, 0x1eca, 0x1eca, 0x1ecc, 0x1ecc, 0x1ece, 0x1ece, 0x1ed0, 0x1ed0, 0x1ed2,

0x1ed2, 0x1ed4, 0x1ed4, 0x1ed6, 0x1ed6, 0x1ed8, 0x1ed8, 0x1eda, 0x1eda, 0x1edc,

0x1edc, 0x1ede, 0x1ede, 0x1ee0, 0x1ee0, 0x1ee2, 0x1ee2, 0x1ee4, 0x1ee4, 0x1ee6,

0x1ee6, 0x1ee8, 0x1ee8, 0x1eea, 0x1eea, 0x1eec, 0x1eec, 0x1eee, 0x1eee, 0x1ef0,

0x1ef0, 0x1ef2, 0x1ef2, 0x1ef4, 0x1ef4, 0x1ef6, 0x1ef6, 0x1ef8, 0x1ef8, 0x1efa,

0x1efb, 0x1efc, 0x1efd, 0x1efe, 0x1eff, 0x1f08, 0x1f09, 0x1f0a, 0x1f0b, 0x1f0c,

0x1f0d, 0x1f0e, 0x1f0f, 0x1f08, 0x1f09, 0x1f0a, 0x1f0b, 0x1f0c, 0x1f0d, 0x1f0e,

0x1f0f, 0x1f18, 0x1f19, 0x1f1a, 0x1f1b, 0x1f1c, 0x1f1d, 0x1f16, 0x1f17, 0x1f18,

0x1f19, 0x1f1a, 0x1f1b, 0x1f1c, 0x1f1d, 0x1f1e, 0x1f1f, 0x1f28, 0x1f29, 0x1f2a,

0x1f2b, 0x1f2c, 0x1f2d, 0x1f2e, 0x1f2f, 0x1f28, 0x1f29, 0x1f2a, 0x1f2b, 0x1f2c,

0x1f2d, 0x1f2e, 0x1f2f, 0x1f38, 0x1f39, 0x1f3a, 0x1f3b, 0x1f3c, 0x1f3d, 0x1f3e,

0x1f3f, 0x1f38, 0x1f39, 0x1f3a, 0x1f3b, 0x1f3c, 0x1f3d, 0x1f3e, 0x1f3f, 0x1f48,

0x1f49, 0x1f4a, 0x1f4b, 0x1f4c, 0x1f4d, 0x1f46, 0x1f47, 0x1f48, 0x1f49, 0x1f4a,

0x1f4b, 0x1f4c, 0x1f4d, 0x1f4e, 0x1f4f, 0x1f50, 0x1f59, 0x1f52, 0x1f5b, 0x1f54,

0x1f5d, 0x1f56, 0x1f5f, 0x1f58, 0x1f59, 0x1f5a, 0x1f5b, 0x1f5c, 0x1f5d, 0x1f5e,

0x1f5f, 0x1f68, 0x1f69, 0x1f6a, 0x1f6b, 0x1f6c, 0x1f6d, 0x1f6e, 0x1f6f, 0x1f68,

0x1f69, 0x1f6a, 0x1f6b, 0x1f6c, 0x1f6d, 0x1f6e, 0x1f6f, 0x1fba, 0x1fbb, 0x1fc8,

0x1fc9, 0x1fca, 0x1fcb, 0x1fda, 0x1fdb, 0x1ff8, 0x1ff9, 0x1fea, 0x1feb, 0x1ffa,

0x1ffb, 0x1f7e, 0x1f7f, 0x1f88, 0x1f89, 0x1f8a, 0x1f8b, 0x1f8c, 0x1f8d, 0x1f8e,

0x1f8f, 0x1f88, 0x1f89, 0x1f8a, 0x1f8b, 0x1f8c, 0x1f8d, 0x1f8e, 0x1f8f, 0x1f98,

0x1f99, 0x1f9a, 0x1f9b, 0x1f9c, 0x1f9d, 0x1f9e, 0x1f9f, 0x1f98, 0x1f99, 0x1f9a,

0x1f9b, 0x1f9c, 0x1f9d, 0x1f9e, 0x1f9f, 0x1fa8, 0x1fa9, 0x1faa, 0x1fab, 0x1fac,

0x1fad, 0x1fae, 0x1faf, 0x1fa8, 0x1fa9, 0x1faa, 0x1fab, 0x1fac, 0x1fad, 0x1fae,

0x1faf, 0x1fb8, 0x1fb9, 0x1fb2, 0x1fbc, 0x1fb4, 0x1fb5, 0x1fb6, 0x1fb7, 0x1fb8,

0x1fb9, 0x1fba, 0x1fbb, 0x1fbc, 0x1fbd, 0x1fbe, 0x1fbf, 0x1fc0, 0x1fc1, 0x1fc2,

0x1fcc, 0x1fc4, 0x1fc5, 0x1fc6, 0x1fc7, 0x1fc8, 0x1fc9, 0x1fca, 0x1fcb, 0x1fcc,

0x1fcd, 0x1fce, 0x1fcf, 0x1fd8, 0x1fd9, 0x1fd2, 0x1fd3, 0x1fd4, 0x1fd5, 0x1fd6,

0x1fd7, 0x1fd8, 0x1fd9, 0x1fda, 0x1fdb, 0x1fdc, 0x1fdd, 0x1fde, 0x1fdf, 0x1fe8,

0x1fe9, 0x1fe2, 0x1fe3, 0x1fe4, 0x1fec, 0x1fe6, 0x1fe7, 0x1fe8, 0x1fe9, 0x1fea,

0x1feb, 0x1fec, 0x1fed, 0x1fee, 0x1fef, 0x1ff0, 0x1ff1, 0x1ff2, 0x1ffc

};

static const wchar kchMinUpper4 = 0x2170;

static const wchar kchLimUpper4 = 0x2180;

static const short kdchUpper4 = -16;

static const wchar kchMinUpper5 = 0x24d0;

static const wchar kchLimUpper5 = 0x24ea;

static const short kdchUpper5 = -26;

static const wchar kchMinUpper6 = 0xff41;

static const wchar kchLimUpper6 = 0xff5b;

static const short kdchUpper6 = -32;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Mapping to lower case.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static const wchar kchMinLower1 = 0x0041;

static const wchar kchLimLower1 = 0x0217;

static const wchar g\_mpchchLower1[kchLimLower1 - kchMinLower1] =

{

0x0061, 0x0062, 0x0063, 0x0064, 0x0065, 0x0066, 0x0067, 0x0068, 0x0069, 0x006a,

0x006b, 0x006c, 0x006d, 0x006e, 0x006f, 0x0070, 0x0071, 0x0072, 0x0073, 0x0074,

0x0075, 0x0076, 0x0077, 0x0078, 0x0079, 0x007a, 0x005b, 0x005c, 0x005d, 0x005e,

0x005f, 0x0060, 0x0061, 0x0062, 0x0063, 0x0064, 0x0065, 0x0066, 0x0067, 0x0068,

0x0069, 0x006a, 0x006b, 0x006c, 0x006d, 0x006e, 0x006f, 0x0070, 0x0071, 0x0072,

0x0073, 0x0074, 0x0075, 0x0076, 0x0077, 0x0078, 0x0079, 0x007a, 0x007b, 0x007c,

0x007d, 0x007e, 0x007f, 0x0080, 0x0081, 0x0082, 0x0083, 0x0084, 0x0085, 0x0086,

0x0087, 0x0088, 0x0089, 0x008a, 0x008b, 0x008c, 0x008d, 0x008e, 0x008f, 0x0090,

0x0091, 0x0092, 0x0093, 0x0094, 0x0095, 0x0096, 0x0097, 0x0098, 0x0099, 0x009a,

0x009b, 0x009c, 0x009d, 0x009e, 0x009f, 0x00a0, 0x00a1, 0x00a2, 0x00a3, 0x00a4,

0x00a5, 0x00a6, 0x00a7, 0x00a8, 0x00a9, 0x00aa, 0x00ab, 0x00ac, 0x00ad, 0x00ae,

0x00af, 0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7, 0x00b8,

0x00b9, 0x00ba, 0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf, 0x00e0, 0x00e1, 0x00e2,

0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7, 0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec,

0x00ed, 0x00ee, 0x00ef, 0x00f0, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6,

0x00d7, 0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x00fe, 0x00df, 0x00e0,

0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7, 0x00e8, 0x00e9, 0x00ea,

0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef, 0x00f0, 0x00f1, 0x00f2, 0x00f3, 0x00f4,

0x00f5, 0x00f6, 0x00f7, 0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x00fe,

0x00ff, 0x0101, 0x0101, 0x0103, 0x0103, 0x0105, 0x0105, 0x0107, 0x0107, 0x0109,

0x0109, 0x010b, 0x010b, 0x010d, 0x010d, 0x010f, 0x010f, 0x0111, 0x0111, 0x0113,

0x0113, 0x0115, 0x0115, 0x0117, 0x0117, 0x0119, 0x0119, 0x011b, 0x011b, 0x011d,

0x011d, 0x011f, 0x011f, 0x0121, 0x0121, 0x0123, 0x0123, 0x0125, 0x0125, 0x0127,

0x0127, 0x0129, 0x0129, 0x012b, 0x012b, 0x012d, 0x012d, 0x012f, 0x012f, 0x0069,

0x0131, 0x0133, 0x0133, 0x0135, 0x0135, 0x0137, 0x0137, 0x0138, 0x013a, 0x013a,

0x013c, 0x013c, 0x013e, 0x013e, 0x0140, 0x0140, 0x0142, 0x0142, 0x0144, 0x0144,

0x0146, 0x0146, 0x0148, 0x0148, 0x0149, 0x014b, 0x014b, 0x014d, 0x014d, 0x014f,

0x014f, 0x0151, 0x0151, 0x0153, 0x0153, 0x0155, 0x0155, 0x0157, 0x0157, 0x0159,

0x0159, 0x015b, 0x015b, 0x015d, 0x015d, 0x015f, 0x015f, 0x0161, 0x0161, 0x0163,

0x0163, 0x0165, 0x0165, 0x0167, 0x0167, 0x0169, 0x0169, 0x016b, 0x016b, 0x016d,

0x016d, 0x016f, 0x016f, 0x0171, 0x0171, 0x0173, 0x0173, 0x0175, 0x0175, 0x0177,

0x0177, 0x00ff, 0x017a, 0x017a, 0x017c, 0x017c, 0x017e, 0x017e, 0x017f, 0x0180,

0x0253, 0x0183, 0x0183, 0x0185, 0x0185, 0x0254, 0x0188, 0x0188, 0x0256, 0x0257,

0x018c, 0x018c, 0x018d, 0x0258, 0x0259, 0x025b, 0x0192, 0x0192, 0x0260, 0x0263,

0x0195, 0x0269, 0x0268, 0x0199, 0x0199, 0x019a, 0x019b, 0x026f, 0x0272, 0x019e,

0x019f, 0x01a1, 0x01a1, 0x01a3, 0x01a3, 0x01a5, 0x01a5, 0x01a6, 0x01a8, 0x01a8,

0x0283, 0x01aa, 0x01ab, 0x01ad, 0x01ad, 0x0288, 0x01b0, 0x01b0, 0x028a, 0x028b,

0x01b4, 0x01b4, 0x01b6, 0x01b6, 0x0292, 0x01b9, 0x01b9, 0x01ba, 0x01bb, 0x01bd,

0x01bd, 0x01be, 0x01bf, 0x01c0, 0x01c1, 0x01c2, 0x01c3, 0x01c6, 0x01c6, 0x01c6,

0x01c9, 0x01c9, 0x01c9, 0x01cc, 0x01cc, 0x01cc, 0x01ce, 0x01ce, 0x01d0, 0x01d0,

0x01d2, 0x01d2, 0x01d4, 0x01d4, 0x01d6, 0x01d6, 0x01d8, 0x01d8, 0x01da, 0x01da,

0x01dc, 0x01dc, 0x01dd, 0x01df, 0x01df, 0x01e1, 0x01e1, 0x01e3, 0x01e3, 0x01e5,

0x01e5, 0x01e7, 0x01e7, 0x01e9, 0x01e9, 0x01eb, 0x01eb, 0x01ed, 0x01ed, 0x01ef,

0x01ef, 0x01f0, 0x01f3, 0x01f3, 0x01f3, 0x01f5, 0x01f5, 0x01f6, 0x01f7, 0x01f8,

0x01f9, 0x01fb, 0x01fb, 0x01fd, 0x01fd, 0x01ff, 0x01ff, 0x0201, 0x0201, 0x0203,

0x0203, 0x0205, 0x0205, 0x0207, 0x0207, 0x0209, 0x0209, 0x020b, 0x020b, 0x020d,

0x020d, 0x020f, 0x020f, 0x0211, 0x0211, 0x0213, 0x0213, 0x0215, 0x0215, 0x0217

};

static const wchar kchMinLower2 = 0x0386;

static const wchar kchLimLower2 = 0x0557;

static const wchar g\_mpchchLower2[kchLimLower2 - kchMinLower2] =

{

0x03ac, 0x0387, 0x03ad, 0x03ae, 0x03af, 0x038b, 0x03cc, 0x038d, 0x03cd, 0x03ce,

0x0390, 0x03b1, 0x03b2, 0x03b3, 0x03b4, 0x03b5, 0x03b6, 0x03b7, 0x03b8, 0x03b9,

0x03ba, 0x03bb, 0x03bc, 0x03bd, 0x03be, 0x03bf, 0x03c0, 0x03c1, 0x03a2, 0x03c3,

0x03c4, 0x03c5, 0x03c6, 0x03c7, 0x03c8, 0x03c9, 0x03ca, 0x03cb, 0x03ac, 0x03ad,

0x03ae, 0x03af, 0x03b0, 0x03b1, 0x03b2, 0x03b3, 0x03b4, 0x03b5, 0x03b6, 0x03b7,

0x03b8, 0x03b9, 0x03ba, 0x03bb, 0x03bc, 0x03bd, 0x03be, 0x03bf, 0x03c0, 0x03c1,

0x03c2, 0x03c3, 0x03c4, 0x03c5, 0x03c6, 0x03c7, 0x03c8, 0x03c9, 0x03ca, 0x03cb,

0x03cc, 0x03cd, 0x03ce, 0x03cf, 0x03d0, 0x03d1, 0x03d2, 0x03d3, 0x03d4, 0x03d5,

0x03d6, 0x03d7, 0x03d8, 0x03d9, 0x03da, 0x03db, 0x03dc, 0x03dd, 0x03de, 0x03df,

0x03e0, 0x03e1, 0x03e3, 0x03e3, 0x03e5, 0x03e5, 0x03e7, 0x03e7, 0x03e9, 0x03e9,

0x03eb, 0x03eb, 0x03ed, 0x03ed, 0x03ef, 0x03ef, 0x03f0, 0x03f1, 0x03f2, 0x03f3,

0x03f4, 0x03f5, 0x03f6, 0x03f7, 0x03f8, 0x03f9, 0x03fa, 0x03fb, 0x03fc, 0x03fd,

0x03fe, 0x03ff, 0x0400, 0x0451, 0x0452, 0x0453, 0x0454, 0x0455, 0x0456, 0x0457,

0x0458, 0x0459, 0x045a, 0x045b, 0x045c, 0x040d, 0x045e, 0x045f, 0x0430, 0x0431,

0x0432, 0x0433, 0x0434, 0x0435, 0x0436, 0x0437, 0x0438, 0x0439, 0x043a, 0x043b,

0x043c, 0x043d, 0x043e, 0x043f, 0x0440, 0x0441, 0x0442, 0x0443, 0x0444, 0x0445,

0x0446, 0x0447, 0x0448, 0x0449, 0x044a, 0x044b, 0x044c, 0x044d, 0x044e, 0x044f,

0x0430, 0x0431, 0x0432, 0x0433, 0x0434, 0x0435, 0x0436, 0x0437, 0x0438, 0x0439,

0x043a, 0x043b, 0x043c, 0x043d, 0x043e, 0x043f, 0x0440, 0x0441, 0x0442, 0x0443,

0x0444, 0x0445, 0x0446, 0x0447, 0x0448, 0x0449, 0x044a, 0x044b, 0x044c, 0x044d,

0x044e, 0x044f, 0x0450, 0x0451, 0x0452, 0x0453, 0x0454, 0x0455, 0x0456, 0x0457,

0x0458, 0x0459, 0x045a, 0x045b, 0x045c, 0x045d, 0x045e, 0x045f, 0x0461, 0x0461,

0x0463, 0x0463, 0x0465, 0x0465, 0x0467, 0x0467, 0x0469, 0x0469, 0x046b, 0x046b,

0x046d, 0x046d, 0x046f, 0x046f, 0x0471, 0x0471, 0x0473, 0x0473, 0x0475, 0x0475,

0x0477, 0x0477, 0x0479, 0x0479, 0x047b, 0x047b, 0x047d, 0x047d, 0x047f, 0x047f,

0x0481, 0x0481, 0x0482, 0x0483, 0x0484, 0x0485, 0x0486, 0x0487, 0x0488, 0x0489,

0x048a, 0x048b, 0x048c, 0x048d, 0x048e, 0x048f, 0x0491, 0x0491, 0x0493, 0x0493,

0x0495, 0x0495, 0x0497, 0x0497, 0x0499, 0x0499, 0x049b, 0x049b, 0x049d, 0x049d,

0x049f, 0x049f, 0x04a1, 0x04a1, 0x04a3, 0x04a3, 0x04a5, 0x04a5, 0x04a7, 0x04a7,

0x04a9, 0x04a9, 0x04ab, 0x04ab, 0x04ad, 0x04ad, 0x04af, 0x04af, 0x04b1, 0x04b1,

0x04b3, 0x04b3, 0x04b5, 0x04b5, 0x04b7, 0x04b7, 0x04b9, 0x04b9, 0x04bb, 0x04bb,

0x04bd, 0x04bd, 0x04bf, 0x04bf, 0x04c0, 0x04c2, 0x04c2, 0x04c4, 0x04c4, 0x04c5,

0x04c6, 0x04c8, 0x04c8, 0x04c9, 0x04ca, 0x04cc, 0x04cc, 0x04cd, 0x04ce, 0x04cf,

0x04d1, 0x04d1, 0x04d3, 0x04d3, 0x04d5, 0x04d5, 0x04d7, 0x04d7, 0x04d9, 0x04d9,

0x04db, 0x04db, 0x04dd, 0x04dd, 0x04df, 0x04df, 0x04e1, 0x04e1, 0x04e3, 0x04e3,

0x04e5, 0x04e5, 0x04e7, 0x04e7, 0x04e9, 0x04e9, 0x04eb, 0x04eb, 0x04ec, 0x04ed,

0x04ef, 0x04ef, 0x04f1, 0x04f1, 0x04f3, 0x04f3, 0x04f5, 0x04f5, 0x04f6, 0x04f7,

0x04f9, 0x04f9, 0x04fa, 0x04fb, 0x04fc, 0x04fd, 0x04fe, 0x04ff, 0x0500, 0x0501,

0x0502, 0x0503, 0x0504, 0x0505, 0x0506, 0x0507, 0x0508, 0x0509, 0x050a, 0x050b,

0x050c, 0x050d, 0x050e, 0x050f, 0x0510, 0x0511, 0x0512, 0x0513, 0x0514, 0x0515,

0x0516, 0x0517, 0x0518, 0x0519, 0x051a, 0x051b, 0x051c, 0x051d, 0x051e, 0x051f,

0x0520, 0x0521, 0x0522, 0x0523, 0x0524, 0x0525, 0x0526, 0x0527, 0x0528, 0x0529,

0x052a, 0x052b, 0x052c, 0x052d, 0x052e, 0x052f, 0x0530, 0x0561, 0x0562, 0x0563,

0x0564, 0x0565, 0x0566, 0x0567, 0x0568, 0x0569, 0x056a, 0x056b, 0x056c, 0x056d,

0x056e, 0x056f, 0x0570, 0x0571, 0x0572, 0x0573, 0x0574, 0x0575, 0x0576, 0x0577,

0x0578, 0x0579, 0x057a, 0x057b, 0x057c, 0x057d, 0x057e, 0x057f, 0x0580, 0x0581,

0x0582, 0x0583, 0x0584, 0x0585, 0x0586,

};

static const wchar kchMinLower3 = 0x10a0;

static const wchar kchLimLower3 = 0x10c6;

static const short kdchLower3 = 48;

static const wchar kchMinLower4 = 0x1e00;

static const wchar kchLimLower4 = 0x1ffd;

static const wchar g\_mpchchLower4[kchLimLower4 - kchMinLower4] =

{

0x1e01, 0x1e01, 0x1e03, 0x1e03, 0x1e05, 0x1e05, 0x1e07, 0x1e07, 0x1e09, 0x1e09,

0x1e0b, 0x1e0b, 0x1e0d, 0x1e0d, 0x1e0f, 0x1e0f, 0x1e11, 0x1e11, 0x1e13, 0x1e13,

0x1e15, 0x1e15, 0x1e17, 0x1e17, 0x1e19, 0x1e19, 0x1e1b, 0x1e1b, 0x1e1d, 0x1e1d,

0x1e1f, 0x1e1f, 0x1e21, 0x1e21, 0x1e23, 0x1e23, 0x1e25, 0x1e25, 0x1e27, 0x1e27,

0x1e29, 0x1e29, 0x1e2b, 0x1e2b, 0x1e2d, 0x1e2d, 0x1e2f, 0x1e2f, 0x1e31, 0x1e31,

0x1e33, 0x1e33, 0x1e35, 0x1e35, 0x1e37, 0x1e37, 0x1e39, 0x1e39, 0x1e3b, 0x1e3b,

0x1e3d, 0x1e3d, 0x1e3f, 0x1e3f, 0x1e41, 0x1e41, 0x1e43, 0x1e43, 0x1e45, 0x1e45,

0x1e47, 0x1e47, 0x1e49, 0x1e49, 0x1e4b, 0x1e4b, 0x1e4d, 0x1e4d, 0x1e4f, 0x1e4f,

0x1e51, 0x1e51, 0x1e53, 0x1e53, 0x1e55, 0x1e55, 0x1e57, 0x1e57, 0x1e59, 0x1e59,

0x1e5b, 0x1e5b, 0x1e5d, 0x1e5d, 0x1e5f, 0x1e5f, 0x1e61, 0x1e61, 0x1e63, 0x1e63,

0x1e65, 0x1e65, 0x1e67, 0x1e67, 0x1e69, 0x1e69, 0x1e6b, 0x1e6b, 0x1e6d, 0x1e6d,

0x1e6f, 0x1e6f, 0x1e71, 0x1e71, 0x1e73, 0x1e73, 0x1e75, 0x1e75, 0x1e77, 0x1e77,

0x1e79, 0x1e79, 0x1e7b, 0x1e7b, 0x1e7d, 0x1e7d, 0x1e7f, 0x1e7f, 0x1e81, 0x1e81,

0x1e83, 0x1e83, 0x1e85, 0x1e85, 0x1e87, 0x1e87, 0x1e89, 0x1e89, 0x1e8b, 0x1e8b,

0x1e8d, 0x1e8d, 0x1e8f, 0x1e8f, 0x1e91, 0x1e91, 0x1e93, 0x1e93, 0x1e95, 0x1e95,

0x1e96, 0x1e97, 0x1e98, 0x1e99, 0x1e9a, 0x1e9b, 0x1e9c, 0x1e9d, 0x1e9e, 0x1e9f,

0x1ea1, 0x1ea1, 0x1ea3, 0x1ea3, 0x1ea5, 0x1ea5, 0x1ea7, 0x1ea7, 0x1ea9, 0x1ea9,

0x1eab, 0x1eab, 0x1ead, 0x1ead, 0x1eaf, 0x1eaf, 0x1eb1, 0x1eb1, 0x1eb3, 0x1eb3,

0x1eb5, 0x1eb5, 0x1eb7, 0x1eb7, 0x1eb9, 0x1eb9, 0x1ebb, 0x1ebb, 0x1ebd, 0x1ebd,

0x1ebf, 0x1ebf, 0x1ec1, 0x1ec1, 0x1ec3, 0x1ec3, 0x1ec5, 0x1ec5, 0x1ec7, 0x1ec7,

0x1ec9, 0x1ec9, 0x1ecb, 0x1ecb, 0x1ecd, 0x1ecd, 0x1ecf, 0x1ecf, 0x1ed1, 0x1ed1,

0x1ed3, 0x1ed3, 0x1ed5, 0x1ed5, 0x1ed7, 0x1ed7, 0x1ed9, 0x1ed9, 0x1edb, 0x1edb,

0x1edd, 0x1edd, 0x1edf, 0x1edf, 0x1ee1, 0x1ee1, 0x1ee3, 0x1ee3, 0x1ee5, 0x1ee5,

0x1ee7, 0x1ee7, 0x1ee9, 0x1ee9, 0x1eeb, 0x1eeb, 0x1eed, 0x1eed, 0x1eef, 0x1eef,

0x1ef1, 0x1ef1, 0x1ef3, 0x1ef3, 0x1ef5, 0x1ef5, 0x1ef7, 0x1ef7, 0x1ef9, 0x1ef9,

0x1efa, 0x1efb, 0x1efc, 0x1efd, 0x1efe, 0x1eff, 0x1f00, 0x1f01, 0x1f02, 0x1f03,

0x1f04, 0x1f05, 0x1f06, 0x1f07, 0x1f00, 0x1f01, 0x1f02, 0x1f03, 0x1f04, 0x1f05,

0x1f06, 0x1f07, 0x1f10, 0x1f11, 0x1f12, 0x1f13, 0x1f14, 0x1f15, 0x1f16, 0x1f17,

0x1f10, 0x1f11, 0x1f12, 0x1f13, 0x1f14, 0x1f15, 0x1f1e, 0x1f1f, 0x1f20, 0x1f21,

0x1f22, 0x1f23, 0x1f24, 0x1f25, 0x1f26, 0x1f27, 0x1f20, 0x1f21, 0x1f22, 0x1f23,

0x1f24, 0x1f25, 0x1f26, 0x1f27, 0x1f30, 0x1f31, 0x1f32, 0x1f33, 0x1f34, 0x1f35,

0x1f36, 0x1f37, 0x1f30, 0x1f31, 0x1f32, 0x1f33, 0x1f34, 0x1f35, 0x1f36, 0x1f37,

0x1f40, 0x1f41, 0x1f42, 0x1f43, 0x1f44, 0x1f45, 0x1f46, 0x1f47, 0x1f40, 0x1f41,

0x1f42, 0x1f43, 0x1f44, 0x1f45, 0x1f4e, 0x1f4f, 0x1f50, 0x1f51, 0x1f52, 0x1f53,

0x1f54, 0x1f55, 0x1f56, 0x1f57, 0x1f58, 0x1f51, 0x1f5a, 0x1f53, 0x1f5c, 0x1f55,

0x1f5e, 0x1f57, 0x1f60, 0x1f61, 0x1f62, 0x1f63, 0x1f64, 0x1f65, 0x1f66, 0x1f67,

0x1f60, 0x1f61, 0x1f62, 0x1f63, 0x1f64, 0x1f65, 0x1f66, 0x1f67, 0x1f70, 0x1f71,

0x1f72, 0x1f73, 0x1f74, 0x1f75, 0x1f76, 0x1f77, 0x1f78, 0x1f79, 0x1f7a, 0x1f7b,

0x1f7c, 0x1f7d, 0x1f7e, 0x1f7f, 0x1f80, 0x1f81, 0x1f82, 0x1f83, 0x1f84, 0x1f85,

0x1f86, 0x1f87, 0x1f80, 0x1f81, 0x1f82, 0x1f83, 0x1f84, 0x1f85, 0x1f86, 0x1f87,

0x1f90, 0x1f91, 0x1f92, 0x1f93, 0x1f94, 0x1f95, 0x1f96, 0x1f97, 0x1f90, 0x1f91,

0x1f92, 0x1f93, 0x1f94, 0x1f95, 0x1f96, 0x1f97, 0x1fa0, 0x1fa1, 0x1fa2, 0x1fa3,

0x1fa4, 0x1fa5, 0x1fa6, 0x1fa7, 0x1fa0, 0x1fa1, 0x1fa2, 0x1fa3, 0x1fa4, 0x1fa5,

0x1fa6, 0x1fa7, 0x1fb0, 0x1fb1, 0x1fb2, 0x1fb3, 0x1fb4, 0x1fb5, 0x1fb6, 0x1fb7,

0x1fb0, 0x1fb1, 0x1f70, 0x1f71, 0x1fb3, 0x1fbd, 0x1fbe, 0x1fbf, 0x1fc0, 0x1fc1,

0x1fc2, 0x1fc3, 0x1fc4, 0x1fc5, 0x1fc6, 0x1fc7, 0x1f72, 0x1f73, 0x1f74, 0x1f75,

0x1fc3, 0x1fcd, 0x1fce, 0x1fcf, 0x1fd0, 0x1fd1, 0x1fd2, 0x1fd3, 0x1fd4, 0x1fd5,

0x1fd6, 0x1fd7, 0x1fd0, 0x1fd1, 0x1f76, 0x1f77, 0x1fdc, 0x1fdd, 0x1fde, 0x1fdf,

0x1fe0, 0x1fe1, 0x1fe2, 0x1fe3, 0x1fe4, 0x1fe5, 0x1fe6, 0x1fe7, 0x1fe0, 0x1fe1,

0x1f7a, 0x1f7b, 0x1fe5, 0x1fed, 0x1fee, 0x1fef, 0x1ff0, 0x1ff1, 0x1ff2, 0x1ff3,

0x1ff4, 0x1ff5, 0x1ff6, 0x1ff7, 0x1f78, 0x1f79, 0x1f7c, 0x1f7d, 0x1ff3

};

static const wchar kchMinLower5 = 0x2160;

static const wchar kchLimLower5 = 0x2170;

static const short kdchLower5 = 16;

static const wchar kchMinLower6 = 0x24b6;

static const wchar kchLimLower6 = 0x24d0;

static const short kdchLower6 = 26;

static const wchar kchMinLower7 = 0xff21;

static const wchar kchLimLower7 = 0xff3b;

static const short kdchLower7 = 32;

wchar\_t CharToLowerCase(wchar\_t chSrc)

{

wchar\_t chDst = chSrc;

if (chSrc < kchLimLower1)

{

if (chSrc >= kchMinLower1)

{

chDst = g\_mpchchLower1[chSrc - kchMinLower1];

}

}

else if (chSrc < kchLimLower4)

{

if (chSrc < kchMinLower3)

{

if (chSrc < kchLimLower2)

{

if (chSrc >= kchMinLower2)

{

chDst = g\_mpchchLower2[chSrc - kchMinLower2];

}

}

}

else if (chSrc < kchLimLower3)

{

chDst = chSrc + kdchLower3;

}

else if (chSrc >= kchMinLower4)

{

chDst = g\_mpchchLower4[chSrc - kchMinLower4];

}

}

else if (chSrc < kchLimLower6)

{

if (chSrc < kchLimLower5)

{

if (chSrc >= kchMinLower5)

{

chDst = chSrc + kdchLower5;

}

}

else if (chSrc >= kchMinLower6)

{

chDst = chSrc + kdchLower6;

}

}

else if (chSrc < kchLimLower7)

{

if (chSrc >= kchMinLower7)

{

chDst = chSrc + kdchLower7;

}

}

return chDst;

}

wchar\_t CharToUpperCase(wchar\_t chSrc)

{

wchar\_t chDst = chSrc;

if (chSrc < kchLimUpper1)

{

if (chSrc >= kchMinUpper1)

{

chDst = g\_mpchchUpper1[chSrc - kchMinUpper1];

}

}

else if (chSrc < kchLimUpper4)

{

if (chSrc < kchMinUpper3)

{

if (chSrc < kchLimUpper2)

{

if (chSrc >= kchMinUpper2)

{

chDst = g\_mpchchUpper2[chSrc - kchMinUpper2];

}

}

}

else if (chSrc < kchLimUpper3)

{

chDst = g\_mpchchUpper3[chSrc - kchMinUpper3];

}

else if (chSrc >= kchMinUpper4)

{

chDst = chSrc + kdchUpper4;

}

}

else if (chSrc < kchMinUpper6)

{

if (chSrc < kchLimUpper5)

{

if (chSrc >= kchMinUpper5)

{

chDst = chSrc + kdchUpper5;

}

}

}

else if (chSrc < kchLimUpper6)

{

chDst = chSrc + kdchUpper6;

}

return chDst;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

wchar\_t CharToLowerCase(wchar\_t inChar);

wchar\_t CharToUpperCase(wchar\_t inChar);

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCommonPch.h"

#include "common\vtregistry.h"

#include "Common\vtinfo.h"

#if DBG

VirtualTableRegistry::TableEntry VirtualTableRegistry::m\_knownVtables[MAX\_KNOWN\_VTABLES];

UINT VirtualTableRegistry::m\_knownVtableCount = 0;

void VirtualTableRegistry::Add(INT\_PTR vtable, LPCSTR className)

{

Assert(m\_knownVtableCount < MAX\_KNOWN\_VTABLES);

if (m\_knownVtableCount < MAX\_KNOWN\_VTABLES)

{

m\_knownVtables[m\_knownVtableCount].vtable = vtable;

m\_knownVtables[m\_knownVtableCount].className = className;

++m\_knownVtableCount;

}

}

VtableHashMap \*

VirtualTableRegistry::CreateVtableHashMap(ArenaAllocator \* alloc)

{

VtableHashMap \* vtableHashMap = Anew(alloc, VtableHashMap, alloc, MAX\_KNOWN\_VTABLES);

// All classes that derive from RecyclableObject must include DEFINE\_VTABLE\_CTOR which invokes VirtualTableRegistry::Add

// at class initialization time. Here we add them to our hash table for easy lookup. Note that on a release build

// the vtables are merged and thus not all of our types will be registered. So, we can only use this method

// in a debug build. If we wanted use in a release build, we'll have to explicitly add the vtables to the hash and

// then validate that we have got all the vtables by comparing in chk build against VirtualTableRegistry.

for (UINT i=0; i < VirtualTableRegistry::m\_knownVtableCount; i++)

{

INT\_PTR vtable = VirtualTableRegistry::m\_knownVtables[i].vtable;

LPCSTR className = VirtualTableRegistry::m\_knownVtables[i].className;

vtableHashMap->Add(vtable, className);

}

return vtableHashMap;

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// These definitions allow us to programmatically access a class' vtable for:

//

// 1. Cross-context marshalling: when an object needs to be marshalled, we

// replace its vtable with another that will force the marshalling

//

// 2. Heap enumeration: for any instance that we want to report, basically

// everything that inherits from RecyclableObject, we register the class'

// vtable with the enumeration code. This allows us to compare a GC heap

// address against the vtable list to determine whether it's a type we care about

// (we don't distinguish between RecyclableObjects and other types

// of storage in the GC heap).

//----------------------------------------------------------------------------

#pragma once

enum VirtualTableInfoCtorEnum

{

VirtualTableInfoCtorValue

};

#define MAX\_KNOWN\_VTABLES 300

#if DBG

#ifdef HEAP\_ENUMERATION\_VALIDATION

#define ENABLE\_VALIDATE\_VTABLE\_CTOR 1

#define HEAP\_ENUMERATION\_LIBRARY\_OBJECT\_COOKIE -1

#endif

#endif

#ifdef ENABLE\_VALIDATE\_VTABLE\_CTOR

#define POST\_ALLOCATION\_VALIDATION 1

// PostAllocationCallback allows us to validate that an object derived from a known

// base class has included DEFINE\_VTABLE\_CTOR. Although this could be statically determined,

// the language doesn't support a way to enforce that, so we do it at runtime on object allocation.

//

// Any class that includes DEFINE\_VTABLE\_CTOR will also override ValidateVtableRegistered() function

// with a type-specific version that compares the vtable of the type against VirtualTableInfo<T>::Address.

// Any class that doesn't include DEFINE\_VTABLE\_CTOR will pick up its parent's version of ValidateVtableRegistered

// and the type-specific vtable comparison will fail.

#define DEFINE\_VALIDATE\_HAS\_VTABLE\_CTOR(T) \

void PostAllocationCallbackForHeapEnumValidation(const type\_info& objType, Js::DynamicObject\* obj) \

{ \

if (Js::Configuration::Global.flags.ValidateHeapEnum) \

{ \

((Js::RecyclableObject\*)obj)->ValidateVtableRegistered(objType); \

} \

Js::JavascriptLibrary \*library = obj->GetLibrary(); \

if (! library || ! library->GetScriptContext()->IsInitialized()) \

{ \

obj->SetHeapEnumValidationCookie(HEAP\_ENUMERATION\_LIBRARY\_OBJECT\_COOKIE); \

} \

} \

#define DECLARE\_VALIDATE\_VTABLE\_REGISTERED\_NOBASE\_ABSTRACT(T) \

virtual void ValidateVtableRegistered(const type\_info& objType) = 0;

#define VALIDATE\_VTABLE\_REGISTERED\_BODY(T) \

{ \

if (typeid(T) != objType) \

{ \

AssertMsg(typeid(T) == objType, "Class derived from Js::RecyclableObject missing DEFINE\_VTABLE\_CTOR"); \

Output::Print(L"%S missing DEFINE\_VTABLE\_CTOR\n", objType.name()); \

} \

}

#define DECLARE\_VALIDATE\_VTABLE\_REGISTERED\_NOBASE(T) \

virtual void ValidateVtableRegistered(const type\_info& objType) VALIDATE\_VTABLE\_REGISTERED\_BODY(T) \

#define DEFINE\_VALIDATE\_VTABLE\_REGISTERED(T) \

void ValidateVtableRegistered(const type\_info& objType) override VALIDATE\_VTABLE\_REGISTERED\_BODY(T)

#else

#define DEFINE\_VALIDATE\_HAS\_VTABLE\_CTOR(T)

#define DECLARE\_VALIDATE\_VTABLE\_REGISTERED\_NOBASE\_ABSTRACT(T)

#define DECLARE\_VALIDATE\_VTABLE\_REGISTERED\_NOBASE(T)

#define DEFINE\_VALIDATE\_VTABLE\_REGISTERED(T)

#define VALIDATE\_HAS\_VTABLE\_CTOR(T, obj)

#endif

class VirtualTableInfoBase

{

public:

static INT\_PTR GetVirtualTable(void \* ptr) { return (\*(INT\_PTR\*)ptr); }

};

template <typename T>

class VirtualTableInfo : public VirtualTableInfoBase

{

public:

static INT\_PTR const Address;

static INT\_PTR RegisterVirtualTable(INT\_PTR vtable);

static void SetVirtualTable(void \* ptr) { new (ptr) T(VirtualTableInfoCtorValue); }

static bool HasVirtualTable(void \* ptr) { return GetVirtualTable(ptr) == Address; }

};

#if !defined(USED\_IN\_STATIC\_LIB)

#pragma warning(disable:4238) // class rvalue used as lvalue

template <typename T>

INT\_PTR const VirtualTableInfo<T>::Address = VirtualTableInfo<T>::RegisterVirtualTable(\*(INT\_PTR const\*)&T(VirtualTableInfoCtorValue));

#endif

#define DEFINE\_VTABLE\_CTOR\_NOBASE\_ABSTRACT(T) \

T(VirtualTableInfoCtorEnum v) {} \

enum RegisterVTableEnum { RegisterVTable = 1 };

#define DEFINE\_VTABLE\_CTOR\_NOBASE(T) \

friend class VirtualTableInfo<T>; \

DEFINE\_VTABLE\_CTOR\_NOBASE\_ABSTRACT(T)

#define DEFINE\_VTABLE\_CTOR\_ABSTRACT(T, Base, ...) \

T(VirtualTableInfoCtorEnum v) : Base(v), \_\_VA\_ARGS\_\_ {}

#define DEFINE\_VTABLE\_CTOR(T, Base) \

friend class VirtualTableInfo<T>; \

DEFINE\_VTABLE\_CTOR\_ABSTRACT(T, Base) \

DEFINE\_VALIDATE\_VTABLE\_REGISTERED(T);

// Used by non-RecyclableObject

#define DEFINE\_VTABLE\_CTOR\_NO\_REGISTER(T, Base, ...) \

friend class VirtualTableInfo<T>; \

DEFINE\_VTABLE\_CTOR\_ABSTRACT(T, Base, \_\_VA\_ARGS\_\_) \

enum RegisterVTableEnum { RegisterVTable = 0 };

#define DEFINE\_VTABLE\_CTOR\_MEMBER\_INIT(T, Base, Member) \

friend class VirtualTableInfo<T>; \

T(VirtualTableInfoCtorEnum v) : Base(v), Member(v) {} \

DEFINE\_VALIDATE\_VTABLE\_REGISTERED(T);

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

#define DEFINE\_GETCPPNAME\_ABSTRACT() \

virtual const char\* GetCppName() const = 0;

#define DEFINE\_GETCPPNAME() \

virtual const char\* GetCppName() const { return typeid(this).name(); }

#else

#define DEFINE\_GETCPPNAME\_ABSTRACT()

#define DEFINE\_GETCPPNAME()

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// Note: VirtualTableInfo<T>::RegisterVirtualTable can be put into .cpp file but for that

// we'll have to define quite a lot of explicit template instantiations for all classes that

// use DEFINE\_VTABLE\_CTOR, to fix unresolved externals.

#if DBG

// static

template <typename T>

inline INT\_PTR VirtualTableInfo<T>::RegisterVirtualTable(INT\_PTR vtable)

{

#if ENABLE\_VALIDATE\_VTABLE\_CTOR

//printf("m\_vtableMapHash->Add(VirtualTableInfo<%s>::Address, dummy);\n", typeid(T).name());

#endif

if (T::RegisterVTable)

{

VirtualTableRegistry::Add(vtable, typeid(T).name());

}

return vtable;

}

#else

// static

template <typename T>

inline INT\_PTR VirtualTableInfo<T>::RegisterVirtualTable(INT\_PTR vtable)

{

return vtable;

}

#endif DBG

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#if DBG

typedef JsUtil::BaseDictionary<INT\_PTR, LPCSTR, ArenaAllocator> VtableHashMap;

class VirtualTableRegistry

{

public:

static void Add(INT\_PTR vtable, LPCSTR className);

static VtableHashMap \* CreateVtableHashMap(ArenaAllocator \* allocator);

private:

struct TableEntry

{

INT\_PTR vtable;

LPCSTR className;

};

static TableEntry m\_knownVtables[];

static UINT m\_knownVtableCount;

};

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

class AllocSizeMath

{

static const size\_t MaxMemory = static\_cast<size\_t>(-1);

public:

// Works for both 32bit and 64bit size\_t arithmetic. It's also pretty

// optimal in the cases where either left or right or both are small, compile-

// time constants.

static size\_t Add(size\_t left, size\_t right)

{

size\_t allocSize = left + right;

if (allocSize < left)

{

// Integer overflow in computation, allocate max memory which will fail with out of memory

return MaxMemory;

}

return allocSize;

}

template <typename T>

static T Min(const T& a, const T& b)

{

return (a < b ? a : b);

}

// Optimized for right being a constant power of 2...

static size\_t Mul(size\_t left, size\_t right)

{

size\_t allocSize = left \* right;

if (left != (allocSize / right))

{

// Integer overflow in computation, allocate max memory which will fail with out of memory

return MaxMemory;

}

return allocSize;

}

static size\_t Align(size\_t size, size\_t alignment)

{

if (size >= (MaxMemory & ~(alignment - 1)))

{

return MaxMemory & ~(alignment - 1);

}

return Math::Align(size, alignment);

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#pragma intrinsic(memcpy)

extern void \_\_stdcall js\_memcpy\_s(\_\_bcount(sizeInBytes) void \*dst, size\_t sizeInBytes, \_\_in\_bcount(count) const void \*src, size\_t count);

extern void \_\_stdcall js\_wmemcpy\_s(\_\_ecount(sizeInWords) wchar\_t \*dst, size\_t sizeInWords, \_\_in\_ecount(count) const wchar\_t \*src, size\_t count);

// A virtualized thread id. The physical thread on which an instance of the runtime is executed can change but a

// ThreadContextId should be invariant.

// Many parts of the runtime expect to only be called by the execution, or "main", thread. Hosts have the prerogative

// of allocating physical execution threads, thus the physical execution thread is not invariant. Hosts also own the

// virtualization of the thread id and are responsible for making ThreadContextIds invariant.

typedef void \* ThreadContextId;

#define NoThreadContextId (ThreadContextId)NULL

// Functions that need to be implemented by user of Common library

namespace Js

{

// Forward declaration

class ScriptContext;

};

namespace JsUtil

{

struct ExternalApi

{

// Returns the current execution ThreadContextId

static ThreadContextId GetCurrentThreadContextId();

static bool RaiseOutOfMemoryIfScriptActive();

static bool RaiseStackOverflowIfScriptActive(Js::ScriptContext \* scriptContext, PVOID returnAddress);

static bool RaiseOnIntOverflow();

static void RecoverUnusedMemory();

#if DBG || defined(EXCEPTION\_CHECK)

static BOOL IsScriptActiveOnCurrentThreadContext();

#endif

// By default, implemented in Dll\Jscript\ScriptEngine.cpp

// Anyone who statically links with jscript.common.common.lib has to implement this

// This is used to determine which regkey we should read while loading the configuration

static LPWSTR GetFeatureKeyName();

};

};

// Just an alias to ExternalApi::GetCurrentThreadContextId

inline ThreadContextId GetCurrentThreadContextId()

{

return JsUtil::ExternalApi::GetCurrentThreadContextId();

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#undef AssertMsg

#undef Assert

#if defined(DBG) && !defined(DIAG\_DAC)

// AutoDebug functions that are only available in DEBUG builds

\_declspec(selectany) int AssertCount = 0;

\_declspec(selectany) int AssertsToConsole = false;

\_declspec(thread, selectany) int IsInAssert = false;

#if !defined(USED\_IN\_STATIC\_LIB)

#define REPORT\_ASSERT(f, comment) Js::Throw::ReportAssert(\_\_FILE\_\_, \_\_LINE\_\_, STRINGIZE((f)), comment)

#define LOG\_ASSERT() Js::Throw::LogAssert()

#else

#define REPORT\_ASSERT(f, comment) FALSE

#define LOG\_ASSERT()

#endif

#ifdef NTBUILD

#include <ntassert.h>

#define RAISE\_ASSERTION(comment) NT\_ASSERTMSG(comment, FALSE)

#else

#include <assert.h>

#define RAISE\_ASSERTION(comment) DbgRaiseAssertionFailure()

#endif

#define AssertMsg(f, comment) \

{ \

if (!(f)) \

{ \

AssertCount++; \

LOG\_ASSERT(); \

IsInAssert = TRUE; \

if (!REPORT\_ASSERT(f, comment)) \

{ \

RAISE\_ASSERTION(comment); \

} \

IsInAssert = FALSE; \

\_\_analysis\_assume(false); \

} \

}

#define Assert(exp) AssertMsg(exp, #exp)

#define AssertVerify(exp) Assert(exp)

#define Assume(x) Assert(x)

#define DebugOnly(x) x

#else // DBG

#define AssertMsg(f, comment) ((void) 0)

#define Assert(exp) ((void) 0)

#ifdef NTBUILD

#include <ntassert.h>

#define AssertVerify(exp) NT\_VERIFY(exp) // Execute the expression but don't do anything with the result in non-debug builds

#else

#define AssertVerify(exp) (exp)

#endif

#define Assume(x) \_\_assume(x)

#define DebugOnly(x)

#endif // DBG

#define AnalysisAssert(x) Assert(x); \_\_analysis\_assume(x)

#define AnalysisAssertMsg(x, comment) AssertMsg(x, comment); \_\_analysis\_assume(x)

#define Unused(var) var;

#define UNREACHED (0)

#ifndef CompileAssert

#define CompileAssert(e) static\_assert(e, #e)

#endif

// We set IsPointer<T>::IsTrue to true if T is a pointer type

// Otherwise, it's set to false

template <class T>

struct IsPointer

{

enum

{

IsTrue = false

};

};

template <class T>

struct IsPointer<T\*>

{

enum

{

IsTrue = true

};

};

// Trick adopted from WinRT/WinTypes/Value.h

template <class T1, class T2>

struct IsSame

{

enum

{

IsTrue = false

};

};

template <class T1>

struct IsSame<T1, T1>

{

enum

{

IsTrue = true

};

};

// From Legacy engine - don't use

#define AssertPvCb(pv, cb) AssertMsg(0 != (pv) || 0 == (cb), "bad ptr")

#define AssertPvCbN(pv, cb) //NO-OP

#define AssertPvCbR(pv, cb) AssertMsg(0 != (pv) || 0 == (cb), "bad ptr")

#define AssertPsz(psz) AssertMsg(0 != (psz), "bad psz")

#define AssertPszN(psz) //NO-OP

#define AssertMem(pvar) AssertPvCb(pvar, sizeof(\*(pvar)))

#define AssertMemN(pvar) AssertPvCbN(pvar, sizeof(\*(pvar)))

#define AssertMemR(pvar) AssertPvCbR(pvar, sizeof(\*(pvar)))

#define AssertArrMem(prgv, cv) AssertPvCb(prgv, (cv) \* sizeof(\*(prgv)))

#define AssertArrMemR(prgv, cv) AssertPvCbR(prgv, (cv) \* sizeof(\*(prgv)))

#define AssertThis() Assert(0 != (this) && (this)->AssertValid())

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include <initguid.h>

#define szJScript9Lock L"{16d51579-a30b-4c8b-a276-0ff4dc41e755}"

DEFINE\_GUID(JScript9Lock, 0x16d51579, 0xa30b, 0x4c8b, 0xa2, 0x76, 0x0f, 0xf4, 0xdc, 0x41, 0xe7, 0x55);

#define szChakraLock L"{1b7cd997-e5ff-4932-a7a6-2a9e636da385}"

DEFINE\_GUID(ChakraLock, 0x1b7cd997, 0xe5ff, 0x4932, 0xa7, 0xa6, 0x2a, 0x9e, 0x63, 0x6d, 0xa3, 0x85);

#define szChakraCoreLock L"{EB5ADA93-9744-4F8F-B94C-D21B4D6B4E71}"

DEFINE\_GUID(ChakraCoreLock, 0xeb5ada93, 0x9744, 0x4f8f, 0xb9, 0x4c, 0xd2, 0x1b, 0x4d, 0x6b, 0x4e, 0x71);

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

class AutoFILE : public BasePtr<FILE>

{

public:

AutoFILE(FILE \* file = nullptr) : BasePtr<FILE>(file) {};

~AutoFILE()

{

Close();

}

AutoFILE& operator=(FILE \* file)

{

Close();

this->ptr = file;

return \*this;

}

void Close()

{

if (ptr != nullptr)

{

fclose(ptr);

}

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template <typename T>

class BasePtr

{

public:

BasePtr(T \* ptr = nullptr) : ptr(ptr) {}

T \*\* operator&() { Assert(ptr == nullptr); return &ptr; }

T \* operator->() const { Assert(ptr != nullptr); return ptr; }

operator T\*() const { return ptr; }

// Detach currently owned ptr. WARNING: This object no longer owns/manages the ptr.

T \* Detach()

{

T \* ret = ptr;

ptr = nullptr;

return ret;

}

protected:

T \* ptr;

private:

BasePtr(const BasePtr<T>& ptr); // Disable

BasePtr& operator=(BasePtr<T> const& ptr); // Disable

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#include "core\BinaryFeatureControl.h"

//For making direct call in release binaries.

#if !defined(DELAYLOAD\_SET\_CFG\_TARGET)

extern "C"

WINBASEAPI

BOOL

WINAPI

GetProcessMitigationPolicy(

\_\_in HANDLE hProcess,

\_\_in PROCESS\_MITIGATION\_POLICY MitigationPolicy,

\_\_out\_bcount(nLength) PVOID lpBuffer,

\_\_in SIZE\_T nLength

);

#endif // ENABLE\_DEBUG\_CONFIG\_OPTIONS

bool

BinaryFeatureControl::RecyclerTest()

{

#ifdef RECYCLER\_TEST

return true;

#else

return false;

#endif

}

BOOL

BinaryFeatureControl::GetMitigationPolicyForProcess(\_\_in HANDLE hProcess, \_\_in PROCESS\_MITIGATION\_POLICY MitigationPolicy, \_\_out\_bcount(nLength) PVOID lpBuffer, \_\_in SIZE\_T nLength)

{

#if !defined(DELAYLOAD\_SET\_CFG\_TARGET)

return GetProcessMitigationPolicy(hProcess, MitigationPolicy, lpBuffer, nLength);

#else

return FALSE;

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// This call binary level provides control to turn on or off a feature.

// A default implementation that enables all features is included in jscript.common.common.lib

// To override, include an object that includes the definition of all the functions in this file

// on the linker command line. The linker always processes symbols from objects on the command line

// first, thus the override will be chosen instead of the default one.

class BinaryFeatureControl

{

public:

static bool RecyclerTest();

static BOOL GetMitigationPolicyForProcess(\_\_in HANDLE hProcess, \_\_in PROCESS\_MITIGATION\_POLICY MitigationPolicy, \_\_out\_bcount(nLength) PVOID lpBuffer, \_\_in SIZE\_T nLength);

};

<?xml version="1.0" encoding="utf-8"?>

<Project DefaultTargets="Build" ToolsVersion="12.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Import Condition="'$(ChakraBuildPathImported)'!='true'" Project="$(SolutionDir)Chakra.Build.Paths.props" />

<Import Project="$(BuildConfigPropsPath)Chakra.Build.ProjectConfiguration.props" />

<PropertyGroup Label="Globals">

<TargetName>Chakra.Common.Core</TargetName>

<ProjectGuid>{CC4153FF-AC3E-4734-B562-CC23812DF31B}</ProjectGuid>

<RootNamespace>JS</RootNamespace>

<Keyword>Win32Proj</Keyword>

</PropertyGroup>

<PropertyGroup Label="Configuration">

<ConfigurationType>StaticLibrary</ConfigurationType>

</PropertyGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />

<Import Project="$(BuildConfigPropsPath)Chakra.Build.props" />

<PropertyGroup>

<\_ProjectFileVersion>10.0.30319.1</\_ProjectFileVersion>

</PropertyGroup>

<ItemDefinitionGroup>

<ClCompile>

<AdditionalIncludeDirectories>

$(MSBuildThisFileDirectory)..;

%(AdditionalIncludeDirectories)

</AdditionalIncludeDirectories>

<PrecompiledHeader>Use</PrecompiledHeader>

<PrecompiledHeaderFile>CommonCorePch.h</PrecompiledHeaderFile>

</ClCompile>

</ItemDefinitionGroup>

<ItemGroup>

<ClCompile Include="$(MSBuildThisFileDirectory)BinaryFeatureControl.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CmdParser.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CodexAssert.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)ConfigFlagsTable.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)ConfigParser.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)DbgHelpSymbolManager.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)DelayLoadLibrary.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)EtwTraceCore.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)FaultInjection.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Output.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)PerfCounter.cpp" />

<None Include="PerfCounterImpl.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)PerfCounterSet.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)ProfileInstrument.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)ProfileMemory.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)StackBackTrace.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SysInfo.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CommonCorePch.cpp">

<PrecompiledHeader>Create</PrecompiledHeader>

</ClCompile>

</ItemGroup>

<ItemGroup>

<ClInclude Include="AllocSizeMath.h" />

<ClInclude Include="api.h" />

<ClInclude Include="Assertions.h" />

<ClInclude Include="AtomLockGuids.h" />

<ClInclude Include="AutoFILE.h" />

<ClInclude Include="BinaryFeatureControl.h" />

<ClInclude Include="BasePtr.h" />

<ClInclude Include="CmdParser.h" />

<ClInclude Include="CommonCorePch.h" />

<ClInclude Include="CommonTypedefs.h" />

<ClInclude Include="ConfigFlagsTable.h" />

<ClInclude Include="ConfigParser.h" />

<ClInclude Include="CriticalSection.h" />

<ClInclude Include="DbgHelpSymbolManager.h" />

<ClInclude Include="DelayLoadLibrary.h" />

<ClInclude Include="EtwTraceCore.h" />

<ClInclude Include="FaultInjection.h" />

<ClInclude Include="FaultTypes.h" />

<ClInclude Include="FinalizableObject.h" />

<ClInclude Include="ICustomConfigFlags.h" />

<ClInclude Include="Output.h" />

<ClInclude Include="PerfCounter.h" />

<ClInclude Include="PerfCounterSet.h" />

<ClInclude Include="ProfileInstrument.h" />

<ClInclude Include="ProfileMemory.h" />

<ClInclude Include="StackBackTrace.h" />

<ClInclude Include="SysInfo.h" />

<ClInclude Include="..\Warnings.h" />

<ClInclude Include="..\CommonDefines.h" />

<ClInclude Include="..\CommonBasic.h" />

<ClInclude Include="..\CommonMin.h" />

<ClInclude Include="..\CommonMinMemory.h" />

<ClInclude Include="..\Common.h" />

<ClInclude Include="..\BackEndAPI.h" />

<ClInclude Include="..\commoninl.h" />

<ClInclude Include="..\DictionaryStats.h" />

<ClInclude Include="..\EnumClassHelp.h" />

<ClInclude Include="..\ConfigFlagsList.h" />

<ClInclude Include="..\EnumHelp.h" />

<ClInclude Include="..\TargetVer.h" />

<ClInclude Include="..\TemplateParameter.h" />

</ItemGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.targets" Condition="exists('$(BuildConfigPropsPath)Chakra.Build.targets')" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />

</Project>

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#include "core\ICustomConfigFlags.h"

#include "core\CmdParser.h"

using namespace Js;

///----------------------------------------------------------------------------

///

/// CmdLineArgsParser::ParseString

///

/// Parses an string token. There are 2 ways to specify it.

/// 1. Quoted - " " Any character within quotes is parsed as string.

/// if the quotes are not closed, its an error.

/// 2. UnQuoted - End of string is indicated by a space/end of stream.

/// If fTreatColonAsSeperator is mentioned, then we break

/// at colon also.

///

///

/// Empty string "" is treated as Exception()

///

///----------------------------------------------------------------------------

LPWSTR

CmdLineArgsParser::ParseString(\_\_inout\_ecount(ceBuffer) LPWSTR buffer, size\_t ceBuffer, bool fTreatColonAsSeperator)

{

wchar\_t \*out = buffer;

size\_t len = 0;

if('"' == CurChar())

{

NextChar();

while('"' != CurChar())

{

if(0 == CurChar())

{

throw Exception(L"Unmatched quote");

}

//

// MaxTokenSize - 1 because we need 1 extra position for null termination

//

if (len >= ceBuffer - 1)

{

throw Exception(L"String token too large to parse");

}

out[len++] = CurChar();

NextChar();

}

NextChar();

}

else

{

bool fDone = false;

while(!fDone)

{

switch(CurChar())

{

case ' ':

case ',':

case 0:

fDone = 1;

break;

case '-':

case ':':

if(fTreatColonAsSeperator)

{

fDone = true;

break;

}

else

{

// Fallthrough

}

default:

if(len >= MaxTokenSize -1)

{

throw Exception(L"String token too large to parse");

}

out[len++] = CurChar();

NextChar();

}

}

}

if(0 == len)

{

throw Exception(L"String Token Expected");

}

out[len] = '\0';

return buffer;

}

///----------------------------------------------------------------------------

///

/// CmdLineArgsParser::ParseSourceFunctionIds

///

/// Parses for sourceContextId and FunctionId pairs

///----------------------------------------------------------------------------

Js::SourceFunctionNode

CmdLineArgsParser::ParseSourceFunctionIds()

{

uint functionId, sourceId;

if ('\*' == CurChar())

{

sourceId = 1;

functionId = (uint)-2;

NextChar();

}

else if ('+' == CurChar())

{

sourceId = 1;

functionId = (uint)-1;

NextChar();

}

else

{

functionId = sourceId = ParseInteger();

if ('.' == CurChar())

{

NextChar();

if ('\*' == CurChar())

{

functionId = (uint)-2;

NextChar();

}

else if ('+' == CurChar())

{

functionId = (uint)-1;

NextChar();

}

else

{

functionId = ParseInteger();

}

}

else

{

sourceId = 1;

}

}

return SourceFunctionNode(sourceId, functionId);

}

///----------------------------------------------------------------------------

///

/// CmdLineArgsParser::ParseInteger

///

/// Parses signed integer. Checks for overflow and underflows.

///----------------------------------------------------------------------------

int

CmdLineArgsParser::ParseInteger()

{

int result = 0;

int sign = 1;

if('-' == CurChar())

{

sign = -1;

NextChar();

}

if(!IsDigit())

{

throw Exception(L"Integer Expected");

}

int base = 10;

if ('0' == CurChar())

{

NextChar();

if (CurChar() == 'x')

{

NextChar();

base = 16;

}

// Should the else case be parse as octal?

}

while(IsDigit() || (base == 16 && IsHexDigit()))

{

int currentDigit = (int)(CurChar() - '0');

if (currentDigit > 9)

{

Assert(base == 16);

if (CurChar() < 'F')

{

currentDigit = 10 + (int)(CurChar() - 'A');

}

else

{

currentDigit = 10 + (int)(CurChar() - 'a');

}

Assert(currentDigit < 16);

}

result = result \* base + (int)(CurChar() - '0');

if(result < 0)

{

// overflow or underflow in case sign = -1

throw Exception(L"Integer too large to parse");

}

NextChar();

}

return result \* sign;

}

///----------------------------------------------------------------------------

///

/// CmdLineArgsParser::ParseRange

///

/// Parses :-

/// range = int | int '-' int | range, range

///

///----------------------------------------------------------------------------

void

CmdLineArgsParser::ParseRange(Js::Range \*pRange)

{

SourceFunctionNode r1 = ParseSourceFunctionIds();

SourceFunctionNode r2;

switch(CurChar())

{

case '-':

NextChar();

r2 = ParseSourceFunctionIds();

if (r1.sourceContextId > r2.sourceContextId)

{

throw Exception(L"Left source index must be smaller than the Right source Index");

}

if ((r1.sourceContextId == r2.sourceContextId) &&

(r1.functionId > r2.functionId))

{

throw Exception(L"Left functionId must be smaller than the Right functionId when Source file is the same");

}

pRange->Add(r1, r2);

switch(CurChar())

{

case ',':

NextChar();

ParseRange(pRange);

break;

case ' ':

case 0:

break;

default:

throw Exception(L"Unexpected character while parsing Range");

}

break;

case ',':

pRange->Add(r1);

NextChar();

ParseRange(pRange);

break;

case ' ':

case 0:

pRange->Add(r1);

break;

default:

throw Exception(L"Unexpected character while parsing Range");

}

}

void

CmdLineArgsParser::ParseNumberRange(Js::NumberRange \*pRange)

{

int start = ParseInteger();

int end;

switch (CurChar())

{

case '-':

NextChar();

end = ParseInteger();

if (start > end)

{

throw Exception(L"Range start must be less than range end");

}

pRange->Add(start, end);

switch (CurChar())

{

case ',':

NextChar();

ParseNumberRange(pRange);

break;

case ' ':

case 0:

break;

default:

throw Exception(L"Unexpected character while parsing Range");

}

break;

case ',':

pRange->Add(start);

NextChar();

ParseNumberRange(pRange);

break;

case ' ':

case 0:

pRange->Add(start);

break;

default:

throw Exception(L"Unexpected character while parsing Range");

}

}

///----------------------------------------------------------------------------

///

/// CmdLineArgsParser::ParsePhase

///

/// Parses comma separated list of:

/// phase[:range]

/// phase is a string defined in Js:PhaseNames.

///

///----------------------------------------------------------------------------

void

CmdLineArgsParser::ParsePhase(Js::Phases \*pPhaseList)

{

wchar\_t buffer[MaxTokenSize];

ZeroMemory(buffer, sizeof(buffer));

Phase phase = ConfigFlagsTable::GetPhase(ParseString(buffer));

if(InvalidPhase == phase)

{

throw Exception(L"Invalid phase :");

}

pPhaseList->Enable(phase);

switch(CurChar())

{

case ':':

NextChar();

ParseRange(pPhaseList->GetRange(phase));

break;

case ',':

NextChar();

ParsePhase(pPhaseList);

break;

default:

break;

}

}

void

CmdLineArgsParser::ParseNumberSet(Js::NumberSet \* numberPairSet)

{

while (true)

{

int x = ParseInteger();

numberPairSet->Add(x);

if (CurChar() != ';')

{

break;

}

NextChar();

}

}

void

CmdLineArgsParser::ParseNumberPairSet(Js::NumberPairSet \* numberPairSet)

{

while (true)

{

int line = ParseInteger();

int col = -1;

if (CurChar() == ',')

{

NextChar();

col = ParseInteger();

}

numberPairSet->Add(line, col);

if (CurChar() != ';')

{

break;

}

NextChar();

}

}

bool

CmdLineArgsParser::ParseBoolean()

{

if (CurChar() == ':')

{

throw Exception(L"':' not expected with a boolean flag");

}

else if (CurChar() != '-' && CurChar() != ' ' && CurChar() != 0)

{

throw Exception(L"Invalid character after boolean flag");

}

else

{

return (CurChar() != '-');

}

}

BSTR

CmdLineArgsParser::GetCurrentString()

{

wchar\_t buffer[MaxTokenSize];

ZeroMemory(buffer, sizeof(buffer));

switch (CurChar())

{

case ':':

NextChar();

return SysAllocString(ParseString(buffer, MaxTokenSize, false));

case ' ':

case 0:

NextChar();

return nullptr;

default:

throw Exception(L"Expected ':'");

}

}

///----------------------------------------------------------------------------

///

/// CmdLineArgsParser::ParseFlag

///

/// Parses:

/// flag[:parameter]

/// Flag is a string defined in Js:FlagNames.

/// The type of expected parameter depends upon the flag. It can be

/// 1. String

/// 2. Number

/// 3. Boolean

/// 4. Phase

///

/// In case of boolean the presence no parameter is expected. the value of the

/// boolean flag is set to 'true'

///

///----------------------------------------------------------------------------

void

CmdLineArgsParser::ParseFlag()

{

wchar\_t buffer[MaxTokenSize];

ZeroMemory(buffer, sizeof(buffer));

LPWSTR flagString = ParseString(buffer);

Flag flag = ConfigFlagsTable::GetFlag(flagString);

if(InvalidFlag == flag)

{

if (pCustomConfigFlags != nullptr)

{

if (pCustomConfigFlags->ParseFlag(flagString, this))

{

return;

}

}

throw Exception(L"Invalid Flag");

}

FlagTypes flagType = ConfigFlagsTable::GetFlagType(flag);

AssertMsg(InvalidFlagType != flagType, "Invalid flag type");

this->flagTable.Enable(flag);

if(FlagBoolean == flagType)

{

Boolean boolValue = ParseBoolean();

this->flagTable.SetAsBoolean(flag, boolValue);

}

else

{

switch(CurChar())

{

case ':':

NextChar();

switch(flagType)

{

case FlagPhases:

ParsePhase(this->flagTable.GetAsPhase(flag));

break;

case FlagString:

\*this->flagTable.GetAsString(flag) = ParseString(buffer, MaxTokenSize, false);

break;

case FlagNumber:

\*this->flagTable.GetAsNumber(flag) = ParseInteger();

break;

case FlagNumberSet:

ParseNumberSet(this->flagTable.GetAsNumberSet(flag));

break;

case FlagNumberPairSet:

ParseNumberPairSet(this->flagTable.GetAsNumberPairSet(flag));

break;

case FlagNumberRange:

ParseNumberRange(this->flagTable.GetAsNumberRange(flag));

break;

default:

AssertMsg(0, "Flag not Handled");

}

break;

case ' ':

case 0:

break;

default:

throw Exception(L"Expected ':'");

}

}

}

///----------------------------------------------------------------------------

///

/// CmdLineArgsParser::Parse

///

/// The main loop which parses 1 flag at a time

///

///----------------------------------------------------------------------------

int

CmdLineArgsParser::Parse(int argc, \_\_in\_ecount(argc) LPWSTR argv[])

{

int err = 0;

for(int i = 1; i < argc; i++)

{

if ((err = Parse(argv[i])) != 0)

{

break;

}

}

return err;

}

int CmdLineArgsParser::Parse(\_\_in LPWSTR oneArg) throw()

{

int err = 0;

wchar\_t buffer[MaxTokenSize];

ZeroMemory(buffer, sizeof(buffer));

this->pszCurrentArg = oneArg;

AssertMsg(NULL != this->pszCurrentArg, "How can command line give NULL argv's");

try

{

switch(CurChar())

{

case '-' :

case '/':

NextChar();

if('?' == CurChar())

{

PrintUsage();

return -1;

}

else

{

ParseFlag();

}

break;

default:

if(NULL != this->flagTable.Filename)

{

throw Exception(L"Duplicate filename entry");

}

this->flagTable.Filename = ParseString(buffer, MaxTokenSize, false);

break;

}

}

catch(Exception &exp)

{

wprintf(L"%s : %s\n", (LPCWSTR)exp, oneArg);

err = -1;

}

return err;

}

///----------------------------------------------------------------------------

///

/// CmdLineArgsParser::CmdLineArgsParser

///

/// Constructor

///

///----------------------------------------------------------------------------

CmdLineArgsParser::CmdLineArgsParser(ICustomConfigFlags \* pCustomConfigFlags, Js::ConfigFlagsTable& flagTable) :

flagTable(flagTable), pCustomConfigFlags(pCustomConfigFlags)

{

this->pszCurrentArg = NULL;

}

CmdLineArgsParser::~CmdLineArgsParser()

{

flagTable.FinalizeConfiguration();

}

void CmdLineArgsParser::PrintUsage()

{

if (pCustomConfigFlags)

{

pCustomConfigFlags->PrintUsage();

return;

}

Js::ConfigFlagsTable::PrintUsageString();

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class CmdLineArgsParser

///

/// Parses the following grammar

///

/// range = integer | integer - integer | range,range

/// parameter = integer | string | phase[:range]

/// flag = string

/// phase = string

/// [-|/]flag[:parameter]

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

class CmdLineArgsParser : private ICmdLineArgsParser

{

// Data

private:

static const int MaxTokenSize = 512;

Js::ConfigFlagsTable& flagTable;

LPWSTR pszCurrentArg;

ICustomConfigFlags \* pCustomConfigFlags;

// Methods

public:

int Parse(int argc, \_\_in\_ecount(argc) LPWSTR argv[]);

int Parse(\_\_in LPWSTR token) throw();

CmdLineArgsParser(ICustomConfigFlags \* pCustomConfigFlags = nullptr, Js::ConfigFlagsTable& flagTable = Js::Configuration::Global.flags);

~CmdLineArgsParser();

// Helper Classes

private:

///----------------------------------------------------------------------------

///

/// class Exception

///

///----------------------------------------------------------------------------

class Exception {

LPCWSTR pszMsg;

public:

Exception(LPCWSTR message):

pszMsg(message)

{}

operator LPCWSTR () const

{

return this->pszMsg;

}

};

// Implementation

private:

bool ParseBoolean();

LPWSTR ParseString(\_\_inout\_ecount(ceBuffer) LPWSTR buffer, size\_t ceBuffer = MaxTokenSize, bool fTreatColonAsSeperator = true);

int ParseInteger();

Js::SourceFunctionNode ParseSourceFunctionIds();

void ParsePhase(Js::Phases \*pPhase);

void ParseRange(Js::Range \*range);

void ParseNumberRange(Js::NumberRange \*range);

void ParseFlag();

void ParseNumberSet(Js::NumberSet \* numberSet);

void ParseNumberPairSet(Js::NumberPairSet \* numberPairSet);

void PrintUsage();

wchar\_t CurChar()

{

return this->pszCurrentArg[0];

}

void NextChar()

{

this->pszCurrentArg++;

}

bool IsDigit()

{

return (CurChar() >='0' && CurChar() <= '9');

}

bool IsHexDigit()

{

return (CurChar() >= '0' && CurChar() <= '9') ||

(CurChar() >= 'A' && CurChar() <= 'F') ||

(CurChar() >= 'a' && CurChar() <= 'f');

}

// Implements ICmdLineArgsParser

virtual BSTR GetCurrentString() override;

virtual bool GetCurrentBoolean() override

{

return ParseBoolean();

}

virtual int GetCurrentInt() override

{

NextChar();

return ParseInteger();

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

// Method is expected to be implemented to link with codex.lib

// We have separate implementations for Chakra and IE

void CodexAssert(bool condition)

{

Assert(condition);

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonDefines.h"

#include "CommonMin.h"

#pragma warning(push)

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(HEAP\_TRACK\_ALLOC) || defined(ENABLE\_DEBUG\_CONFIG\_OPTIONS)

#include <typeinfo.h>

#endif

#pragma warning(pop)

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template<class T> inline

\_Post\_equal\_to\_(a < b ? a : b) \_Post\_satisfies\_(return <= a && return <= b)

const T& min(const T& a, const T& b) { return a < b ? a : b; }

template<class T> inline

\_Post\_equal\_to\_(a > b ? a : b) \_Post\_satisfies\_(return >= a && return >= b)

const T& max(const T& a, const T& b) { return a > b ? a : b; }

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

typedef wchar\_t wchar;

typedef unsigned int uint;

typedef unsigned short ushort;

typedef unsigned long ulong;

typedef signed char sbyte;

typedef \_\_int8 int8;

typedef \_\_int16 int16;

typedef \_\_int32 int32;

typedef \_\_int64 int64;

typedef unsigned char byte;

typedef unsigned \_\_int8 uint8;

typedef unsigned \_\_int16 uint16;

typedef unsigned \_\_int32 uint32;

typedef unsigned \_\_int64 uint64;

#if defined (\_WIN64)

typedef \_\_int64 intptr;

typedef unsigned \_\_int64 uintptr;

#else

typedef \_\_int32 intptr;

typedef unsigned \_\_int32 uintptr;

#endif

// charcount\_t represents a count of characters in a JavascriptString

// It is unsigned and the maximum value is (INT\_MAX-1)

typedef uint32 charcount\_t;

//A Unicode code point

typedef uint32 codepoint\_t;

const codepoint\_t INVALID\_CODEPOINT = (codepoint\_t)-1;

// Synonym for above, 2^31-1 is used as the limit to protect against addition overflow

typedef uint32 CharCount;

const CharCount MaxCharCount = INT\_MAX-1;

// As above, but 2^32-1 is used to signal a 'flag' condition (e.g. undefined)

typedef uint32 CharCountOrFlag;

const CharCountOrFlag CharCountFlag = (CharCountOrFlag)-1;

#define QUOTE(s) #s

#define STRINGIZE(s) QUOTE(s)

#define STRINGIZEW(s) TEXT(QUOTE(s))

namespace Js

{

typedef uint32 LocalFunctionId;

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#include "Memory\PageHeapBlockTypeFilter.h"

#include <initguid.h>

// {17DC713D-8B3E-4434-9DC8-90C275C75194}

DEFINE\_GUID(HybridDebuggingGuid, 0x17dc713d, 0x8b3e, 0x4434, 0x9d, 0xc8, 0x90, 0xc2, 0x75, 0xc7, 0x51, 0x94);

#undef DebugBreak

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAM")

namespace Js

{

NumberSet::NumberSet() : set(&NoCheckHeapAllocator::Instance) {}

void NumberSet::Add(uint32 x)

{

set.Item(x);

}

bool NumberSet::Contains(uint32 x)

{

return set.Contains(x);

}

NumberPairSet::NumberPairSet() : set(&NoCheckHeapAllocator::Instance) {}

void NumberPairSet::Add(uint32 x, uint32 y)

{

set.Item(NumberPair(x, y));

}

bool NumberPairSet::Contains(uint32 x, uint32 y)

{

return set.Contains(NumberPair(x, y));

}

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class String

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

String::String()

{

this->pszValue = NULL;

}

String::String(\_\_in\_opt LPWSTR psz)

{

this->pszValue = NULL;

Set(psz);

}

String::~String()

{

if(NULL != this->pszValue)

{

NoCheckHeapDeleteArray(wcslen(this->pszValue) + 1, this->pszValue);

}

}

///----------------------------------------------------------------------------

///

/// String::Set

///

/// Frees the existing string if any

/// allocates a new buffer to copy the new string

///

///----------------------------------------------------------------------------

void

String::Set(\_\_in\_opt LPWSTR pszValue)

{

if(NULL != this->pszValue)

{

NoCheckHeapDeleteArray(wcslen(this->pszValue) + 1, this->pszValue);

}

if(NULL != pszValue)

{

size\_t size = 1 + wcslen(pszValue);

this->pszValue = NoCheckHeapNewArray(wchar\_t, size);

wcscpy\_s(this->pszValue, size, pszValue);

}

else

{

this->pszValue = NULL;

}

}

template <>

bool RangeUnitContains<SourceFunctionNode>(RangeUnit<SourceFunctionNode> unit, SourceFunctionNode n)

{

Assert(n.functionId != (uint32)-1);

if ((n.sourceContextId >= unit.i.sourceContextId) &&

(n.sourceContextId <= unit.j.sourceContextId)

)

{

if ((n.sourceContextId == unit.j.sourceContextId) && (-2 == unit.j.functionId) || //#.#-#.\* case

(n.sourceContextId == unit.i.sourceContextId) && (-2 == unit.i.functionId) //#.\*-#.# case

)

{

return true;

}

if ((n.sourceContextId == unit.j.sourceContextId) && (-1 == unit.j.functionId) || //#.#-#.+ case

(n.sourceContextId == unit.i.sourceContextId) && (-1 == unit.i.functionId) //#.+-#.# case

)

{

return n.functionId != 0;

}

if (((n.sourceContextId == unit.i.sourceContextId) && (n.functionId < unit.i.functionId)) || //excludes all values less than functionId LHS

((n.sourceContextId == unit.j.sourceContextId) && (n.functionId > unit.j.functionId))) ////excludes all values greater than functionId RHS

{

return false;

}

return true;

}

return false;

}

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class Phases

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

bool

Phases::IsEnabled(Phase phase)

{

return this->phaseList[(int)phase].valid;

}

bool

Phases::IsEnabled(Phase phase, uint soruceContextId, Js::LocalFunctionId functionId)

{

return this->phaseList[(int)phase].valid &&

this->phaseList[(int)phase].range.InRange(SourceFunctionNode(soruceContextId, functionId));

}

bool

Phases::IsEnabledForAll(Phase phase)

{

return this->phaseList[(int)phase].valid &&

this->phaseList[(int)phase].range.ContainsAll();

}

Range \*

Phases::GetRange(Phase phase)

{

return &this->phaseList[(int)phase].range;

}

void

Phases::Enable(Phase phase)

{

this->phaseList[(int)phase].valid = true;

}

Phase

Phases::GetFirstPhase()

{

int i= -1;

while(!this->phaseList[++i].valid)

{

if(i >= PhaseCount - 1)

{

return InvalidPhase;

}

}

return Phase(i);

}

//

// List of names of all the flags

//

const wchar\_t\* const FlagNames[FlagCount + 1] =

{

#define FLAG(type, name, ...) L## #name ,

#include "ConfigFlagsList.h"

NULL

#undef FLAG

};

//

// List of names of all the Phases

//

const wchar\_t\* const PhaseNames[PhaseCount + 1] =

{

#define PHASE(name) L## #name,

#include "ConfigFlagsList.h"

NULL

#undef PHASE

};

//

// Description of flags

//

const wchar\_t\* const FlagDecriptions[FlagCount + 1] =

{

#define FLAG(type, name, description, ...) L##description,

#include "ConfigFlagsList.h"

NULL

#undef FLAG

};

//

// Parent flag categorization of flags

//

const Flag FlagParents[FlagCount + 1] =

{

#define FLAG(type, name, description, defaultValue, parentName, ...) parentName##Flag,

#include "ConfigFlagsList.h"

InvalidFlag

#undef FLAG

};

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class ConfigFlagsTable

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::ConfigFlagsTable

///

/// Constructor initializes all the flags with their default values. The nDummy

/// variable is used to prevent the compiler error due to the trailing comma

/// when we generate the list of flags.

///

///----------------------------------------------------------------------------

#define FLAG(type, name, description, defaultValue, ...) \

\

name ## ( ## defaultValue ##), \

ConfigFlagsTable::ConfigFlagsTable():

#include "ConfigFlagsList.h"

#undef FLAG

nDummy(0)

{

for(int i=0; i < FlagCount; flagPresent[i++] = false);

// set mark for parent flags

ZeroMemory(this->flagIsParent, sizeof(this->flagIsParent));

#define FLAG(type, name, description, defaultValue, parentName, ...) \

if ((int)parentName##Flag < FlagCount) this->flagIsParent[(int) parentName##Flag] = true;

#include "ConfigFlagsList.h"

#undef FLAG

// set all parent flags to their default (setting all child flags to their right values)

this->SetAllParentFlagsAsDefaultValue();

}

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::SetAllParentFlagsAsDefaultValue

///

/// Iterate through all parent flags and set their default value

///

/// Note: only Boolean type supported for now

///----------------------------------------------------------------------------

String \*

ConfigFlagsTable::GetAsString(Flag flag) const

{

return reinterpret\_cast<String\* >(GetProperty(flag));

}

Phases \*

ConfigFlagsTable::GetAsPhase(Flag flag) const

{

return reinterpret\_cast<Phases\*>(GetProperty(flag));

}

Boolean \*

ConfigFlagsTable::GetAsBoolean(Flag flag) const

{

return reinterpret\_cast<Boolean\*>(GetProperty(flag));

}

Number \*

ConfigFlagsTable::GetAsNumber(Flag flag) const

{

return reinterpret\_cast<Number\* >(GetProperty(flag));

}

NumberSet \*

ConfigFlagsTable::GetAsNumberSet(Flag flag) const

{

return reinterpret\_cast<NumberSet\* >(GetProperty(flag));

}

NumberPairSet \*

ConfigFlagsTable::GetAsNumberPairSet(Flag flag) const

{

return reinterpret\_cast<NumberPairSet\* >(GetProperty(flag));

}

NumberRange \*

ConfigFlagsTable::GetAsNumberRange(Flag flag) const

{

return reinterpret\_cast<NumberRange\* >(GetProperty(flag));

}

void

ConfigFlagsTable::Enable(Flag flag)

{

this->flagPresent[flag] = true;

}

void

ConfigFlagsTable::Disable(Flag flag)

{

this->flagPresent[flag] = false;

}

bool

ConfigFlagsTable::IsEnabled(Flag flag)

{

return this->flagPresent[flag];

}

bool

ConfigFlagsTable::IsParentFlag(Flag flag) const

{

return this->flagIsParent[flag];

}

void

ConfigFlagsTable::SetAllParentFlagsAsDefaultValue()

{

for (int i = 0; i < FlagCount; i++)

{

Flag currentFlag = (Flag) i;

if (this->IsParentFlag(currentFlag))

{

// only supporting Boolean for now

AssertMsg(this->GetFlagType(currentFlag) == FlagBoolean, "only supporting boolean flags as parent flags");

Boolean defaultParentValue = this->GetDefaultValueAsBoolean(currentFlag);

this->SetAsBoolean(currentFlag, defaultParentValue);

}

}

}

void ConfigFlagsTable::FinalizeConfiguration()

{

TransferAcronymFlagConfiguration();

TranslateFlagConfiguration();

}

void ConfigFlagsTable::TransferAcronymFlagConfiguration()

{

// Transfer acronym flag configuration into the corresponding actual flag

#define FLAG(...)

#define FLAGNRA(Type, Name, Acronym, ...) \

if(!IsEnabled(Name##Flag) && IsEnabled(Acronym##Flag)) \

{ \

Enable(Name##Flag); \

Name = Acronym; \

}

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

#define FLAGRA(Type, Name, Acronym, ...) FLAGNRA(Type, Name, Acronym, \_\_VA\_ARGS\_\_)

#endif

#include "ConfigFlagsList.h"

}

void ConfigFlagsTable::TranslateFlagConfiguration()

{

const auto VerifyExecutionModeLimits = [this]()

{

const Number zero = static\_cast<Number>(0);

const Number maxUint8 = static\_cast<Number>(static\_cast<uint8>(-1)); // entry point call count is uint8

const Number maxUint16 = static\_cast<Number>(static\_cast<uint16>(-1));

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

Assert(MinInterpretCount >= zero);

Assert(MinInterpretCount <= maxUint16);

Assert(MaxInterpretCount >= zero);

Assert(MaxInterpretCount <= maxUint16);

Assert(MinSimpleJitRunCount >= zero);

Assert(MinSimpleJitRunCount <= maxUint8);

Assert(MaxSimpleJitRunCount >= zero);

Assert(MaxSimpleJitRunCount <= maxUint8);

Assert(SimpleJitAfter >= zero);

Assert(SimpleJitAfter <= maxUint8);

Assert(FullJitAfter >= zero);

Assert(FullJitAfter <= maxUint16);

#endif

Assert(AutoProfilingInterpreter0Limit >= zero);

Assert(AutoProfilingInterpreter0Limit <= maxUint16);

Assert(ProfilingInterpreter0Limit >= zero);

Assert(ProfilingInterpreter0Limit <= maxUint16);

Assert(AutoProfilingInterpreter1Limit >= zero);

Assert(AutoProfilingInterpreter1Limit <= maxUint16);

Assert(SimpleJitLimit >= zero);

Assert(SimpleJitLimit <= maxUint8);

Assert(ProfilingInterpreter1Limit >= zero);

Assert(ProfilingInterpreter1Limit <= maxUint16);

Assert(

(

AutoProfilingInterpreter0Limit +

ProfilingInterpreter0Limit +

AutoProfilingInterpreter1Limit +

SimpleJitLimit +

ProfilingInterpreter1Limit

) <= maxUint16);

};

VerifyExecutionModeLimits();

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

if(ForceDynamicProfile)

{

Force.Enable(DynamicProfilePhase);

}

if(ForceJITLoopBody)

{

Force.Enable(JITLoopBodyPhase);

}

if(NoDeferParse)

{

Off.Enable(DeferParsePhase);

}

bool dontEnforceLimitsForSimpleJitAfterOrFullJitAfter = false;

if((IsEnabled(MinInterpretCountFlag) || IsEnabled(MaxInterpretCountFlag)) &&

!(IsEnabled(SimpleJitAfterFlag) || IsEnabled(FullJitAfterFlag)))

{

if(Off.IsEnabled(SimpleJitPhase))

{

Enable(FullJitAfterFlag);

if(IsEnabled(MaxInterpretCountFlag))

{

FullJitAfter = MaxInterpretCount;

}

else

{

FullJitAfter = MinInterpretCount;

dontEnforceLimitsForSimpleJitAfterOrFullJitAfter = true;

}

}

else

{

Enable(SimpleJitAfterFlag);

if(IsEnabled(MaxInterpretCountFlag))

{

SimpleJitAfter = MaxInterpretCount;

}

else

{

SimpleJitAfter = MinInterpretCount;

dontEnforceLimitsForSimpleJitAfterOrFullJitAfter = true;

}

if(IsEnabled(MinInterpretCountFlag) && IsEnabled(MinSimpleJitRunCountFlag) ||

IsEnabled(MaxSimpleJitRunCountFlag))

{

Enable(FullJitAfterFlag);

FullJitAfter = SimpleJitAfter;

if(IsEnabled(MaxSimpleJitRunCountFlag))

{

FullJitAfter += MaxSimpleJitRunCount;

}

else

{

FullJitAfter += MinSimpleJitRunCount;

Assert(dontEnforceLimitsForSimpleJitAfterOrFullJitAfter);

}

}

}

}

// Configure execution mode limits

do

{

if(IsEnabled(AutoProfilingInterpreter0LimitFlag) ||

IsEnabled(ProfilingInterpreter0LimitFlag) ||

IsEnabled(AutoProfilingInterpreter1LimitFlag) ||

IsEnabled(SimpleJitLimitFlag) ||

IsEnabled(ProfilingInterpreter1LimitFlag))

{

break;

}

if(IsEnabled(ExecutionModeLimitsFlag))

{

uint autoProfilingInterpreter0Limit;

uint profilingInterpreter0Limit;

uint autoProfilingInterpreter1Limit;

uint simpleJitLimit;

uint profilingInterpreter1Limit;

const int scannedCount =

swscanf\_s(

static\_cast<LPCWSTR>(ExecutionModeLimits),

L"%u.%u.%u.%u.%u",

&autoProfilingInterpreter0Limit,

&profilingInterpreter0Limit,

&autoProfilingInterpreter1Limit,

&simpleJitLimit,

&profilingInterpreter1Limit);

Assert(scannedCount == 5);

Enable(AutoProfilingInterpreter0LimitFlag);

Enable(ProfilingInterpreter0LimitFlag);

Enable(AutoProfilingInterpreter1LimitFlag);

Enable(SimpleJitLimitFlag);

Enable(ProfilingInterpreter1LimitFlag);

AutoProfilingInterpreter0Limit = autoProfilingInterpreter0Limit;

ProfilingInterpreter0Limit = profilingInterpreter0Limit;

AutoProfilingInterpreter1Limit = autoProfilingInterpreter1Limit;

SimpleJitLimit = simpleJitLimit;

ProfilingInterpreter1Limit = profilingInterpreter1Limit;

break;

}

if(!NewSimpleJit)

{

// Use the defaults for old simple JIT. The flags are not enabled here because the values can be changed later

// based on other flags, only the defaults values are adjusted here.

AutoProfilingInterpreter0Limit = DEFAULT\_CONFIG\_AutoProfilingInterpreter0Limit;

ProfilingInterpreter0Limit = DEFAULT\_CONFIG\_ProfilingInterpreter0Limit;

CompileAssert(

DEFAULT\_CONFIG\_AutoProfilingInterpreter0Limit <= DEFAULT\_CONFIG\_AutoProfilingInterpreterLimit\_OldSimpleJit);

AutoProfilingInterpreter1Limit =

DEFAULT\_CONFIG\_AutoProfilingInterpreterLimit\_OldSimpleJit - DEFAULT\_CONFIG\_AutoProfilingInterpreter0Limit;

CompileAssert(DEFAULT\_CONFIG\_ProfilingInterpreter0Limit <= DEFAULT\_CONFIG\_SimpleJitLimit\_OldSimpleJit);

SimpleJitLimit = DEFAULT\_CONFIG\_SimpleJitLimit\_OldSimpleJit - DEFAULT\_CONFIG\_ProfilingInterpreter0Limit;

ProfilingInterpreter1Limit = 0;

VerifyExecutionModeLimits();

}

if(IsEnabled(SimpleJitAfterFlag))

{

Enable(AutoProfilingInterpreter0LimitFlag);

Enable(ProfilingInterpreter0LimitFlag);

Enable(AutoProfilingInterpreter1LimitFlag);

Enable(EnforceExecutionModeLimitsFlag);

Js::Number iterationsNeeded = SimpleJitAfter;

ProfilingInterpreter0Limit = min(ProfilingInterpreter0Limit, iterationsNeeded);

iterationsNeeded -= ProfilingInterpreter0Limit;

AutoProfilingInterpreter0Limit = iterationsNeeded;

AutoProfilingInterpreter1Limit = 0;

if(IsEnabled(FullJitAfterFlag))

{

Enable(SimpleJitLimitFlag);

Enable(ProfilingInterpreter1LimitFlag);

Assert(SimpleJitAfter <= FullJitAfter);

Js::Number iterationsNeeded = FullJitAfter - SimpleJitAfter;

Js::Number profilingIterationsNeeded =

min(NewSimpleJit

? DEFAULT\_CONFIG\_MinProfileIterations

: DEFAULT\_CONFIG\_MinProfileIterations\_OldSimpleJit,

FullJitAfter) -

ProfilingInterpreter0Limit;

if(NewSimpleJit)

{

ProfilingInterpreter1Limit = min(ProfilingInterpreter1Limit, iterationsNeeded);

iterationsNeeded -= ProfilingInterpreter1Limit;

profilingIterationsNeeded -= ProfilingInterpreter1Limit;

SimpleJitLimit = iterationsNeeded;

}

else

{

SimpleJitLimit = iterationsNeeded;

profilingIterationsNeeded -= min(SimpleJitLimit, profilingIterationsNeeded);

ProfilingInterpreter1Limit = 0;

}

if(profilingIterationsNeeded != 0)

{

Js::Number iterationsToMove = min(AutoProfilingInterpreter1Limit, profilingIterationsNeeded);

AutoProfilingInterpreter1Limit -= iterationsToMove;

ProfilingInterpreter0Limit += iterationsToMove;

profilingIterationsNeeded -= iterationsToMove;

iterationsToMove = min(AutoProfilingInterpreter0Limit, profilingIterationsNeeded);

AutoProfilingInterpreter0Limit -= iterationsToMove;

ProfilingInterpreter0Limit += iterationsToMove;

profilingIterationsNeeded -= iterationsToMove;

Assert(profilingIterationsNeeded == 0);

}

Assert(

(

AutoProfilingInterpreter0Limit +

ProfilingInterpreter0Limit +

AutoProfilingInterpreter1Limit +

SimpleJitLimit +

ProfilingInterpreter1Limit

) == FullJitAfter);

}

Assert(

(

AutoProfilingInterpreter0Limit +

ProfilingInterpreter0Limit +

AutoProfilingInterpreter1Limit

) == SimpleJitAfter);

EnforceExecutionModeLimits = true;

break;

}

if(IsEnabled(FullJitAfterFlag))

{

Enable(AutoProfilingInterpreter0LimitFlag);

Enable(ProfilingInterpreter0LimitFlag);

Enable(AutoProfilingInterpreter1LimitFlag);

Enable(SimpleJitLimitFlag);

Enable(ProfilingInterpreter1LimitFlag);

Enable(EnforceExecutionModeLimitsFlag);

Js::Number iterationsNeeded = FullJitAfter;

if(NewSimpleJit)

{

ProfilingInterpreter1Limit = min(ProfilingInterpreter1Limit, iterationsNeeded);

iterationsNeeded -= ProfilingInterpreter1Limit;

}

else

{

ProfilingInterpreter1Limit = 0;

SimpleJitLimit = min(SimpleJitLimit, iterationsNeeded);

iterationsNeeded -= SimpleJitLimit;

}

ProfilingInterpreter0Limit = min(ProfilingInterpreter0Limit, iterationsNeeded);

iterationsNeeded -= ProfilingInterpreter0Limit;

if(NewSimpleJit)

{

SimpleJitLimit = min(SimpleJitLimit, iterationsNeeded);

iterationsNeeded -= SimpleJitLimit;

}

AutoProfilingInterpreter0Limit = min(AutoProfilingInterpreter0Limit, iterationsNeeded);

iterationsNeeded -= AutoProfilingInterpreter0Limit;

AutoProfilingInterpreter1Limit = iterationsNeeded;

Assert(

(

AutoProfilingInterpreter0Limit +

ProfilingInterpreter0Limit +

AutoProfilingInterpreter1Limit +

SimpleJitLimit +

ProfilingInterpreter1Limit

) == FullJitAfter);

EnforceExecutionModeLimits = true;

break;

}

if (IsEnabled(MaxTemplatizedJitRunCountFlag))

{

if (MaxTemplatizedJitRunCount >= 0)

{

MinTemplatizedJitRunCount = MaxTemplatizedJitRunCount;

}

}

if (IsEnabled(MaxAsmJsInterpreterRunCountFlag))

{

if (MaxAsmJsInterpreterRunCount >= 0)

{

MinAsmJsInterpreterRunCount = MaxAsmJsInterpreterRunCount;

}

}

} while(false);

#endif

if( (

#ifdef ENABLE\_PREJIT

Prejit ||

#endif

ForceNative

) &&

!NoNative)

{

Enable(AutoProfilingInterpreter0LimitFlag);

Enable(ProfilingInterpreter0LimitFlag);

Enable(AutoProfilingInterpreter1LimitFlag);

Enable(EnforceExecutionModeLimitsFlag);

// Override any relevant automatic configuration above

AutoProfilingInterpreter0Limit = 0;

ProfilingInterpreter0Limit = 0;

AutoProfilingInterpreter1Limit = 0;

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

if(Off.IsEnabled(SimpleJitPhase))

{

Enable(SimpleJitLimitFlag);

Enable(ProfilingInterpreter1LimitFlag);

SimpleJitLimit = 0;

ProfilingInterpreter1Limit = 0;

}

#endif

EnforceExecutionModeLimits = true;

}

VerifyExecutionModeLimits();

}

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::GetFlag

///

/// Given a string finds the corresponding enum Flag. The comparison is case

/// in-sensitive

///

///----------------------------------------------------------------------------

Flag

ConfigFlagsTable::GetFlag(\_\_in LPCWSTR str)

{

for(int i=0; i < FlagCount; i++)

{

if(0 == \_wcsicmp(str, FlagNames[i]))

{

return Flag(i);

}

}

return InvalidFlag;

}

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::GetPhase

///

/// Given a string finds the corresponding enum Phase. The comparison is case

/// in-sensitive

///

///----------------------------------------------------------------------------

Phase

ConfigFlagsTable::GetPhase(\_\_in LPCWSTR str)

{

for(int i=0; i < PhaseCount; i++)

{

if(0 == \_wcsicmp(str, PhaseNames[i]))

{

return Phase(i);

}

}

return InvalidPhase;

}

void

ConfigFlagsTable::PrintUsageString()

{

printf("List of Phases:\n");

for(int i = 0; i < PhaseCount; i++)

{

if (i % 4 == 0)

{

printf("\n ");

}

printf("%-40ls ", PhaseNames[i]);

}

printf("\n\nList of flags:\n\n");

for(int i = 0; i < FlagCount; i++)

{

printf("%60ls ", FlagNames[i]);

switch(GetFlagType(Flag(i)))

{

case InvalidFlagType:

break;

case FlagString:

printf("[:String] ");

break;

case FlagPhases:

printf("[:Phase] ");

break;

case FlagNumber:

printf("[:Number] ");

break;

case FlagBoolean:

printf(" ");

break;

case FlagNumberSet:

printf("[:NumberSet] ");

break;

case FlagNumberPairSet:

printf("[:NumberPairSet] ");

break;

case FlagNumberRange:

printf("[:NumberRange] ");

break;

default:

Assert(false);

\_\_assume(false);

}

printf("%ls\n", FlagDecriptions[i]);

}

}

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::GetFlagType

///

/// Given a flag it returns the type (PhaseFlag, StringFlag ...). This could

/// easily have been a lookup table like FlagNames and PhaseNames but this

/// seems more concise

///

///----------------------------------------------------------------------------

FlagTypes

ConfigFlagsTable::GetFlagType(Flag flag)

{

switch(flag)

{

#define FLAG(type, name, ...) \

case name##Flag : \

return Flag##type; \

#include "ConfigFlagsList.h"

default:

return InvalidFlagType;

}

}

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::GetProperty

///

/// Get the field corresponding to the flag. used as an internal method for

/// the various GetAs\* methods.

///

///----------------------------------------------------------------------------

void \*

ConfigFlagsTable::GetProperty(Flag flag) const

{

switch(flag)

{

#define FLAG(type, name, ...) \

\

case name##Flag : \

return reinterpret\_cast<void\*>(const\_cast<type\*>(&##name)); \

#include "ConfigFlagsList.h"

default:

return NULL;

}

}

void

ConfigFlagsTable::VerboseDump()

{

#define FLAG(type, name, ...) \

if (IsEnabled(name##Flag)) \

{ \

Output::Print(L"-%s", L#name); \

switch (Flag##type) \

{ \

case FlagBoolean: \

if (!\*GetAsBoolean(name##Flag)) \

{ \

Output::Print(L"-"); \

} \

break; \

case FlagString: \

if (GetAsString(name##Flag) != nullptr) \

{ \

Output::Print(L":%s", \*GetAsString(name##Flag)); \

} \

break; \

case FlagNumber: \

Output::Print(L":%d", \*GetAsNumber(name##Flag)); \

break; \

}; \

Output::Print(L"\n"); \

}

#include "ConfigFlagsList.h"

#undef FLAG

}

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::GetDefaultValueAsBoolean

///

/// Get the default value of a given boolean flag. If the flag is not of boolean

/// type, will assert on CHK or return FALSE on FRE.

///

///----------------------------------------------------------------------------

Boolean

ConfigFlagsTable::GetDefaultValueAsBoolean(Flag flag) const

{

Boolean retValue = FALSE;

switch (flag)

{

#define FLAG(type, name, description, defaultValue, ...) FLAGDEFAULT##type(name, defaultValue)

// define an overload for each FlagTypes - type

// \* all defaults we don't care about

#define FLAGDEFAULTPhases(name, defaultValue)

#define FLAGDEFAULTString(name, defaultValue)

#define FLAGDEFAULTNumber(name, defaultValue)

#define FLAGDEFAULTNumberSet(name, defaultValue)

#define FLAGDEFAULTNumberRange(name, defaultValue)

#define FLAGDEFAULTNumberPairSet(name, defaultValue)

// \* and those we do care about

#define FLAGDEFAULTBoolean(name, defaultValue) \

case name##Flag: \

retValue = (Boolean) defaultValue; \

break; \

#include "ConfigFlagsList.h"

#undef FLAGDEFAULTBoolean

#undef FLAGDEFAULTNumberRange

#undef FLAGDEFAULTNumberPairSet

#undef FLAGDEFAULTNumberSet

#undef FLAGDEFAULTNumber

#undef FLAGDEFAULTString

#undef FLAGDEFAULTPhases

#undef FLAG

#undef FLAGREGOVREXPBoolean

#undef FLAGREGOVREXPNumberRange

#undef FLAGREGOVREXPNumberPairSet

#undef FLAGREGOVREXPNumberSet

#undef FLAGREGOVREXPNumber

#undef FLAGREGOVREXPString

#undef FLAGREGOVREXPPhases

#undef FLAGREGOVRBoolean

#undef FLAGREGOVRNumberRange

#undef FLAGREGOVRNumberPairSet

#undef FLAGREGOVRNumberSet

#undef FLAGREGOVRNumber

#undef FLAGREGOVRString

#undef FLAGREGOVRPhases

default:

// not found - or not a boolean flag

Assert(false);

}

return retValue;

}

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::SetAsBoolean

///

/// Set the value of a boolean flag. If the flag is a parent flag, all children flag

// will be set accordingly.

///

///----------------------------------------------------------------------------

void

ConfigFlagsTable::SetAsBoolean(Flag flag, Boolean value)

{

AssertMsg(this->GetFlagType(flag) == FlagBoolean, "flag not a boolean type");

Boolean\* settingAsBoolean = this->GetAsBoolean(flag);

Assert(settingAsBoolean != nullptr);

Output::VerboseNote(L"FLAG %s = %d\n", FlagNames[(int) flag], value);

\*settingAsBoolean = value;

// check if parent flag

if (this->IsParentFlag(flag))

{

// parent flag, will iterate through all child flags

Flag childFlag = GetNextChildFlag(flag, /\* no currentChildFlag \*/ InvalidFlag);

while (childFlag != InvalidFlag)

{

Boolean childDefaultValue = GetDefaultValueAsBoolean(childFlag);

// if the parent flag is TRUE, the children flag values are based on their default values

// if the parent flag is FALSE, the children flag values are FALSE (always - as disabled)

Boolean childValue = value == TRUE ? childDefaultValue : FALSE;

Output::VerboseNote(L"FLAG %s = %d - setting child flag %s = %d\n", FlagNames[(int) flag], value, FlagNames[(int) childFlag], childValue);

this->SetAsBoolean(childFlag, childValue);

// get next child flag

childFlag = GetNextChildFlag(flag, /\* currentChildFlag \*/ childFlag);

}

}

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

// in case the flag is marked as 'callback' - to call the method

#define FLAG(type, name, description, defaultValue, parentName, hasCallback) FLAGCALLBACK##hasCallback(type, name)

#define FLAGCALLBACKFALSE(type, name)

#define FLAGCALLBACKTRUE(type, name) FLAGDOCALLBACK##type(name)

// define an overload for each FlagTypes - type

// \* all defaults we don't care about - should assert

#define FLAGDOCALLBACKNumberRange(name) Assert(false);

#define FLAGDOCALLBACKPhases(name) Assert(false);

#define FLAGDOCALLBACKString(name) Assert(false);

#define FLAGDOCALLBACKNumber(name) Assert(false);

#define FLAGDOCALLBACKNumberSet(name) Assert(false);

#define FLAGDOCALLBACKNumberPairSet(name) Assert(false);

// \* and those we do care about

#define FLAGDOCALLBACKBoolean(name) if( flag == name##Flag ) this->FlagSetCallback\_##name(value);

#include "ConfigFlagsList.h"

#undef FLAGDOCALLBACKBoolean

#undef FLAGDOCALLBACKNumberRange

#undef FLAGDOCALLBACKNumberPairSet

#undef FLAGDOCALLBACKNumberSet

#undef FLAGDOCALLBACKNumber

#undef FLAGDOCALLBACKString

#undef FLAGDOCALLBACKPhases

#undef FLAGCALLBACKTRUE

#undef FLAGCALLBACKFALSE

#undef FLAG

#endif

}

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::GetParentFlag

///

/// Get the parent flag corresponding to the flag, if any, otherwise returns NoParentFlag

///

///----------------------------------------------------------------------------

Flag

ConfigFlagsTable::GetParentFlag(Flag flag) const

{

Flag parentFlag = FlagParents[(int)flag];

return parentFlag;

}

///----------------------------------------------------------------------------

///

/// ConfigFlagsTable::GetNextChildFlag

///

/// Get the next child flag for a given parent flag. If no currentChildFlag, use

/// InvalidFlag or NoParentFlag as start iterator.

///

///----------------------------------------------------------------------------

Flag

ConfigFlagsTable::GetNextChildFlag(Flag parentFlag, Flag currentChildFlag) const

{

// start at the current+1

int startIndex = (int)currentChildFlag + 1;

// otherwise start from beginning

if (currentChildFlag == InvalidFlag || currentChildFlag == NoParentFlag)

{

// reset the start index

startIndex = 0;

}

for(int i=startIndex; i < FlagCount; i++)

{

Flag currentFlag = (Flag)i;

Flag parentFlagForCurrentFlag = GetParentFlag(currentFlag);

if(parentFlagForCurrentFlag == parentFlag)

{

// found a match

return currentFlag;

}

}

// no more

return InvalidFlag;

}

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

//

// Special overrides for flags being set

//

void

ConfigFlagsTable::FlagSetCallback\_ES6All(Boolean value)

{

// iterate through all ES6 flags - and set them explicitly (except ES6Verbose)

Flag parentFlag = ES6Flag;

// parent ES6 flag, will iterate through all child ES6 flags

Flag childFlag = GetNextChildFlag(parentFlag, /\* no currentChildFlag \*/ InvalidFlag);

while (childFlag != InvalidFlag)

{

// skip verbose

if (childFlag != ES6VerboseFlag)

{

Boolean childValue = value;

Output::VerboseNote(L"FLAG %s = %d - setting child flag %s = %d\n", FlagNames[(int) parentFlag], value, FlagNames[(int) childFlag], childValue);

this->SetAsBoolean(childFlag, childValue);

}

// get next child flag

childFlag = GetNextChildFlag(parentFlag, /\* currentChildFlag \*/ childFlag);

}

}

void

ConfigFlagsTable::FlagSetCallback\_ES6Experimental(Boolean value)

{

if (value)

{

EnableExperimentalFlag();

}

}

#endif

void

ConfigFlagsTable::EnableExperimentalFlag()

{

AutoCriticalSection autocs(&csExperimentalFlags);

#define FLAG\_REGOVR\_EXP(type, name, description, defaultValue, parentName, hasCallback) this->SetAsBoolean(Js::Flag::name##Flag, true);

#include "ConfigFlagsList.h"

#undef FLAG\_REGOVR\_EXP

}

//

// Configuration options

//

Configuration::Configuration() : isHybridDebugging(false)

{

if(IsDebuggerPresent())

{

if(IsEqualGUID(hybridDebuggingGuid, HybridDebuggingGuid))

{

isHybridDebugging = true;

}

}

}

bool Configuration::EnableJitInDebugMode()

{

return (!IsHybridDebugging() || CONFIG\_FLAG(EnableJitInHybridDebugging)) && CONFIG\_FLAG(EnableJitInDiagMode);

}

bool Configuration::IsHybridDebugging()

{

return isHybridDebugging;

}

Configuration Configuration::Global;

} //namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// enum FlagTypes

///

/// Different types of flags supported by the debug table

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

enum FlagTypes

{

InvalidFlagType,

FlagString,

FlagPhases,

FlagNumber,

FlagBoolean,

FlagNumberSet,

FlagNumberPairSet,

FlagNumberRange

};

// Shadow of enum SCRIPTHOSTTYPE defined in activscp.idl.

enum HostType

{

HostTypeDefault = 0, // Used to detect engines with uninitialized host type.

HostTypeBrowser = 1, // Currently this implies enabled legacy language features, use it for IE.

HostTypeApplication = 2, // Currently this implies legacy-free language features, use it for WWA.

HostTypeWebview = 3, // Webview in an WWA/XAML app with WinRT access.

HostTypeMin = HostTypeBrowser,

HostTypeMax = HostTypeWebview

};

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// enum Flag

///

/// All the flags available. The list of flags are described in the file

/// ConfigFlagsList.h

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

enum Flag

{

#define FLAG(type, name, ...) name##Flag,

#include "ConfigFlagsList.h"

FlagCount,

InvalidFlag,

NoParentFlag,

FlagMax = NoParentFlag

} ;

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// enum Phase

///

/// Different Phases/Subphases of the backend. The list of phases is described

/// in the file ConfigFlagsList.h

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

enum Phase

{

#define PHASE(name) name##Phase,

#include "ConfigFlagsList.h"

PhaseCount,

InvalidPhase

#undef PHASE

};

extern const wchar\_t\* const FlagNames[FlagCount + 1];

extern const wchar\_t\* const PhaseNames[PhaseCount + 1];

extern const Flag FlagParents[FlagCount + 1];

typedef int Number;

typedef bool Boolean;

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class String

///

/// A simple string wrapper for storing strings for the flags. It creates a

/// copy of the string which it releases in the destructor. Simple operator

/// fiddling to make this class as interoperable with strings as possible :-

///

/// 1. Single argument constructor takes care of initialization

/// 2. Assignment operator overloaded for wchar\_t\*

/// 3. JavascriptConversion to const wchar\_t \* is defined. Making this constant ensures

/// that a non constant wchar\_t \* does not point to our buffer which can

/// potentially corrupt it.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

class String

{

PREVENT\_COPYCONSTRUCT(String)

// Data

private:

LPWSTR pszValue;

// Construction

public:

inline String();

inline String(\_\_in\_opt LPWSTR psz);

inline ~String();

// Methods

public:

///----------------------------------------------------------------------------

///

/// Assignment operator defined for easy access

///

///----------------------------------------------------------------------------

String& operator=(\_\_in\_opt LPWSTR psz)

{

Set(psz);

return \*this;

}

String &operator =(const String &other)

{

return (\*this = other.pszValue);

}

///----------------------------------------------------------------------------

///

/// Typecast operator defined so we can directly use it with strings

///

///----------------------------------------------------------------------------

operator LPCWSTR () const

{

return this->pszValue;

}

// Implementation

private:

void Set(\_\_in\_opt LPWSTR pszValue);

};

class NumberSet

{

public:

NumberSet();

void Add(uint32 x);

bool Contains(uint32 x);

bool Empty() const { return set.Count() == 0; }

private:

JsUtil::BaseHashSet<uint32, NoCheckHeapAllocator, PrimeSizePolicy> set;

};

class NumberPair

{

public:

NumberPair(uint32 x, uint32 y) : x(x), y(y) {}

NumberPair() : x((uint32)-1), y((uint32)-1) {}

operator hash\_t() const { return (x << 16) + y; }

bool operator ==(const NumberPair &other) const { return x == other.x && y == other.y; }

private:

uint32 x;

uint32 y;

};

class NumberPairSet

{

public:

NumberPairSet();

void Add(uint32 x, uint32 y);

bool Contains(uint32 x, uint32 y);

bool Empty() const { return set.Count() == 0; }

private:

JsUtil::BaseHashSet<NumberPair, NoCheckHeapAllocator, PrimeSizePolicy> set;

};

struct SourceFunctionNode

{

uint sourceContextId;

Js::LocalFunctionId functionId;

SourceFunctionNode(uint sourceContextId, Js::LocalFunctionId functionId) : sourceContextId(sourceContextId), functionId(functionId) {}

SourceFunctionNode() : sourceContextId(0), functionId(0){}

};

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// struct RangeUnit

///

/// Used to store a pair of bounds

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

template <typename TRangeUnitData>

struct RangeUnit

{

TRangeUnitData i, j;

RangeUnit() {}

RangeUnit(TRangeUnitData x, TRangeUnitData y) :

i(x),

j(y)

{}

};

template <typename TRangeUnitData>

bool RangeUnitContains(RangeUnit<TRangeUnitData> unit, TRangeUnitData item)

{

return (item >= unit.i && item <= unit.j);

}

template <>

bool RangeUnitContains<SourceFunctionNode>(RangeUnit<SourceFunctionNode> unit, SourceFunctionNode n);

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class Range

///

/// Data structure to store a range of integers. (Can trivially be templatized

/// but resisting the temptation for the sake of simplicity.)

///

/// Sample: 1,3-11,15,20

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

template <typename TRangeUnitData>

class RangeBase

{

// Nested Classes

public:

typedef RangeUnit<TRangeUnitData> Unit;

// Data

protected:

SList<Unit, NoCheckHeapAllocator> range;

// Construction

public:

RangeBase() : range(&NoCheckHeapAllocator::Instance) {}

// Methods

public:

inline bool InRange(TRangeUnitData i);

inline bool ContainsAll();

inline void Add(TRangeUnitData i);

inline void Add(TRangeUnitData i, TRangeUnitData j);

#if DBG\_RANGE

template <typename TFunction>

void Map(TFunction fn) const

{

range.Map(fn);

}

#endif

};

class Range : public RangeBase<SourceFunctionNode> {};

class NumberRange : public RangeBase<int> {};

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class Phases

///

/// Maintains the list of all the phases. Each individual phase can be enabled

/// and an optional range of numbers can be added to it

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

class Phases

{

// Nested Types

private:

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// struct UnitPhase

///

/// Stores information about a particular phase. Current information is :

/// 1. Is the phase enabled/disabled?

/// 2. A range of numbers for the given phase. This range can represent a

/// variety of information. For example, this range can be the function

/// numbers of all the functions where a particular optimization is

/// enabled or where opcode dump is enabled.

///

///

/// Its a tiny private struct used by Phases for simply storing data. So

/// making this a struct with all members public.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

struct UnitPhase

{

// Construction

public:

UnitPhase() : valid(false) { }

// Data

public:

bool valid;

Range range;

};

// Data

private:

UnitPhase phaseList[PhaseCount];

// Constructors

public:

Phases() {}

// Methods

public:

void Enable(Phase phase);

bool IsEnabled(Phase phase);

bool IsEnabled(Phase phase, uint sourceContextId, Js::LocalFunctionId functionId);

bool IsEnabledForAll(Phase phase);

Range \* GetRange(Phase phase);

Phase GetFirstPhase();

};

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class ConfigFlagsTable

///

/// Maintains the list of all the flags. Individual flags can be accessed by :-

///

/// 1. flagTable->verbose; // here verbose is the name of the flag

/// // It will be strongly typed with the

/// // type of the flag is as described in

/// // ConfigFlagsTable.h

///

/// 2. flagTable->GetAsBoolean(verboseFlag);

/// // this access is weekly typed. Even if the

/// // flag is not of type Boolean, it will be

/// // reinterpret\_cast to Boolean \*

///

/// The former access should be used in the code for reading the flag values

/// while the later is useful for adding values into the table.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

class ConfigFlagsTable

{

// Constructor

public:

ConfigFlagsTable();

// Methods

public:

static Flag GetFlag(\_\_in LPCWSTR str);

static Phase GetPhase(\_\_in LPCWSTR str);

static void PrintUsageString();

static FlagTypes GetFlagType(Flag flag);

String\* GetAsString(Flag flag) const;

Phases\* GetAsPhase(Flag flag) const;

Boolean\* GetAsBoolean(Flag flag) const;

Number\* GetAsNumber(Flag flag) const;

NumberSet\* GetAsNumberSet(Flag flag) const;

NumberPairSet \* GetAsNumberPairSet(Flag flag) const;

NumberRange \* GetAsNumberRange(Flag flag) const;

void SetAsBoolean(Flag flag, Boolean value);

Boolean GetDefaultValueAsBoolean(Flag flag) const;

// indicates whether a flag is a parent flag

bool IsParentFlag(Flag flag) const;

// get the parent flag of a given flag, if any, otherwise returns InvalidFlag.

Flag GetParentFlag(Flag flag) const;

// get the next child flag of a given parent flag, if any, otherwise returns InvalidFlag. Pass InvalidFlag as currentChildFlag if no current child flag to iterate from.

Flag GetNextChildFlag(Flag parentFlag, Flag currentChildFlag) const;

void Enable(Flag flag);

bool IsEnabled(Flag flag);

void Disable(Flag flag);

void VerboseDump();

// Data

public:

///----------------------------------------------------------------------------

///

/// Declaration of each flag as a member variable of the corresponding type.

/// These variables are made public because get/set are too ugly for simple

/// flags. Besides there are flags like phase lists which are not simple and

/// its better to have a uniform way to access them.

///

/// if(ConfigFlag->dump) is much better than

/// if(ConfigFlag->GetDump())

///

///----------------------------------------------------------------------------

#define FLAG(type, name, ...) \

\

type name;\

#include "ConfigFlagsList.h"

bool flagPresent[FlagCount];

private:

//

// This variable is defined for the constructor

//

int nDummy;

// indicates whether a flag is a parent flag

bool flagIsParent[FlagMax + 1];

// Implementation

private:

void \* GetProperty(Flag flag) const;

void SetAllParentFlagsAsDefaultValue();

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

// special callback logic

void FlagSetCallback\_ES6All(Boolean value);

void FlagSetCallback\_ES6Experimental(Boolean value);

#endif

public:

void FinalizeConfiguration();

void EnableExperimentalFlag();

// Some config flags are expected to be constant during the lifetime of

// a ScriptContext object. However, some other flags can change,

// possibly multiple times.

//

// To keep the flags constant during the lifetime of a ScriptContext

// object, we copy the ones that it needs into ThreadContext and have

// it fetch them from ThreadContext instead of here. Given that a

// ScriptContext object is bound to a ThreadContext object and never

// gets reassigned, this keeps the flags constant while the

// ScriptContext object is alive.

//

// Currently, among the flags used by ScriptContext, only the

// experimental flags are altered after initialization. Therefore, only

// access to these flags are serialized.

//

// Note that this lock is acquired automatically only when the

// experimental flags are initialized via the EnableExperimentalFlag()

// method. It should be manually acquired anywhere else where these

// flags are accessed.

CriticalSection csExperimentalFlags;

private:

void TransferAcronymFlagConfiguration();

void TranslateFlagConfiguration();

};

class Profiler;

class Configuration

{

public:

ConfigFlagsTable flags;

static Configuration Global;

bool EnableJitInDebugMode();

bool IsHybridDebugging();

// Public in case the client wants to have

// a separate config from the global one

Configuration();

private:

GUID hybridDebuggingGuid; // Set to HybridDebuggingGuid when hybrid debugging - set by the out of process debugging component - jscript9diag.

// Otherwise, remains un-initialized

bool isHybridDebugging;

};

//Create macros for a useful subset of the config options that either get the value from the configuration (if the option is enabled) or

//just use the hard coded default value (if not). All the ...IsEnabled(...) default to false.

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

#define CONFIG\_ISENABLED(flag) (Js::Configuration::Global.flags.IsEnabled(flag))

#define CUSTOM\_CONFIG\_ISENABLED(flags, flag) (flags.IsEnabled(flag))

#define CONFIG\_FLAG(flag) (Js::Configuration::Global.flags.##flag##)

#define CUSTOM\_CONFIG\_FLAG(flags, flag) (flags.##flag##)

#define CONFIG\_FLAG\_RELEASE(flag) CONFIG\_FLAG(flag)

#define CONFIG\_FLAG\_CONTAINS(flag, func) (Js::Configuration::Global.flags.##flag##.Contains((func)->GetLocalFunctionId()))

#define PHASE\_OFF\_PROFILED\_BYTE\_CODE(phase, func) Js::Configuration::Global.flags.OffProfiledByteCode.IsEnabled((phase),(func)->GetSourceContextId(),(func)->GetLocalFunctionId())

#define PHASE\_OFF\_PROFILED\_BYTE\_CODE\_ALL(phase) Js::Configuration::Global.flags.OffProfiledByteCode.IsEnabledForAll((phase))

#define PHASE\_OFF\_PROFILED\_BYTE\_CODE\_OPTFUNC(phase, func) ((func) ? PHASE\_OFF\_PROFILED\_BYTE\_CODE(phase, func) : PHASE\_OFF\_PROFILED\_BYTE\_CODE\_ALL(phase))

#define PHASE\_OFF1(phase) Js::Configuration::Global.flags.Off.IsEnabled((phase))

#define CUSTOM\_PHASE\_OFF1(flags, phase) flags.Off.IsEnabled((phase))

#define PHASE\_OFF\_ALL(phase) Js::Configuration::Global.flags.Off.IsEnabledForAll((phase))

#define PHASE\_OFF(phase, func) PHASE\_OFF\_RAW((phase), (func)->GetSourceContextId(), (func)->GetLocalFunctionId())

#define PHASE\_OFF\_OPTFUNC(phase, func) ((func) ? PHASE\_OFF(phase, func) : PHASE\_OFF\_ALL(phase))

#define PHASE\_OFF\_RAW(phase, sourceId, functionId) \

Js::Configuration::Global.flags.Off.IsEnabled((phase), (sourceId), (functionId))

#define PHASE\_ON1(phase) Js::Configuration::Global.flags.On.IsEnabled((phase))

#define CUSTOM\_PHASE\_ON1(flags, phase) flags.On.IsEnabled((phase))

#define PHASE\_ON(phase, func) PHASE\_ON\_RAW((phase), (func)->GetSourceContextId(), (func)->GetLocalFunctionId())

#define PHASE\_ON\_RAW(phase, sourceId, functionId) \

Js::Configuration::Global.flags.On.IsEnabled((phase), (sourceId), (functionId))

#define PHASE\_FORCE1(phase) Js::Configuration::Global.flags.Force.IsEnabled((phase))

#define CUSTOM\_PHASE\_FORCE1(flags, phase) flags.Force.IsEnabled((phase))

#define PHASE\_FORCE(phase, func) PHASE\_FORCE\_RAW((phase), (func)->GetSourceContextId(), (func)->GetLocalFunctionId())

#define PHASE\_FORCE\_OPTFUNC(phase, func) ((func) ? PHASE\_FORCE(phase, func) : PHASE\_FORCE1(phase))

#define PHASE\_FORCE\_RAW(phase, sourceId, functionId) \

Js::Configuration::Global.flags.Force.IsEnabled((phase), (sourceId), (functionId))

#define PHASE\_STRESS1(phase) Js::Configuration::Global.flags.Stress.IsEnabled((phase))

#define PHASE\_STRESS(phase, func) PHASE\_STRESS\_RAW((phase), (func)->GetSourceContextId(), (func)->GetLocalFunctionId())

#define PHASE\_STRESS\_RAW(phase, sourceId, functionId) \

Js::Configuration::Global.flags.Stress.IsEnabled((phase), (sourceId), (functionId))

#define PHASE\_TRACE1(phase) Js::Configuration::Global.flags.Trace.IsEnabled((phase))

#define CUSTOM\_PHASE\_TRACE1(flags, phase) flags.Trace.IsEnabled((phase))

#define PHASE\_TRACE(phase, func) PHASE\_TRACE\_RAW((phase), (func)->GetSourceContextId(), (func)->GetLocalFunctionId())

#define PHASE\_TRACE\_RAW(phase, sourceId, functionId) \

Js::Configuration::Global.flags.Trace.IsEnabled((phase), (sourceId), (functionId))

#if DBG

// Enabling the ability to trace the StringConcat phase only in debug builds due to performance impact

#define PHASE\_TRACE\_StringConcat PHASE\_TRACE1(Js::StringConcatPhase)

#else

#define PHASE\_TRACE\_StringConcat (false)

#endif

#define PHASE\_VERBOSE\_TRACE1(phase) \

((PHASE\_TRACE1((phase))) && Js::Configuration::Global.flags.Verbose)

#define CUSTOM\_PHASE\_VERBOSE\_TRACE1(flags, phase) \

((CUSTOM\_PHASE\_TRACE1((flags), (phase))) && flags.Verbose)

#define PHASE\_VERBOSE\_TRACE(phase, func) \

((PHASE\_TRACE((phase), (func))) && Js::Configuration::Global.flags.Verbose)

#define PHASE\_VERBOSE\_TRACE\_RAW(phase, sourceId, functionId) \

((PHASE\_TRACE\_RAW((phase), (sourceId), (functionId))) && Js::Configuration::Global.flags.Verbose)

#define PHASE\_DUMP(phase, func) Js::Configuration::Global.flags.Dump.IsEnabled((phase), (func)->GetSourceContextId(),(func)->GetLocalFunctionId())

#define PHASE\_STATS1(phase) Js::Configuration::Global.flags.Stats.IsEnabled((phase))

#define CUSTOM\_PHASE\_STATS1(flags, phase) flags.Stats.IsEnabled((phase))

#define PHASE\_VERBOSE\_STATS1(phase) \

((PHASE\_STATS1(phase)) && Js::Configuration::Global.flags.Verbose)

#define PHASE\_STATS\_ALL(phase) Js::Configuration::Global.flags.Stats.IsEnabledForAll((phase))

#define PHASE\_STATS(phase, func) PHASE\_STATS\_RAW((phase), (func)->GetSourceContextId(), (func)->GetLocalFunctionId())

#define PHASE\_STATS\_RAW(phase, sourceId, functionId) \

Js::Configuration::Global.flags.Stats.IsEnabled((phase), (sourceId), (functionId))

#define PHASE\_VERBOSE\_STATS(phase, func) \

((PHASE\_STATS(phase, func)) && Js::Configuration::Global.flags.Verbose)

#define PHASE\_VERBOSE\_STATS\_RAW(phase, sourceId, functionId) \

((PHASE\_STATS\_RAW(phase, sourceId, functionId)) && Js::Configuration::Global.flags.Verbose)

#define PHASE\_TESTTRACE1(phase) Js::Configuration::Global.flags.TestTrace.IsEnabled((phase))

#define PHASE\_TESTTRACE(phase, func) PHASE\_TESTTRACE\_RAW((phase), (func)->GetSourceContextId(), (func)->GetLocalFunctionId())

#define PHASE\_TESTTRACE\_RAW(phase, sourceId, functionId) \

Js::Configuration::Global.flags.TestTrace.IsEnabled((phase), (sourceId), (functionId))

#ifdef ENABLE\_BASIC\_TELEMETRY

#define PHASE\_TESTTRACE1\_TELEMETRY(phase) PHASE\_TESTTRACE1((phase))

#else

#define PHASE\_TESTTRACE1\_TELEMETRY(phase) (false)

#endif

#define PHASE\_PRINT\_TRACE1(phase, ...) \

if (PHASE\_TRACE1(phase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define CUSTOM\_PHASE\_PRINT\_TRACE1(flags, phase, ...) \

if (CUSTOM\_PHASE\_TRACE1(flags, phase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_TRACE(phase, func, ...) \

if (PHASE\_TRACE(phase, func)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_TRACE\_RAW(phase, sourceId, functionId, ...) \

if (PHASE\_TRACE\_RAW(phase, sourceId, functionId)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_VERBOSE\_TRACE1(phase, ...) \

if (PHASE\_VERBOSE\_TRACE1(phase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define CUSTOM\_PHASE\_PRINT\_VERBOSE\_TRACE1(flags, phase, ...) \

if (CUSTOM\_PHASE\_VERBOSE\_TRACE1(flags, phase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_VERBOSE\_TRACE(phase, func, ...) \

if (PHASE\_VERBOSE\_TRACE(phase, func)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_VERBOSE\_TRACE\_RAW(phase, sourceId, functionId, ...) \

if (PHASE\_VERBOSE\_TRACE\_RAW(phase, sourceId, functionId)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_VERBOSE\_TESTTRACE1(phase) (PHASE\_TESTTRACE1(phase) && Js::Configuration::Global.flags.Verbose)

#define PHASE\_VERBOSE\_TESTTRACE(phase, func) \

(PHASE\_TESTTRACE(phase, func) && Js::Configuration::Global.flags.Verbose)

#define PHASE\_VERBOSE\_TESTTRACE\_RAW(phase, sourceId, functionId) \

(PHASE\_TESTTRACE\_RAW(phase, sourceId, functionId) && Js::Configuration::Global.flags.Verbose)

#define PHASE\_PRINT\_TESTTRACE1(phase, ...) \

if (PHASE\_TESTTRACE1(phase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_TESTTRACE(phase, func, ...) \

if (PHASE\_TESTTRACE(phase, func)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_TESTTRACE\_RAW(phase, sourceId, functionId, ...) \

if (PHASE\_TESTTRACE\_RAW(phase, sourceId, functionId)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_VERBOSE\_TESTTRACE1(phase, ...) \

if (PHASE\_VERBOSE\_TESTTRACE1(phase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_VERBOSE\_TESTTRACE(phase, func, ...) \

if (PHASE\_VERBOSE\_TESTTRACE(phase, func)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#define PHASE\_PRINT\_VERBOSE\_TESTTRACE\_RAW(phase, sourceId, functionId, ...) \

if (PHASE\_VERBOSE\_TESTTRACE\_RAW(phase, sourceId, functionId)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

#else

#define CONFIG\_ISENABLED(flag) (false) //All flags.IsEnabled(foo) are false by default.

#define CUSTOM\_CONFIG\_ISENABLED(flags, flag) (false) //All flags.IsEnabled(foo) are false by default.

#define CONFIG\_FLAG(flag) (DEFAULT\_CONFIG\_##flag##)

#define CUSTOM\_CONFIG\_FLAG(flags, flag) (DEFAULT\_CONFIG\_##flag##)

#define CONFIG\_FLAG\_RELEASE(flag) (Js::Configuration::Global.flags.##flag##)

#define CONFIG\_FLAG\_CONTAINS(flag, func) (false)

#define PHASE\_OFF\_PROFILED\_BYTE\_CODE(phase, func) (false)

#define PHASE\_OFF\_PROFILED\_BYTE\_CODE\_ALL(phase) (false)

#define PHASE\_OFF\_PROFILED\_BYTE\_CODE\_OPTFUNC(phase, func) (false)

#define PHASE\_OFF1(phase) (false) //All flags.Off.IsEnabled(foo) are false by default

#define CUSTOM\_PHASE\_OFF1(flags, phase) (false) //All flags.Off.IsEnabled(foo) are false by default

#define PHASE\_OFF\_ALL(phase) (false)

#define PHASE\_OFF(phase, func) (false)

#define PHASE\_OFF\_RAW(phase, sourceId, functionId) (false)

#define PHASE\_OFF\_OPTFUNC(phase, func) (false)

#define PHASE\_ON1(phase) (false)

#define CUSTOM\_PHASE\_ON1(flags, phase) (false)

#define PHASE\_ON(phase, func) (false)

#define PHASE\_ON\_RAW(phase, sourceId, funcionId) (false)

#define PHASE\_FORCE1(phase) (false)

#define CUSTOM\_PHASE\_FORCE1(flags, phase) (false)

#define PHASE\_FORCE(phase, func) (false)

#define PHASE\_FORCE\_RAW(phase, sourceId, functionId) (false)

#define PHASE\_FORCE\_OPTFUNC(phase, func) (false)

#define PHASE\_STRESS1(phase) (false)

#define PHASE\_STRESS(phase, func) (false)

#define PHASE\_STRESS\_RAW(phase, sourceId, functionId) (false)

#define PHASE\_TRACE1(phase) (false)

#define CUSTOM\_PHASE\_TRACE1(phase) (false)

#define PHASE\_TRACE(phase, func) (false)

#define PHASE\_TRACE\_RAW(phase, sourceId, functionId) (false)

#define PHASE\_TRACE\_StringConcat (false)

#define PHASE\_VERBOSE\_TRACE1(phase) (false)

#define CUSTOM\_PHASE\_VERBOSE\_TRACE1(flags, phase) (false)

#define PHASE\_VERBOSE\_TRACE(phase, func) (false)

#define PHASE\_VERBOSE\_TRACE\_RAW(phase, sourceId, functionId) (false)

#define PHASE\_TESTTRACE1(phase) (false)

#define PHASE\_TESTTRACE(phase, func) (false)

#define PHASE\_TESTTRACE\_RAW(phase, sourceId, functionId) (false)

#define PHASE\_TESTTRACE1\_TELEMETRY(phase) (false)

#define PHASE\_PRINT\_TRACE1(phase, ...)

#define CUSTOM\_PHASE\_PRINT\_TRACE1(phase, ...)

#define PHASE\_PRINT\_TRACE(phase, func, ...)

#define PHASE\_PRINT\_TRACE\_RAW(phase, sourceId, functionId, ...)

#define PHASE\_PRINT\_VERBOSE\_TRACE1(phase, ...)

#define CUSTOM\_PHASE\_PRINT\_VERBOSE\_TRACE1(phase, ...)

#define PHASE\_PRINT\_VERBOSE\_TRACE(phase, func, ...)

#define PHASE\_PRINT\_VERBOSE\_TRACE\_RAW(phase, sourceId, functionId, ...)

#define PHASE\_VERBOSE\_TESTTRACE1(phase) (false)

#define PHASE\_VERBOSE\_TESTTRACE(phase, func) (false)

#define PHASE\_VERBOSE\_TESTTRACE\_RAW(phase, sourceId, functionId) (false)

#define PHASE\_VERBOSE\_TRACE1(phase) (false)

#define PHASE\_VERBOSE\_TRACE(phase, func) (false)

#define PHASE\_VERBOSE\_TRACE\_RAW(phase, sourceId, functionId) (false)

#define PHASE\_PRINT\_TESTTRACE1(phase, ...)

#define PHASE\_PRINT\_TESTTRACE(phase, func, ...)

#define PHASE\_PRINT\_TESTTRACE\_RAW(phase, sourceId, functionId, ...)

#define PHASE\_PRINT\_VERBOSE\_TESTTRACE1(phase, ...)

#define PHASE\_PRINT\_VERBOSE\_TESTTRACE(phase, func, ...)

#define PHASE\_PRINT\_VERBOSE\_TESTTRACE\_RAW(phase, sourceId, functionId, ...)

#define PHASE\_DUMP(phase, func) (false) //All flags.Dump.IsEnabled(foo) are false by default

#define PHASE\_STATS1(phase) (false)

#define CUSTOM\_PHASE\_STATS1(flags, phase) (false)

#define PHASE\_VERBOSE\_STATS1(phase) (false)

#define PHASE\_STATS\_ALL(phase) (false)

#define PHASE\_STATS(phase, func) (false)

#define PHASE\_STATS\_RAW(phase, sourceId, functionId) (false)

#define PHASE\_VERBOSE\_STATS(phase, func) (false)

#define PHASE\_VERBOSE\_STATS\_RAW(phase, sourceId, functionId) (false)

#endif

#ifdef ENABLE\_REGEX\_CONFIG\_OPTIONS

#define REGEX\_CONFIG\_FLAG(flag) (Js::Configuration::Global.flags.##flag##)

#else

#define REGEX\_CONFIG\_FLAG(flag) (DEFAULT\_CONFIG\_##flag##)

#endif

#ifdef SUPPORT\_INTRUSIVE\_TESTTRACES

#define PHASE\_PRINT\_INTRUSIVE\_TESTTRACE1(phase, ...) \

PHASE\_PRINT\_TESTTRACE1(phase, \_\_VA\_ARGS\_\_)

#else

#define PHASE\_PRINT\_INTRUSIVE\_TESTTRACE1(phase, ...) (false)

#endif

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class RangeBase

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

template <typename TRangeUnitData>

void

RangeBase<TRangeUnitData>::Add(TRangeUnitData i)

{

Add(i, i);

}

template <typename TRangeUnitData>

void

RangeBase<TRangeUnitData>::Add(TRangeUnitData i, TRangeUnitData j)

{

range.Prepend(RangeUnit<TRangeUnitData>(i, j));

}

template <typename TRangeUnitData>

bool

RangeBase<TRangeUnitData>::ContainsAll()

{

return range.Empty();

}

///----------------------------------------------------------------------------

///

/// RangeBase::InRange

///

/// Searches for each element in the list of UnitRanges. If the given integer

/// is between the 2 values, then return true; If no element is present in range

/// then, then we return true

///

///----------------------------------------------------------------------------

template <typename TRangeUnitData>

bool RangeBase<TRangeUnitData>::InRange(TRangeUnitData n)

{

if (range.Empty())

{

return true;

}

else

{

return range.MapUntil([n](RangeUnit<TRangeUnitData> const& unit)

{

return RangeUnitContains(unit, n);

});

}

}

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#include <io.h>

#include <fcntl.h>

#include <share.h>

#include <strsafe.h>

#include "Memory\MemoryLogger.h"

#include "Memory\ForcedMemoryConstraints.h"

#include "core\ICustomConfigFlags.h"

#include "core\CmdParser.h"

#include "core\ConfigParser.h"

ConfigParser ConfigParser::s\_moduleConfigParser(Js::Configuration::Global.flags);

#ifdef ENABLE\_TRACE

class ArenaHost

{

AllocationPolicyManager m\_allocationPolicyManager;

PageAllocator m\_pageAllocator;

ArenaAllocator m\_allocator;

public:

ArenaHost(\_\_in\_z wchar\_t\* arenaName) :

m\_allocationPolicyManager(/\* needConcurrencySupport = \*/ true),

m\_pageAllocator(&m\_allocationPolicyManager, Js::Configuration::Global.flags),

m\_allocator(arenaName, &m\_pageAllocator, Js::Throw::OutOfMemory)

{

}

ArenaAllocator\* GetAllocator() { return &m\_allocator; }

};

ArenaAllocator\* GetOutputAllocator1()

{

static ArenaHost s\_arenaHost(L"For Output::Trace (1)");

return s\_arenaHost.GetAllocator();

}

ArenaAllocator\* GetOutputAllocator2()

{

static ArenaHost s\_arenaHost(L"For Output::Trace (2)");

return s\_arenaHost.GetAllocator();

}

#endif

void ConfigParser::ParseOnModuleLoad(CmdLineArgsParser& parser, HANDLE hmod)

{

Assert(!s\_moduleConfigParser.HasReadConfig());

s\_moduleConfigParser.ParseRegistry(parser);

s\_moduleConfigParser.ParseConfig(hmod, parser);

s\_moduleConfigParser.ProcessConfiguration(hmod);

// 'parser' destructor post-processes some configuration

}

void ConfigParser::ParseRegistry(CmdLineArgsParser &parser)

{

HKEY hk;

bool includeUserHive = true;

if (NOERROR == RegOpenKeyExW(HKEY\_LOCAL\_MACHINE, JsUtil::ExternalApi::GetFeatureKeyName(), 0, KEY\_READ, &hk))

{

DWORD dwValue;

DWORD dwSize = sizeof(dwValue);

ParseRegistryKey(hk, parser);

// HKLM can prevent user config from being read.

if (NOERROR == RegGetValueW(hk, nullptr, L"AllowUserConfig", RRF\_RT\_DWORD, nullptr, (LPBYTE)&dwValue, &dwSize) && dwValue == 0)

{

includeUserHive = false;

}

RegCloseKey(hk);

}

if (includeUserHive && NOERROR == RegOpenKeyExW(HKEY\_CURRENT\_USER, JsUtil::ExternalApi::GetFeatureKeyName(), 0, KEY\_READ, &hk))

{

ParseRegistryKey(hk, parser);

RegCloseKey(hk);

}

}

void ConfigParser::ParseRegistryKey(HKEY hk, CmdLineArgsParser &parser)

{

DWORD dwSize;

DWORD dwValue;

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

wchar\_t regBuffer[MaxRegSize];

dwSize = sizeof(regBuffer);

if (NOERROR == RegGetValueW(hk, nullptr, L"JScript9", RRF\_RT\_REG\_SZ, nullptr, (LPBYTE)regBuffer, &dwSize))

{

LPWSTR regValue = regBuffer, nextValue = nullptr;

regValue = wcstok\_s(regBuffer, L" ", &nextValue);

while (regValue != nullptr)

{

int err = 0;

if ((err = parser.Parse(regValue)) != 0)

{

break;

}

regValue = wcstok\_s(nullptr, L" ", &nextValue);

}

}

#endif

// MemSpect - This setting controls whether MemSpect instrumentation is enabled.

// The value is treated as a bit field with the following bits:

// 0x01 - Track Arena memory

// 0x02 - Track Recycler memory

// 0x04 - Track Page allocations

dwValue = 0;

dwSize = sizeof(dwValue);

if (NOERROR == ::RegGetValueW(hk, nullptr, L"MemSpect", RRF\_RT\_DWORD, nullptr, (LPBYTE)&dwValue, &dwSize))

{

if (dwValue & 0x01)

{

ArenaMemoryTracking::Activate();

}

if (dwValue & 0x02)

{

RecyclerMemoryTracking::Activate();

}

if (dwValue & 0x04)

{

PageTracking::Activate();

}

}

// JScriptJIT - This setting controls the JIT/interpretation of Jscript code.

// The legal values are as follows:

// 1- Force JIT code to be generated for everything.

// 2- Force interpretation without profiling (turn off JIT)

// 3- Default

// 4- Interpreter, simple JIT, and full JIT run a predetermined number of times. Requires >= 3 calls to functions.

// 5- Interpreter, simple JIT, and full JIT run a predetermined number of times. Requires >= 4 calls to functions.

// 6- Force interpretation with profiling

//

// This reg key is present in released builds. The QA team's tests use these switches to

// get reliable JIT coverage in servicing runs done by IE/Windows. Because this reg key is

// released, the number of possible values is limited to reduce surface area.

dwValue = 0;

dwSize = sizeof(dwValue);

if (NOERROR == RegGetValueW(hk, nullptr, L"JScriptJIT", RRF\_RT\_DWORD, nullptr, (LPBYTE)&dwValue, &dwSize))

{

Js::ConfigFlagsTable &configFlags = Js::Configuration::Global.flags;

switch (dwValue)

{

case 1:

configFlags.Enable(Js::ForceNativeFlag);

configFlags.ForceNative = true;

break;

case 6:

configFlags.Enable(Js::ForceDynamicProfileFlag);

configFlags.ForceDynamicProfile = true;

// fall through

case 2:

configFlags.Enable(Js::NoNativeFlag);

configFlags.NoNative = true;

break;

case 3:

break;

case 4:

configFlags.Enable(Js::AutoProfilingInterpreter0LimitFlag);

configFlags.Enable(Js::ProfilingInterpreter0LimitFlag);

configFlags.Enable(Js::AutoProfilingInterpreter1LimitFlag);

configFlags.Enable(Js::SimpleJitLimitFlag);

configFlags.Enable(Js::ProfilingInterpreter1LimitFlag);

configFlags.Enable(Js::EnforceExecutionModeLimitsFlag);

configFlags.AutoProfilingInterpreter0Limit = 0;

configFlags.AutoProfilingInterpreter1Limit = 0;

if (

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

configFlags.NewSimpleJit

#else

DEFAULT\_CONFIG\_NewSimpleJit

#endif

)

{

configFlags.ProfilingInterpreter0Limit = 0;

configFlags.SimpleJitLimit = 0;

configFlags.ProfilingInterpreter1Limit = 2;

}

else

{

configFlags.ProfilingInterpreter0Limit = 1;

configFlags.SimpleJitLimit = 1;

configFlags.ProfilingInterpreter1Limit = 0;

}

configFlags.EnforceExecutionModeLimits = true;

break;

case 5:

configFlags.Enable(Js::AutoProfilingInterpreter0LimitFlag);

configFlags.Enable(Js::ProfilingInterpreter0LimitFlag);

configFlags.Enable(Js::AutoProfilingInterpreter1LimitFlag);

configFlags.Enable(Js::SimpleJitLimitFlag);

configFlags.Enable(Js::ProfilingInterpreter1LimitFlag);

configFlags.Enable(Js::EnforceExecutionModeLimitsFlag);

configFlags.AutoProfilingInterpreter0Limit = 0;

configFlags.ProfilingInterpreter0Limit = 0;

configFlags.AutoProfilingInterpreter1Limit = 1;

if (

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

configFlags.NewSimpleJit

#else

DEFAULT\_CONFIG\_NewSimpleJit

#endif

)

{

configFlags.SimpleJitLimit = 0;

configFlags.ProfilingInterpreter1Limit = 2;

}

else

{

configFlags.SimpleJitLimit = 2;

configFlags.ProfilingInterpreter1Limit = 0;

}

configFlags.EnforceExecutionModeLimits = true;

break;

}

}

// EnumerationCompat

// This setting allows disabling a couple of changes to enumeration:

// - A change that causes deleted property indexes to be reused for new properties, thereby changing the order in which

// properties are enumerated

// - A change that creates a true snapshot of the type just before enumeration, and enumerating only those properties. A

// property that was deleted before enumeration and is added back during enumeration will not be enumerated.

// Values:

// 0 - Default

// 1 - Compatibility mode for enumeration order (disable changes described above)

// This FCK does not apply to WWAs. WWAs should use the RC compat mode to disable these changes.

dwValue = 0;

dwSize = sizeof(dwValue);

if (NOERROR == RegGetValueW(hk, nullptr, L"EnumerationCompat", RRF\_RT\_DWORD, nullptr, (LPBYTE)&dwValue, &dwSize))

{

if(dwValue == 1)

{

Js::Configuration::Global.flags.EnumerationCompat = true;

}

}

#ifdef ENABLE\_PROJECTION

// FailFastIfDisconnectedDelegate

// This setting allows enabling fail fast if the delegate invoked is disconnected

// 0 - Default return the error RPC\_E\_DISCONNECTED if disconnected delegate is invoked

// 1 - Fail fast if disconnected delegate

dwValue = 0;

dwSize = sizeof(dwValue);

if (NOERROR == RegGetValueW(hk, nullptr, L"FailFastIfDisconnectedDelegate", RRF\_RT\_DWORD, nullptr, (LPBYTE)&dwValue, &dwSize))

{

if(dwValue == 1)

{

Js::Configuration::Global.flags.FailFastIfDisconnectedDelegate = true;

}

}

#endif

// ES6 feature control

// This setting allows enabling\disabling es6 features

// 0 - Enable ES6 flag - Also default behavior

// 1 - Disable ES6 flag

dwValue = 0;

dwSize = sizeof(dwValue);

if (NOERROR == RegGetValueW(hk, nullptr, L"DisableES6", RRF\_RT\_DWORD, nullptr, (LPBYTE)&dwValue, &dwSize))

{

Js::ConfigFlagsTable &configFlags = Js::Configuration::Global.flags;

if (dwValue == 1)

{

configFlags.Enable(Js::ES6Flag);

configFlags.SetAsBoolean(Js::ES6Flag, false);

}

}

// Asmjs feature control

// This setting allows enabling\disabling asmjs compilation

// 0 - Disable Asmjs phase - Also default behavior

// 1 - Enable Asmjs phase

dwValue = 0;

dwSize = sizeof(dwValue);

if (NOERROR == RegGetValueW(hk, nullptr, L"EnableAsmjs", RRF\_RT\_DWORD, nullptr, (LPBYTE)&dwValue, &dwSize))

{

if (dwValue == 1)

{

Js::Configuration::Global.flags.Asmjs = true;

}

}

}

void ConfigParser::ParseConfig(HANDLE hmod, CmdLineArgsParser &parser)

{

#if defined(ENABLE\_DEBUG\_CONFIG\_OPTIONS) || defined(PARSE\_CONFIG\_FILE)

Assert(!\_hasReadConfig);

\_hasReadConfig = true;

wchar\_t configBuffer[MaxTokenSize];

int err = 0;

wchar\_t modulename[\_MAX\_PATH];

wchar\_t filename[\_MAX\_PATH];

GetModuleFileName((HMODULE)hmod, modulename, \_MAX\_PATH);

wchar\_t drive[\_MAX\_DRIVE];

wchar\_t dir[\_MAX\_DIR];

\_wsplitpath\_s(modulename, drive, \_MAX\_DRIVE, dir, \_MAX\_DIR, nullptr, 0, nullptr, 0);

\_wmakepath\_s(filename, drive, dir, \_configFileName, L".config");

FILE\* configFile;

if (\_wfopen\_s(&configFile, filename, L"r, ccs=UNICODE") != 0 || configFile == nullptr)

{

WCHAR configFileFullName[MAX\_PATH];

StringCchPrintf(configFileFullName, MAX\_PATH, L"%s.config", \_configFileName);

// try the one in the current working directory (Desktop)

if (\_wfullpath(filename, configFileFullName, \_MAX\_PATH) == nullptr)

{

return;

}

if (\_wfopen\_s(&configFile, filename, L"r, ccs=UNICODE") != 0 || configFile == nullptr)

{

return;

}

}

while (fwscanf\_s(configFile, L"%s", configBuffer, MaxTokenSize) != FINISHED)

{

if ((err = parser.Parse(configBuffer)) != 0)

{

break;

}

}

fclose(configFile);

if (err !=0)

{

return;

}

#endif

}

void ConfigParser::ProcessConfiguration(HANDLE hmod)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

bool hasOutput = false;

wchar\_t modulename[\_MAX\_PATH];

GetModuleFileName((HMODULE)hmod, modulename, \_MAX\_PATH);

if (Js::Configuration::Global.flags.Console)

{

int fd;

FILE \*fp;

// fail usually means there is an existing console. We don't really care.

AllocConsole();

fd = \_open\_osfhandle((intptr\_t)GetStdHandle(STD\_OUTPUT\_HANDLE), O\_TEXT);

fp = \_wfdopen(fd, L"w");

\*stdout = \*fp;

setvbuf(stdout, nullptr, \_IONBF, 0);

fd = \_open\_osfhandle((intptr\_t)GetStdHandle(STD\_ERROR\_HANDLE), O\_TEXT);

fp = \_wfdopen(fd, L"w");

\*stderr = \*fp;

setvbuf(stderr, nullptr, \_IONBF, 0);

wchar\_t buffer[\_MAX\_PATH + 70];

if (ConfigParserAPI::FillConsoleTitle(buffer, \_MAX\_PATH + 20, modulename))

{

SetConsoleTitle(buffer);

}

hasOutput = true;

}

if (Js::Configuration::Global.flags.IsEnabled(Js::OutputFileFlag)

&& Js::Configuration::Global.flags.OutputFile != nullptr)

{

SetOutputFile(Js::Configuration::Global.flags.OutputFile, Js::Configuration::Global.flags.OutputFileOpenMode);

hasOutput = true;

}

if (Js::Configuration::Global.flags.DebugWindow)

{

Output::UseDebuggerWindow();

hasOutput = true;

}

#ifdef ENABLE\_TRACE

if (CONFIG\_FLAG(InMemoryTrace))

{

Output::SetInMemoryLogger(

Js::MemoryLogger::Create(::GetOutputAllocator1(),

CONFIG\_FLAG(InMemoryTraceBufferSize) \* 3)); // With stack each trace is 3 entries (header, msg, stack).

hasOutput = true;

}

#ifdef STACK\_BACK\_TRACE

if (CONFIG\_FLAG(TraceWithStack))

{

Output::SetStackTraceHelper(Js::StackTraceHelper::Create(::GetOutputAllocator2()));

}

#endif STACK\_BACK\_TRACE

#endif ENABLE\_TRACE

if (hasOutput)

{

ConfigParserAPI::DisplayInitialOutput(modulename);

Output::Print(L"\n");

Js::Configuration::Global.flags.VerboseDump();

Output::Flush();

}

if (Js::Configuration::Global.flags.ForceSerialized)

{

// Can't generate or execute byte code under forced serialize

Js::Configuration::Global.flags.GenerateByteCodeBufferReturnsCantGenerate = true;

Js::Configuration::Global.flags.ExecuteByteCodeBufferReturnsInvalidByteCode = true;

}

ForcedMemoryConstraint::Apply();

#endif

#ifdef MEMSPECT\_TRACKING

bool all = false;

if (Js::Configuration::Global.flags.Memspect.IsEnabled(Js::AllPhase))

{

all = true;

}

if (all || Js::Configuration::Global.flags.Memspect.IsEnabled(Js::RecyclerPhase))

{

RecyclerMemoryTracking::Activate();

}

if (all || Js::Configuration::Global.flags.Memspect.IsEnabled(Js::PageAllocatorPhase))

{

PageTracking::Activate();

}

if (all || Js::Configuration::Global.flags.Memspect.IsEnabled(Js::ArenaPhase))

{

ArenaMemoryTracking::Activate();

}

#endif

}

HRESULT ConfigParser::SetOutputFile(const WCHAR\* outputFile, const WCHAR\* openMode)

{

// If present, replace the {PID} token with the process ID

const WCHAR\* pidStr = nullptr;

WCHAR buffer[\_MAX\_PATH];

if ((pidStr = wcsstr(outputFile, L"{PID}")) != nullptr)

{

size\_t pidStartPosition = pidStr - outputFile;

WCHAR\* pDest = buffer;

size\_t bufferLen = \_MAX\_PATH;

// Copy the filename before the {PID} token

wcsncpy\_s(pDest, bufferLen, outputFile, pidStartPosition);

pDest += pidStartPosition;

bufferLen = bufferLen - pidStartPosition;

// Copy the PID

\_ultow\_s(GetCurrentProcessId(), pDest, /\*bufferSize=\*/\_MAX\_PATH - pidStartPosition, /\*radix=\*/10);

#pragma prefast(suppress: 26014, "ultow string length is smaller than 256")

pDest += wcslen(pDest);

bufferLen = bufferLen - wcslen(pDest);

// Copy the rest of the string.

#pragma prefast(suppress: 26014, "Overwriting pDset's null terminator is intentional since the string being copied is null terminated")

wcscpy\_s(pDest, bufferLen, outputFile + pidStartPosition + /\*length of {PID}\*/ 5);

outputFile = buffer;

}

wchar\_t fileName[\_MAX\_PATH];

wchar\_t moduleName[\_MAX\_PATH];

GetModuleFileName(0, moduleName, \_MAX\_PATH);

\_wsplitpath\_s(moduleName, nullptr, 0, nullptr, 0, fileName, \_MAX\_PATH, nullptr, 0);

if (\_wcsicmp(fileName, L"WWAHost") == 0 || \_wcsicmp(fileName, L"ByteCodeGenerator") == 0 ||

\_wcsicmp(fileName, L"spartan") == 0 || \_wcsicmp(fileName, L"spartan\_edge") == 0 ||

\_wcsicmp(fileName, L"MicrosoftEdge") == 0 || \_wcsicmp(fileName, L"MicrosoftEdgeCP") == 0)

{

// we need to output to %temp% directory in wwa. we don't have permission otherwise.

if (GetEnvironmentVariable(L"temp", fileName, \_MAX\_PATH) != 0)

{

wcscat\_s(fileName, \_MAX\_PATH, L"\\");

const wchar\_t \* fileNameOnly = wcsrchr(outputFile, L'\\');

// if outputFile is full path we just need filename, discard the path

wcscat\_s(fileName, \_MAX\_PATH, fileNameOnly == nullptr ? outputFile : fileNameOnly);

}

else

{

AssertMsg(FALSE, "Get temp environment failed");

}

outputFile = fileName;

}

FILE \*fp;

if ((fp = \_wfsopen(outputFile, openMode, \_SH\_DENYWR)) != nullptr)

{

Output::SetOutputFile(fp);

return S\_OK;

}

AssertMsg(false, "Could not open file for logging output.");

return E\_FAIL;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// API for methods that the ConfigParser can call

// Implement these methods to customize what happens when the configuration is loaded

class ConfigParserAPI

{

public:

// If the -Console flag is passed in, this method is called with a buffer

// The implementor will fill in the buffer and return true if a custom Console window title is desired

// To not change the console title, return false

static bool FillConsoleTitle(\_\_ecount(cchBufferSize) LPWSTR buffer, size\_t cchBufferSize, \_\_in LPWSTR moduleName);

// If one of the following flags:

// - Console

// - OutputFile

// - DebugWindow

// - InMemoryTrace

// is set, then the ConfigParser will call the following method to give the implementor

// a chance to output any headers at initialization time.

static void DisplayInitialOutput(\_\_in LPWSTR moduleName);

};

class ConfigParser

{

private:

static const int MaxTokenSize = 512;

static const int MaxRegSize = 2048;

static const LPWSTR featureKeyName ;

bool \_hasReadConfig;

Js::ConfigFlagsTable& \_flags;

const LPCWSTR \_configFileName;

// NT version of CRT has the "backward compat" behavior that returns 0 instead of EOF

// for unicode version of fwscanf.

#ifdef NTBUILD

static const int FINISHED = 0;

#else

static const int FINISHED = EOF;

#endif

void ParseRegistryKey(HKEY hk, CmdLineArgsParser &parser);

public:

static ConfigParser s\_moduleConfigParser;

ConfigParser(Js::ConfigFlagsTable& outputFlags, \_\_in LPCWSTR configFileName = L"jscript") :

\_flags(outputFlags),

\_hasReadConfig(false),

\_configFileName(configFileName)

{

}

static void ParseOnModuleLoad(CmdLineArgsParser& parser, HANDLE hMod);

void ParseConfig(HANDLE hmod, CmdLineArgsParser &parser);

void ParseRegistry(CmdLineArgsParser &parser);

void ProcessConfiguration(HANDLE mod);

HRESULT SetOutputFile(const WCHAR\* outputFile, const WCHAR\* openMode);

bool HasReadConfig() { return \_hasReadConfig; }

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

class CriticalSection

{

public:

CriticalSection(DWORD spincount = 0)

{

#pragma prefast(suppress:6031, "InitializeCriticalSectionAndSpinCount always succeed since Vista. No need to check return value");

::InitializeCriticalSectionAndSpinCount(&cs, spincount);

}

~CriticalSection() { ::DeleteCriticalSection(&cs); }

BOOL TryEnter() { return ::TryEnterCriticalSection(&cs); }

void Enter() { ::EnterCriticalSection(&cs); }

void Leave() { ::LeaveCriticalSection(&cs); }

#if DBG

bool IsLocked() const { return cs.OwningThread == (HANDLE)::GetCurrentThreadId(); }

#endif

private:

CRITICAL\_SECTION cs;

};

//FakeCriticalSection mimics CriticalSection apis

class FakeCriticalSection

{

public:

FakeCriticalSection(DWORD spincount = 0) { /\*do nothing\*/spincount++; }

~FakeCriticalSection() {}

BOOL TryEnter() { return true; }

void Enter() {}

void Leave() {}

#if DBG

bool IsLocked() const { return true; }

#endif

};

class AutoCriticalSection

{

public:

AutoCriticalSection(CriticalSection \* cs) : cs(cs) { cs->Enter(); }

~AutoCriticalSection() { cs->Leave(); }

private:

CriticalSection \* cs;

};

class AutoOptionalCriticalSection

{

public:

AutoOptionalCriticalSection(CriticalSection \* cs) : cs(cs)

{

if (cs)

{

cs->Enter();

}

}

~AutoOptionalCriticalSection()

{

if (cs)

{

cs->Leave();

}

}

private:

CriticalSection \* cs;

};

template <class SyncObject = FakeCriticalSection >

class AutoRealOrFakeCriticalSection

{

public:

AutoRealOrFakeCriticalSection(SyncObject \* cs) : cs(cs) { cs->Enter(); }

~AutoRealOrFakeCriticalSection() { cs->Leave(); }

private:

SyncObject \* cs;

};

template <class SyncObject = FakeCriticalSection >

class AutoOptionalRealOrFakeCriticalSection

{

public:

AutoOptionalRealOrFakeCriticalSection(SyncObject \* cs) : cs(cs)

{

if (cs)

{

cs->Enter();

}

}

~AutoOptionalRealOrFakeCriticalSection()

{

if (cs)

{

cs->Leave();

}

}

private:

SyncObject \* cs;

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#ifdef DBGHELP\_SYMBOL\_MANAGER

#include "core\DbgHelpSymbolManager.h"

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAP")

DbgHelpSymbolManager DbgHelpSymbolManager::Instance;

void

DbgHelpSymbolManager::Initialize()

{

wchar\_t \*wszSearchPath = nullptr;

wchar\_t \*wszModuleDrive = nullptr;

wchar\_t \*wszModuleDir = nullptr;

wchar\_t \*wszOldSearchPath = nullptr;

wchar\_t \*wszNewSearchPath = nullptr;

wchar\_t \*wszModuleName = nullptr;

const size\_t ceModuleName = \_MAX\_PATH;

const size\_t ceOldSearchPath = 32767;

const size\_t ceNewSearchPath = ceOldSearchPath + \_MAX\_PATH + 1;

if (isInitialized)

{

return;

}

AutoCriticalSection autocs(&cs);

if (isInitialized)

{

goto end;

}

isInitialized = true;

hProcess = GetCurrentProcess();

// Let's make sure the directory where chakra.dll is, is on the symbol path.

wchar\_t const \* wszModule = AutoSystemInfo::GetJscriptDllFileName();

wszModuleName = NoCheckHeapNewArray(wchar\_t, ceModuleName);

if (wszModuleName == nullptr)

{

goto end;

}

if (wcscmp(wszModule, L"") == 0)

{

if (GetModuleFileName(NULL, wszModuleName, static\_cast<DWORD>(ceModuleName)))

{

wszModule = wszModuleName;

}

else

{

wszModule = nullptr;

}

}

if (wszModule != nullptr)

{

wszModuleDrive = NoCheckHeapNewArray(wchar\_t, \_MAX\_DRIVE);

if (wszModuleDrive == nullptr)

{

goto end;

}

wszModuleDir = NoCheckHeapNewArray(wchar\_t, \_MAX\_DIR);

if (wszModuleDir == nullptr)

{

goto end;

}

\_wsplitpath\_s(wszModule, wszModuleDrive, \_MAX\_DRIVE, wszModuleDir, \_MAX\_DIR, NULL, 0, NULL, 0);

\_wmakepath\_s(wszModuleName, ceModuleName, wszModuleDrive, wszModuleDir, NULL, NULL);

wszOldSearchPath = NoCheckHeapNewArray(wchar\_t, ceOldSearchPath);

if (wszOldSearchPath == nullptr)

{

goto end;

}

wszNewSearchPath = NoCheckHeapNewArray(wchar\_t, ceNewSearchPath);

if (wszNewSearchPath == nullptr)

{

goto end;

}

if (GetEnvironmentVariable(L"\_NT\_SYMBOL\_PATH", wszOldSearchPath, static\_cast<DWORD>(ceOldSearchPath)) != 0)

{

swprintf\_s(wszNewSearchPath, ceNewSearchPath, L"%s;%s", wszOldSearchPath, wszModuleName);

wszSearchPath = wszNewSearchPath;

}

else

{

wszSearchPath = wszModuleName;

}

}

hDbgHelpModule = LoadLibraryEx(L"dbghelp.dll", NULL, LOAD\_LIBRARY\_SEARCH\_SYSTEM32);

if (hDbgHelpModule == nullptr)

{

goto end;

}

typedef BOOL(\_\_stdcall \*PfnSymInitialize)(HANDLE, PCWSTR, BOOL);

PfnSymInitialize pfnSymInitialize = (PfnSymInitialize)GetProcAddress(hDbgHelpModule, "SymInitializeW");

if (pfnSymInitialize)

{

pfnSymInitialize(hProcess, wszSearchPath, TRUE);

pfnSymFromAddrW = (PfnSymFromAddrW)GetProcAddress(hDbgHelpModule, "SymFromAddrW");

pfnSymGetLineFromAddr64W = (PfnSymGetLineFromAddr64W)GetProcAddress(hDbgHelpModule, "SymGetLineFromAddrW64");

// load line information

typedef DWORD(\_\_stdcall \*PfnSymGetOptions)();

typedef VOID(\_\_stdcall \*PfnSymSetOptions)(DWORD);

PfnSymGetOptions pfnSymGetOptions = (PfnSymGetOptions)GetProcAddress(hDbgHelpModule, "SymGetOptions");

PfnSymSetOptions pfnSymSetOptions = (PfnSymSetOptions)GetProcAddress(hDbgHelpModule, "SymSetOptions");

DWORD options = pfnSymGetOptions();

options |= SYMOPT\_LOAD\_LINES;

pfnSymSetOptions(options);

}

end:

if (wszModuleName != nullptr)

{

NoCheckHeapDeleteArray(ceModuleName, wszModuleName);

wszModuleName = nullptr;

}

if (wszModuleDrive != nullptr)

{

NoCheckHeapDeleteArray(\_MAX\_DRIVE, wszModuleDrive);

wszModuleDrive = nullptr;

}

if (wszModuleDir != nullptr)

{

NoCheckHeapDeleteArray(\_MAX\_DIR, wszModuleDir);

wszModuleDir = nullptr;

}

if (wszOldSearchPath != nullptr)

{

NoCheckHeapDeleteArray(ceOldSearchPath, wszOldSearchPath);

wszOldSearchPath = nullptr;

}

if (wszNewSearchPath != nullptr)

{

NoCheckHeapDeleteArray(ceNewSearchPath, wszNewSearchPath);

wszNewSearchPath = nullptr;

}

}

DbgHelpSymbolManager::~DbgHelpSymbolManager()

{

if (hDbgHelpModule)

{

typedef BOOL(\_\_stdcall \*PfnSymCleanup)(HANDLE);

PfnSymCleanup pfnSymCleanup = (PfnSymCleanup)GetProcAddress(hDbgHelpModule, "SymCleanup");

if (pfnSymCleanup)

{

pfnSymCleanup(hProcess);

}

FreeLibrary(hDbgHelpModule);

}

}

BOOL

DbgHelpSymbolManager::SymFromAddr(PVOID address, DWORD64 \* dwDisplacement, PSYMBOL\_INFO pSymbol)

{

if (Instance.pfnSymFromAddrW)

{

return Instance.pfnSymFromAddrW(Instance.hProcess, (DWORD64)address, dwDisplacement, pSymbol);

}

return FALSE;

}

BOOL

DbgHelpSymbolManager::SymGetLineFromAddr64(\_In\_ PVOID address, \_Out\_ PDWORD pdwDisplacement, \_Out\_ PIMAGEHLP\_LINEW64 pLine)

{

if (pdwDisplacement != nullptr)

{

\*pdwDisplacement = 0;

}

if (pLine != nullptr)

{

ZeroMemory(pLine, sizeof(IMAGEHLP\_LINEW64));

pLine->SizeOfStruct = sizeof(IMAGEHLP\_LINE64);

}

if (Instance.pfnSymGetLineFromAddr64W)

{

return Instance.pfnSymGetLineFromAddr64W(Instance.hProcess, (DWORD64)address, pdwDisplacement, pLine);

}

return FALSE;

}

size\_t DbgHelpSymbolManager::PrintSymbol(PVOID address)

{

size\_t retValue = 0;

DWORD64 dwDisplacement = 0;

char buffer[sizeof(SYMBOL\_INFO)+MAX\_SYM\_NAME \* sizeof(TCHAR)];

PSYMBOL\_INFO pSymbol = (PSYMBOL\_INFO)buffer;

pSymbol->SizeOfStruct = sizeof(SYMBOL\_INFO);

pSymbol->MaxNameLen = MAX\_SYM\_NAME;

IMAGEHLP\_LINE64 lineInfo;

lineInfo.SizeOfStruct = sizeof(IMAGEHLP\_LINE64);

if (DbgHelpSymbolManager::SymFromAddr(address, &dwDisplacement, pSymbol))

{

DWORD dwDisplacementDWord = static\_cast<DWORD>(dwDisplacement);

if (DbgHelpSymbolManager::SymGetLineFromAddr64(address, &dwDisplacementDWord, &lineInfo))

{

retValue += Output::Print(L"0x%p %s+0x%llx (%s:%d)", address, pSymbol->Name, dwDisplacement, lineInfo.FileName, lineInfo.LineNumber);

}

else

{

// SymGetLineFromAddr64 failed

retValue += Output::Print(L"0x%p %s+0x%llx", address, pSymbol->Name, dwDisplacement);

}

}

else

{

// SymFromAddr failed

retValue += Output::Print(L"0x%p", address);

}

return retValue;

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#ifdef DBGHELP\_SYMBOL\_MANAGER

#define DBGHELP\_TRANSLATE\_TCHAR

// dbghelp.h is not clean with warning 4091

#pragma warning(push)

#pragma warning(disable: 4091) /\* warning C4091: 'typedef ': ignored on left of '' when no variable is declared \*/

#include <dbghelp.h>

#pragma warning(pop)

class DbgHelpSymbolManager

{

public:

static void EnsureInitialized() { Instance.Initialize(); }

static BOOL SymFromAddr(PVOID address, DWORD64 \* dwDisplacement, PSYMBOL\_INFO pSymbol);

static BOOL SymGetLineFromAddr64(\_In\_ PVOID address, \_Out\_ PDWORD pdwDisplacement, \_Out\_ PIMAGEHLP\_LINEW64 pLine);

static size\_t PrintSymbol(PVOID address);

private:

DbgHelpSymbolManager() : isInitialized(false), hDbgHelpModule(nullptr), pfnSymFromAddrW(nullptr) {}

~DbgHelpSymbolManager();

static DbgHelpSymbolManager Instance;

void Initialize();

bool isInitialized;

CriticalSection cs;

HANDLE hProcess;

HMODULE hDbgHelpModule;

typedef BOOL(\_\_stdcall \*PfnSymFromAddrW)(HANDLE, DWORD64, PDWORD64, PSYMBOL\_INFOW);

PfnSymFromAddrW pfnSymFromAddrW;

typedef BOOL(\_\_stdcall \*PfnSymGetLineFromAddr64W)(HANDLE, DWORD64, PDWORD, PIMAGEHLP\_LINEW64);

PfnSymGetLineFromAddr64W pfnSymGetLineFromAddr64W;

};

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#include "core\DelayLoadLibrary.h"

DelayLoadLibrary::DelayLoadLibrary()

{

m\_hModule = nullptr;

m\_isInit = false;

}

DelayLoadLibrary::~DelayLoadLibrary()

{

if (m\_hModule)

{

FreeLibrary(m\_hModule);

m\_hModule = nullptr;

}

}

void DelayLoadLibrary::Ensure(DWORD dwFlags)

{

if (!m\_isInit)

{

m\_hModule = LoadLibraryEx(GetLibraryName(), nullptr, dwFlags);

m\_isInit = true;

}

}

void DelayLoadLibrary::EnsureFromSystemDirOnly()

{

Ensure(LOAD\_LIBRARY\_SEARCH\_SYSTEM32);

}

FARPROC DelayLoadLibrary::GetFunction(\_\_in LPCSTR lpFunctionName)

{

if (m\_hModule)

{

return GetProcAddress(m\_hModule, lpFunctionName);

}

return nullptr;

}

bool DelayLoadLibrary::IsAvailable()

{

return m\_hModule != nullptr;

}

#if PDATA\_ENABLED

static NtdllLibrary NtdllLibraryObject;

NtdllLibrary\* NtdllLibrary::Instance = &NtdllLibraryObject;

LPCTSTR NtdllLibrary::GetLibraryName() const

{

return L"ntdll.dll";

}

\_Success\_(return == 0)

DWORD NtdllLibrary::AddGrowableFunctionTable( \_Out\_ PVOID \* DynamicTable,

\_In\_reads\_(MaximumEntryCount) PRUNTIME\_FUNCTION FunctionTable,

\_In\_ DWORD EntryCount,

\_In\_ DWORD MaximumEntryCount,

\_In\_ ULONG\_PTR RangeBase,

\_In\_ ULONG\_PTR RangeEnd )

{

if(m\_hModule)

{

if(addGrowableFunctionTable == NULL)

{

addGrowableFunctionTable = (PFnRtlAddGrowableFunctionTable)GetFunction("RtlAddGrowableFunctionTable");

if(addGrowableFunctionTable == NULL)

{

Assert(false);

return 1;

}

}

return addGrowableFunctionTable(DynamicTable,

FunctionTable,

EntryCount,

MaximumEntryCount,

RangeBase,

RangeEnd);

}

return 1;

}

VOID NtdllLibrary::DeleteGrowableFunctionTable( \_In\_ PVOID DynamicTable )

{

if(m\_hModule)

{

if(deleteGrowableFunctionTable == NULL)

{

deleteGrowableFunctionTable = (PFnRtlDeleteGrowableFunctionTable)GetFunction("RtlDeleteGrowableFunctionTable");

if(deleteGrowableFunctionTable == NULL)

{

Assert(false);

return;

}

}

deleteGrowableFunctionTable(DynamicTable);

}

}

VOID NtdllLibrary::GrowFunctionTable(\_Inout\_ PVOID DynamicTable, \_In\_ ULONG NewEntryCount)

{

if (m\_hModule)

{

if (growFunctionTable == nullptr)

{

growFunctionTable = (PFnRtlGrowFunctionTable)GetFunction("RtlGrowFunctionTable");

if (growFunctionTable == nullptr)

{

Assert(false);

return;

}

}

growFunctionTable(DynamicTable, NewEntryCount);

}

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

class DelayLoadLibrary

{

protected:

HMODULE m\_hModule;

bool m\_isInit;

public:

DelayLoadLibrary();

virtual ~DelayLoadLibrary();

virtual LPCTSTR GetLibraryName() const = 0;

FARPROC GetFunction(\_\_in LPCSTR lpFunctionName);

void EnsureFromSystemDirOnly();

bool IsAvailable();

private:

void Ensure(DWORD dwFlags = 0);

};

#if PDATA\_ENABLED

// This needs to be delay loaded because it is available on

// Win8 only

class NtdllLibrary : protected DelayLoadLibrary

{

private:

typedef \_Success\_(return == 0) DWORD (NTAPI \*PFnRtlAddGrowableFunctionTable)(\_Out\_ PVOID \* DynamicTable,

\_In\_reads\_(MaximumEntryCount) PRUNTIME\_FUNCTION FunctionTable,

\_In\_ DWORD EntryCount,

\_In\_ DWORD MaximumEntryCount,

\_In\_ ULONG\_PTR RangeBase,

\_In\_ ULONG\_PTR RangeEnd);

PFnRtlAddGrowableFunctionTable addGrowableFunctionTable;

typedef VOID (NTAPI \*PFnRtlDeleteGrowableFunctionTable)(\_In\_ PVOID DynamicTable);

PFnRtlDeleteGrowableFunctionTable deleteGrowableFunctionTable;

typedef VOID (NTAPI \*PFnRtlGrowFunctionTable)(\_Inout\_ PVOID DynamicTable, \_In\_ ULONG NewEntryCount);

PFnRtlGrowFunctionTable growFunctionTable;

public:

static NtdllLibrary\* Instance;

NtdllLibrary() : DelayLoadLibrary(),

addGrowableFunctionTable(NULL),

deleteGrowableFunctionTable(NULL),

growFunctionTable(NULL)

{

this->EnsureFromSystemDirOnly();

}

LPCTSTR GetLibraryName() const;

\_Success\_(return == 0)

DWORD AddGrowableFunctionTable(\_Out\_ PVOID \* DynamicTable,

\_In\_reads\_(MaximumEntryCount) PRUNTIME\_FUNCTION FunctionTable,

\_In\_ DWORD EntryCount,

\_In\_ DWORD MaximumEntryCount,

\_In\_ ULONG\_PTR RangeBase,

\_In\_ ULONG\_PTR RangeEnd);

VOID DeleteGrowableFunctionTable(\_In\_ PVOID DynamicTable);

VOID GrowFunctionTable(\_\_inout PVOID DynamicTable, \_\_in ULONG NewEntryCount);

};

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#include "core\EtwTraceCore.h"

#ifdef ENABLE\_JS\_ETW

extern "C" {

ETW\_INLINE

VOID EtwCallback(

ULONG controlCode,

PVOID callbackContext)

{

EtwCallbackApi::OnSessionChange(controlCode, callbackContext);

}

}

bool EtwTraceCore::s\_registered = false;

//

// Registers the ETW provider - this is usually done on Jscript DLL load

// After registration, we will receive callbacks when ETW tracing is enabled/disabled.

//

void EtwTraceCore::Register()

{

if (!s\_registered)

{

s\_registered = true;

JS\_ETW(EventRegisterMicrosoft\_IE());

JS\_ETW(EventRegisterMicrosoft\_JScript());

JS\_ETW(EventRegisterMicrosoft\_JScript\_Internal());

// This will be used to distinguish the provider we are getting the callback for.

PROVIDER\_JSCRIPT9\_Context.RegistrationHandle = Microsoft\_JScriptHandle;

BERP\_IE\_Context.RegistrationHandle = Microsoft\_IEHandle;

}

}

//

// Unregister to ensure we do not get callbacks.

//

void EtwTraceCore::UnRegister()

{

if (s\_registered)

{

s\_registered = false;

JS\_ETW(EventUnregisterMicrosoft\_IE());

JS\_ETW(EventUnregisterMicrosoft\_JScript());

JS\_ETW(EventUnregisterMicrosoft\_JScript\_Internal());

}

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef ENABLE\_JS\_ETW

#define PAIR(a,b) a ## b

#define GCETW(e, args) \

if (IsMemProtectMode()) \

{ \

PAIR(EventWriteMEMPROTECT\_ ## e, args); \

} \

else \

{ \

PAIR(EventWriteJSCRIPT\_ ## e, args); \

}

#define JS\_ETW(s) s

#define IS\_JS\_ETW(s) s

// C-style callback

extern "C" {

void EtwCallback(

ULONG controlCode,

PVOID callbackContext);

}

class EtwCallbackApi

{

public:

static void OnSessionChange(ULONG controlCode, PVOID callbackContext);

};

// The macro 'MCGEN\_PRIVATE\_ENABLE\_CALLBACK\_V2' below should be defined before microsoft-scripting-jscript9events.h

// is included. This is a complier assert to ensure this happens. If this does not happen rundown events will not work.

#ifdef JSCRIPT\_METHOD\_METHODLOAD\_OPCODE

CompileAssert(false)

#endif

// Declaring this macro enables the registration of the callback on DLL attach

#define MCGEN\_PRIVATE\_ENABLE\_CALLBACK\_V2(SourceId, ControlCode, Level, MatchAnyKeyword, MatchAllKeyword, FilterData, CallbackContext) \

EtwCallback(ControlCode, CallbackContext)

#include <microsoft-scripting-chakraevents.h>

#include <ieresp\_mshtml.h>

#include <microsoft-scripting-jscript9.internalevents.h>

//

// Encapsulates base routines to initialize ETW tracing in the module

//

class EtwTraceCore

{

public:

static void Register();

static void UnRegister();

static bool s\_registered;

};

#else

#define GCETW(e, ...)

#define JS\_ETW(s)

#define IS\_JS\_ETW(s) (false)

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#ifdef FAULT\_INJECTION

#include "io.h"

#include "share.h"

#undef DBGHELP\_TRANSLATE\_TCHAR

#define \_NO\_CVCONST\_H

// dbghelp.h is not clean with warning 4091

#pragma warning(push)

#pragma warning(disable: 4091) /\* warning C4091: 'typedef ': ignored on left of '' when no variable is declared \*/

#include <dbghelp.h>

#pragma warning(pop)

namespace Js

{

#pragma region helpers

#define FIDELAYLOAD(fn) static decltype(fn)\* pfn##fn = nullptr

FIDELAYLOAD(SymInitialize);

FIDELAYLOAD(SymCleanup);

FIDELAYLOAD(SymFromAddrW);

FIDELAYLOAD(SymFromNameW);

FIDELAYLOAD(SymEnumSymbolsW);

FIDELAYLOAD(SymGetModuleInfoW64);

FIDELAYLOAD(SymMatchStringW);

FIDELAYLOAD(SymSetOptions);

FIDELAYLOAD(MiniDumpWriteDump);

FIDELAYLOAD(SymFunctionTableAccess64);

FIDELAYLOAD(SymGetModuleBase64);

FIDELAYLOAD(StackWalk64);

#undef FIDELAYLOAD

template<typename CharT>

bool isEqualIgnoreCase(CharT c1, CharT c2)

{

return c1 == c2

|| ((c2 <= 'Z') && (c1 >= 'a') && (c1 - c2 == 'a' - 'A'))

|| ((c1 <= 'Z') && (c2 >= 'a') && (c2 - c1 == 'a' - 'A'));

}

template<typename CharT>

CharT \*stristr(const CharT \* cs1,

const CharT \* cs2)

{

CharT \*cp = (CharT \*)cs1;

CharT \*s1, \*s2;

if (!\*cs2)

return (CharT \*)cs1;

while (\*cp)

{

s1 = cp;

s2 = (CharT \*)cs2;

while (\*s1 && \*s2 && isEqualIgnoreCase(\*s1, \*s2))

s1++, s2++;

if (!\*s2)

return cp;

cp++;

}

return nullptr;

}

static wchar\_t\* trimRight(\_Inout\_z\_ wchar\_t\* str)

{

auto tmp = str + wcslen(str);

while (!isprint(\*--tmp));

\*(tmp + 1) = L'\0';

return str;

}

static int8 const\* hexTable = []()->int8\*{

static int8 hex[256] = { 0 };

memset(hex, 0xff, 256);

for (int8 i = '0'; i <= '9'; i++) hex[i] = i - '0';

for (int8 i = 'a'; i <= 'f'; i++) hex[i] = i - 'a' + 10;

for (int8 i = 'A'; i <= 'F'; i++) hex[i] = i - 'A' + 10;

return hex;

}();

template<typename CharT>

static UINT\_PTR HexStrToAddress(const CharT\* str)

{

UINT\_PTR address = 0;

while (\*str == '0' || \*str == '`' || \*str == 'x' || \*str == 'X')

str++; // leading zero

do

{

if (\*str == '`') // amd64 address

continue;

if (hexTable[\*str & 0xff] < 0)

return address;

address = 16 \* address + hexTable[\*str & 0xff];

} while (\*(++str));

return address;

}

#if \_M\_X64

// for amd64 jit frame, RtlCaptureStackBackTrace stops walking after hitting jit frame on amd64

\_\_declspec(noinline)

WORD StackTrace64(\_In\_ DWORD FramesToSkip,

\_In\_ DWORD FramesToCapture,

\_Out\_writes\_to\_(FramesToCapture, return) PVOID \* BackTrace,

\_Out\_opt\_ PDWORD BackTraceHash,

\_In\_opt\_ const CONTEXT\* pCtx = nullptr)

{

CONTEXT Context;

KNONVOLATILE\_CONTEXT\_POINTERS NvContext;

UNWIND\_HISTORY\_TABLE UnwindHistoryTable;

PRUNTIME\_FUNCTION RuntimeFunction;

PVOID HandlerData;

ULONG64 EstablisherFrame;

ULONG64 ImageBase;

ULONG Frame = 0;

if (BackTraceHash)

{

\*BackTraceHash = 0;

}

if (pCtx == nullptr)

{

RtlCaptureContext(&Context);

}

else

{

memcpy(&Context, pCtx, sizeof(CONTEXT));

}

RtlZeroMemory(&UnwindHistoryTable, sizeof(UNWIND\_HISTORY\_TABLE));

while (true)

{

RuntimeFunction = RtlLookupFunctionEntry(Context.Rip, &ImageBase, &UnwindHistoryTable);

RtlZeroMemory(&NvContext, sizeof(KNONVOLATILE\_CONTEXT\_POINTERS));

if (!RuntimeFunction)

{

Context.Rip = (ULONG64)(\*(PULONG64)Context.Rsp);

Context.Rsp += 8;

}

else

{

RtlVirtualUnwind(UNW\_FLAG\_NHANDLER, ImageBase, Context.Rip, RuntimeFunction,

&Context, &HandlerData, &EstablisherFrame, &NvContext);

}

if (!Context.Rip)

{

break;

}

if (FramesToSkip > 0)

{

FramesToSkip--;

continue;

}

if (Frame >= FramesToCapture)

{

break;

}

BackTrace[Frame] = (PVOID)Context.Rip;

if (BackTraceHash)

{

\*BackTraceHash += (Context.Rip & 0xffffffff);

}

Frame++;

}

return (WORD)Frame;

}

#define CaptureStack(FramesToSkip, FramesToCapture, BackTrace, BackTraceHash) \

StackTrace64(FramesToSkip, FramesToCapture, BackTrace, BackTraceHash)

#elif defined (\_M\_IX86)

#pragma optimize( "g", off )

#pragma warning( push )

#pragma warning( disable : 4748 )

#pragma warning( disable : 4995 )

WORD StackTrace86(

\_In\_ DWORD FramesToSkip,

\_In\_ DWORD FramesToCapture,

\_Out\_writes\_to\_(FramesToCapture, return) PVOID \* BackTrace,

\_Inout\_opt\_ PDWORD BackTraceHash,

\_\_in\_opt CONST PCONTEXT InitialContext = NULL

)

{

\_Analysis\_assume\_(FramesToSkip >= 0);

\_Analysis\_assume\_(FramesToCapture >= 0);

DWORD MachineType;

CONTEXT Context;

STACKFRAME64 StackFrame;

if (InitialContext == NULL)

{

//RtlCaptureContext( &Context );

ZeroMemory(&Context, sizeof(CONTEXT));

Context.ContextFlags = CONTEXT\_CONTROL;

\_\_asm

{

Label:

mov[Context.Ebp], ebp;

mov[Context.Esp], esp;

mov eax, [Label];

mov[Context.Eip], eax;

}

}

else

{

CopyMemory(&Context, InitialContext, sizeof(CONTEXT));

}

ZeroMemory(&StackFrame, sizeof(STACKFRAME64));

MachineType = IMAGE\_FILE\_MACHINE\_I386;

StackFrame.AddrPC.Offset = Context.Eip;

StackFrame.AddrPC.Mode = AddrModeFlat;

StackFrame.AddrFrame.Offset = Context.Ebp;

StackFrame.AddrFrame.Mode = AddrModeFlat;

StackFrame.AddrStack.Offset = Context.Esp;

StackFrame.AddrStack.Mode = AddrModeFlat;

WORD FrameCount = 0;

while (FrameCount < FramesToSkip + FramesToCapture)

{

if (!pfnStackWalk64(MachineType, GetCurrentProcess(), GetCurrentThread(), &StackFrame,

NULL, NULL, pfnSymFunctionTableAccess64, pfnSymGetModuleBase64, NULL))

{

break;

}

if (StackFrame.AddrPC.Offset != 0)

{

if (FrameCount >= FramesToSkip)

{

#pragma warning(suppress: 22102)

#pragma warning(suppress: 26014)

BackTrace[FrameCount - FramesToSkip] = (PVOID)StackFrame.AddrPC.Offset;

if (BackTraceHash)

{

\*BackTraceHash += (StackFrame.AddrPC.Offset & 0xffffffff);

}

}

FrameCount++;

}

else

{

break;

}

}

if (FrameCount > FramesToSkip)

{

return (WORD)(FrameCount - FramesToSkip);

}

else

{

return 0;

}

}

#pragma warning( pop )

#pragma optimize( "g", on )

#define CaptureStack(FramesToSkip, FramesToCapture, BackTrace, BackTraceHash) \

RtlCaptureStackBackTrace(FramesToSkip, FramesToCapture, BackTrace, BackTraceHash)

#else

#define CaptureStack(FramesToSkip, FramesToCapture, BackTrace, BackTraceHash) \

RtlCaptureStackBackTrace(FramesToSkip, FramesToCapture, BackTrace, BackTraceHash)

#endif

struct SymbolInfoPackage : public SYMBOL\_INFO\_PACKAGEW

{

SymbolInfoPackage() { Init(); }

void Init()

{

si.SizeOfStruct = sizeof(SYMBOL\_INFOW);

si.MaxNameLen = sizeof(name);

}

};

struct ModuleInfo : public IMAGEHLP\_MODULEW64

{

ModuleInfo() { Init(); }

void Init()

{

SizeOfStruct = sizeof(IMAGEHLP\_MODULEW64);

}

};

bool FaultInjection::InitializeSym()

{

if (symInitialized)

{

return true;

}

// load dbghelp APIs

if (hDbgHelp == NULL)

{

hDbgHelp = LoadLibraryEx(L"dbghelp.dll", 0, 0);

}

if (hDbgHelp == NULL)

{

fwprintf(stderr, L"Failed to load dbghelp.dll for stack walking, gle=0x%08x\n", GetLastError());

fflush(stderr);

return false;

}

#define FIDELAYLOAD(fn) pfn##fn = (decltype(fn)\*)GetProcAddress(hDbgHelp, #fn); \

if (pfn##fn == nullptr){\

fwprintf(stderr, L"Failed to load sigs:%s\n", L#fn); \

fflush(stderr); \

return false; \

}

FIDELAYLOAD(SymInitialize);

FIDELAYLOAD(SymCleanup);

FIDELAYLOAD(SymFromAddrW);

FIDELAYLOAD(SymFromNameW);

FIDELAYLOAD(SymEnumSymbolsW);

FIDELAYLOAD(SymGetModuleInfoW64);

FIDELAYLOAD(SymMatchStringW);

FIDELAYLOAD(SymSetOptions);

FIDELAYLOAD(MiniDumpWriteDump);

FIDELAYLOAD(SymFunctionTableAccess64);

FIDELAYLOAD(SymGetModuleBase64);

FIDELAYLOAD(StackWalk64);

#undef FIDELAYLOAD

// TODO: StackBackTrace.cpp also call SymInitialize, but this can only be called once before cleanup

if (!pfnSymInitialize(GetCurrentProcess(), NULL, TRUE))

{

fwprintf(stderr, L"SymInitialize failed, gle=0x%08x\n", GetLastError());

fflush(stderr);

return false;

}

symInitialized = true;

return true;

}

#pragma endregion helpers

FaultInjection FaultInjection::Global;

static CriticalSection cs\_Sym; // for Sym\* method is not thread safe

const auto& globalFlags = Js::Configuration::Global.flags;

PVOID FaultInjection::vectoredExceptionHandler = nullptr;

DWORD FaultInjection::exceptionFilterRemovalLastError = 0;

int(\*Js::FaultInjection::pfnHandleAV)(int, PEXCEPTION\_POINTERS) = nullptr;

static SymbolInfoPackage sip;

static ModuleInfo mi;

const wchar\_t\* crashStackStart = L"=====Callstack for this exception=======\n";

const wchar\_t\* crashStackEnd = L"=====End of callstack for this exception=======\n";

const wchar\_t\* injectionStackStart = L"=====Fault injecting record=====\n";

const wchar\_t\* injectionStackEnd = L"=====End of Fault injecting record=====\n";

typedef struct \_RANGE{

UINT\_PTR startAddress;

UINT\_PTR endAddress;

}RANGE, \*PRANGE;

typedef struct \_FUNCTION\_SIGNATURES

{

int count;

RANGE signatures[ANYSIZE\_ARRAY];

} FUNCTION\_SIGNATURES, \*PFUNCTION\_SIGNATURES;

// function address ranges of each signature

// use for faster address matching instead of symbol table lookup when reproing

PFUNCTION\_SIGNATURES baselineFuncSigs[FaultInjection::MAX\_FRAME\_COUNT] = { 0 };

// record hit count of each frame when Faults are injected.

unsigned int stackMatchRank[FaultInjection::MAX\_FRAME\_COUNT] = { 0 };

#define FAULT\_TYPE(x) L#x,\

wchar\_t \*FaultInjection::FaultTypeNames[] =

{

#include "FaultTypes.h"

};

#undef FAULT\_TYPE

static\_assert(sizeof(FaultInjection::FaultTypeNames) == FaultInjection::FaultType::FaultTypeCount\*sizeof(wchar\_t\*),

"FaultTypeNames count is wrong");

void FaultInjection::FaultInjectionTypes::EnableType(FaultType type)

{

Assert(type >= 0 && type < FaultType::FaultTypeCount);

setBit(type, 1);

}

bool FaultInjection::FaultInjectionTypes::IsEnabled(FaultType type)

{

Assert(type >= 0 && type < FaultType::FaultTypeCount);

return getBit(type) == 0x1;

}

bool FaultInjection::FaultInjectionTypes::IsEnabled(const wchar\_t\* name)

{

for (int type = 0; type < FaultType::FaultTypeCount; type++)

{

if (wcscmp(FaultTypeNames[type], name) == 0)

return getBit(type) == 0x1;

}

AssertMsg(false, "Unknown fault type name");

return false;

}

FaultInjection::FaultInjection()

{

stackMatchInitialized = Uninitialized;

countOfInjectionPoints = 0;

hDbgHelp = NULL;

InjectionFirstRecord = nullptr;

InjectionLastRecordRef = &InjectionFirstRecord;

InjectionRecordsCount = 0;

FaultInjectionCookie = 0;

baselineFrameCount = 0;

stackHashOfAllInjectionPointsSize = 256;

stackHashOfAllInjectionPoints = (ULONG\_PTR\*)malloc(stackHashOfAllInjectionPointsSize\*sizeof(ULONG\_PTR));

faultInjectionTypes = nullptr;

symInitialized = false;

for (int i = 0; i < MAX\_FRAME\_COUNT; i++)

{

baselineStack[i] = nullptr;

baselineAddresses[i] = 0;

}

}

FaultInjection::~FaultInjection()

{

RemoveExceptionFilters();

// when fault injection count only is passing from jscript.config(in case of running on 3rd part host)

// and the host don't have code to output the fault injection count, we still able to do the fault injection test

if (globalFlags.FaultInjection == FaultMode::CountOnly

|| globalFlags.FaultInjection == FaultMode::StackMatchCountOnly)

{

fprintf(stderr, "FaultInjection - Total Allocation Count:%u\n", countOfInjectionPoints);

fflush(stderr);

FILE \*fp;

char countFileName[64];

sprintf\_s(countFileName, "ChakraFaultInjectionCount\_%u.txt", GetCurrentProcessId());

if (fopen\_s(&fp, countFileName, "w") == 0)

{

fprintf(fp, "FaultInjection - Total Allocation Count:%u\n", countOfInjectionPoints);

fflush(fp);

fclose(fp);

}

for (int i = 0; i < MAX\_FRAME\_COUNT; i++)

{

if (stackMatchRank[i] == 0)

{

break;

}

fwprintf(stderr, L"FaultInjection stack matching rank %d: %u\n", i + 1, stackMatchRank[i]);

}

fflush(stderr);

}

if (globalFlags.FaultInjection == StackHashCountOnly)

{

FILE \*fp;

if (fopen\_s(&fp, "ChakraFaultInjectionHashes.txt", "w") == 0)

{

for (uint i = 0; i < countOfInjectionPoints; i++)

{

fprintf(fp, "%p\n", (void\*)stackHashOfAllInjectionPoints[i]);

}

fflush(fp);

fclose(fp);

}

}

free(stackHashOfAllInjectionPoints);

stackHashOfAllInjectionPoints = nullptr;

if (globalFlags.FaultInjection == FaultMode::DisplayAvailableFaultTypes)

{

Output::Print(L"Available Fault Types:\n");

for (int i = 0; i < FaultType::FaultTypeCount; i++)

{

Output::Print(L"%d-%s\n", i, FaultTypeNames[i]);

}

Output::Flush();

}

InjectionRecord\* head = InjectionFirstRecord;

while (head != nullptr)

{

InjectionRecord\* next = head->next;

if (head->StackData)

{

free(head->StackData);

}

free(head);

head = next;

}

for (int i = 0; i < MAX\_FRAME\_COUNT; i++)

{

if (baselineStack[i])

{

free(baselineStack[i]);

}

if (baselineFuncSigs[i])

{

free(baselineFuncSigs[i]);

}

}

if (stackMatchInitialized == Succeeded)

{

pfnSymCleanup(GetCurrentProcess());

}

if (hDbgHelp)

{

FreeLibrary(hDbgHelp);

}

if (faultInjectionTypes)

{

faultInjectionTypes->~FaultInjectionTypes();

NoCheckHeapDelete(faultInjectionTypes);

}

}

bool FaultInjection::IsFaultEnabled(FaultType faultType)

{

if (!faultInjectionTypes)

{

faultInjectionTypes = NoCheckHeapNew(FaultInjectionTypes);

if ((const wchar\_t\*)globalFlags.FaultInjectionType == nullptr)

{

// no -FaultInjectionType specified, inject all

faultInjectionTypes->EnableAll();

}

else

{

ParseFaultTypes(globalFlags.FaultInjectionType);

}

}

return faultInjectionTypes->IsEnabled(faultType);

}

bool FaultInjection::IsFaultInjectionOn(FaultType faultType)

{

return globalFlags.FaultInjection >= 0 //-FaultInjection switch

&& IsFaultEnabled(faultType);

}

void FaultInjection::ParseFaultTypes(const wchar\_t\* szFaultTypes)

{

auto charCount = wcslen(szFaultTypes) + 1;

wchar\_t\* szTypes = (wchar\_t\*)malloc(charCount\*sizeof(wchar\_t));

AssertMsg(szTypes, "OOM in FaultInjection Infra");

wcscpy\_s(szTypes, charCount, szFaultTypes);

const wchar\_t\* delims = L",";

wchar\_t \*nextTok = nullptr;

wchar\_t\* tok = wcstok\_s(szTypes, delims, &nextTok);

while (tok != NULL)

{

if (wcslen(tok) > 0)

{

if (iswdigit(tok[0]))

{

auto numType = \_wtoi(tok);

for (int i = 0; i< FaultType::FaultTypeCount; i++)

{

if (numType & (1 << i))

{

faultInjectionTypes->EnableType(i);

}

}

}

else if (tok[0] == L'#')

{

// FaultInjectionType:#1-4,#6 format, not flags

auto tok1 = tok + 1;

if (wcslen(tok1)>0 && iswdigit(tok1[0]))

{

wchar\_t\* pDash = wcschr(tok1, L'-');

if (pDash)

{

for (int i = \_wtoi(tok1); i <= \_wtoi(pDash + 1); i++)

{

faultInjectionTypes->EnableType(i);

}

}

else

{

faultInjectionTypes->EnableType(\_wtoi(tok1));

}

}

}

else

{

for (int i = 0; i < FaultType::FaultTypeCount; i++)

{

if (\_wcsicmp(FaultTypeNames[i], tok) == 0)

{

faultInjectionTypes->EnableType(i);

break;

}

}

}

}

tok = wcstok\_s(NULL, delims, &nextTok);

}

free(szTypes);

}

static void SmashLambda(\_Inout\_z\_ wchar\_t\* str)

{

//jscript9test!<lambda\_dc7f9e8c591f1832700d6567e43faa6c>::operator()

const wchar\_t lambdaSig[] = L"<lambda\_";

const int lambdaSigLen = (int)wcslen(lambdaSig);

auto temp = str;

while (temp != nullptr)

{

auto lambdaStart = wcsstr(temp, lambdaSig);

temp = nullptr;

if (lambdaStart != nullptr)

{

auto lambdaEnd = wcschr(lambdaStart, L'>');

temp = lambdaEnd;

if (lambdaEnd != nullptr && lambdaEnd - lambdaStart == lambdaSigLen + 32)

{

lambdaStart += lambdaSigLen;

while (lambdaStart < lambdaEnd)

{

\*(lambdaStart++) = L'?';

}

}

}

}

}

bool FaultInjection::EnsureStackMatchInfraInitialized()

{

if (stackMatchInitialized == Succeeded)

{

return true;

}

else if (stackMatchInitialized == FailedToInitialize)

{

// previous try to initialize and failed

return false;

}

else if (stackMatchInitialized == Uninitialized)

{

stackMatchInitialized = FailedToInitialize; //tried

if (!InitializeSym())

{

return false;

}

// read baseline stack file

FILE \*fp = nullptr;

const wchar\_t \*stackFile = globalFlags.FaultInjectionStackFile;//default: L"stack.txt";

auto err = \_wfopen\_s(&fp, stackFile, L"r");

if (err != 0 || fp == nullptr)

{

fwprintf(stderr, L"Failed to load %s, gle=0x%08x\n", stackFile, GetLastError());

fflush(stderr);

return false;

}

wchar\_t buffer[MAX\_SYM\_NAME]; // assume the file is normal

unsigned int maxLineCount =

(globalFlags.FaultInjectionStackLineCount < 0

|| globalFlags.FaultInjectionStackLineCount > MAX\_FRAME\_COUNT

|| globalFlags.FaultInjection == FaultMode::StackMatchCountOnly)

? MAX\_FRAME\_COUNT : globalFlags.FaultInjectionStackLineCount;

while (fgetws(buffer, MAX\_SYM\_NAME, fp))

{

if (wcscmp(buffer, injectionStackStart) == 0)

{

baselineFrameCount = 0;

continue;

}

if (baselineFrameCount >= maxLineCount)

{

continue; // don't break because we can hit the start marker and reset

}

const wchar\_t jscript9test[] = L"jscript9test!";

const wchar\_t jscript9[] = L"jscript9!";

wchar\_t\* symbolStart = stristr(buffer, jscript9test);

if (symbolStart == nullptr)

{

symbolStart = stristr(buffer, jscript9);

}

if (symbolStart == nullptr)

{

continue;// no "jscript9test!", skip this line

}

if (wcsstr(symbolStart, L"Js::FaultInjection") != NULL)

{ // skip faultinjection infra frames.

continue;

}

auto plus = wcschr(symbolStart, L'+');

if (plus)

{

\*plus = L'\0';

}

else

{

trimRight(symbolStart);

}

SmashLambda(symbolStart);

size\_t len = wcslen(symbolStart);

if (baselineStack[baselineFrameCount] == nullptr)

{

baselineStack[baselineFrameCount] = (wchar\_t\*)malloc((len + 1)\*sizeof(wchar\_t));

AssertMsg(baselineStack[baselineFrameCount], "OOM in FaultInjection Infra");

}

else

{

auto tmp = (wchar\_t\*)realloc(baselineStack[baselineFrameCount], (len + 1)\*sizeof(wchar\_t));

AssertMsg(tmp, "OOM in FaultInjection Infra");

baselineStack[baselineFrameCount] = tmp;

}

wcscpy\_s(baselineStack[baselineFrameCount], len + 1, symbolStart);

baselineFrameCount++;

}

fclose(fp);

OutputDebugString(L"Fault will be injected when hit following stack:\n");

for (uint i = 0; i<baselineFrameCount; i++)

{

OutputDebugString(baselineStack[i]);

OutputDebugString(L"\n");

if (wcschr(baselineStack[i], '\*') != nullptr || wcschr(baselineStack[i], '?') != nullptr)

{

continue; // there's wildcard in this line, don't use address matching

}

// enum symbols, if succeed we compare with address when doing stack matching

pfnSymEnumSymbolsW(GetCurrentProcess(), 0, baselineStack[i],

[](\_In\_ PSYMBOL\_INFOW pSymInfo, \_In\_ ULONG SymbolSize, \_In\_opt\_ PVOID UserContext)->BOOL

{

Assert(UserContext != nullptr); // did passed in the user context

if (pSymInfo->Size > 0)

{

PFUNCTION\_SIGNATURES\* sigs = (PFUNCTION\_SIGNATURES\*)UserContext;

int count = (\*sigs) == nullptr ? 0 : (\*sigs)->count;

auto tmp = (PFUNCTION\_SIGNATURES)realloc(\*sigs, sizeof(FUNCTION\_SIGNATURES) + count\*sizeof(RANGE));

AssertMsg(tmp, "OOM when allocating for FaultInjection Stack matching objects");

\*sigs = tmp;

(\*sigs)->count = count;

(\*sigs)->signatures[count].startAddress = (UINT\_PTR)pSymInfo->Address;

(\*sigs)->signatures[count].endAddress = (UINT\_PTR)(pSymInfo->Address + pSymInfo->Size);

(\*sigs)->count++;

}

return TRUE;

}, &baselineFuncSigs[i]);

}

stackMatchInitialized = Succeeded; // initialized

return true;

}

return false;

}

bool FaultInjection::IsCurrentStackMatch()

{

AutoCriticalSection autocs(&cs\_Sym); // sym\* API is thread unsafe

if (!EnsureStackMatchInfraInitialized())

{

return false;

}

DWORD64 dwSymDisplacement = 0;

auto hProcess = GetCurrentProcess();

static void\* framesBuffer[FaultInjection::MAX\_FRAME\_COUNT];

auto frameCount = CaptureStack(0, MAX\_FRAME\_COUNT, framesBuffer, 0);

uint n = 0;

for (uint i = 0; i < frameCount; i++)

{

if (n >= baselineFrameCount)

{

return true;

}

if (!AutoSystemInfo::Data.IsJscriptModulePointer(framesBuffer[i]))

{ // skip non-Chakra frame

continue;

}

bool match = false;

if (baselineFuncSigs[n] != nullptr)

{

for (int j = 0; j<baselineFuncSigs[n]->count; j++)

{

match = baselineFuncSigs[n]->signatures[j].startAddress <= (UINT\_PTR)framesBuffer[i]

&& (UINT\_PTR)framesBuffer[i] < baselineFuncSigs[n]->signatures[j].endAddress;

if (match)

{

break;

}

}

}

else

{

// fallback to symbol name matching

sip.Init();

if (!pfnSymFromAddrW(hProcess, (DWORD64)framesBuffer[i], &dwSymDisplacement, &sip.si))

{

continue;

}

SmashLambda(sip.si.Name);

// Only search sigs name, can use wildcard in baseline file

match = stristr(baselineStack[n], sip.si.Name) != nullptr

|| pfnSymMatchStringW(sip.si.Name, baselineStack[n], false);// wildcard

}

if (match)

{

stackMatchRank[n]++;

if (n == 0)

{

n++;

continue;

}

}

else if (n > 0)

{

return false;

}

// First line in baseline is found, moving forward.

if (n > 0)

{

n++;

}

}

return false;

}

static bool faultInjectionDebug = false;

bool FaultInjection::InstallExceptionFilters()

{

if (GetEnvironmentVariable(L"FAULTINJECTION\_DEBUG", nullptr, 0) != 0)

{

faultInjectionDebug = true;

}

if (globalFlags.FaultInjection >= 0 && !IsDebuggerPresent())

{

// initialize symbol system here instead of inside the exception filter

// because some hard stack overflow can happen in SymInitialize

// when the exception filter is handling stack overflow exception

if (!FaultInjection::Global.InitializeSym())

{

return false;

}

//C28725: Use Watson instead of this SetUnhandledExceptionFilter.

#pragma prefast(suppress: 28725)

SetUnhandledExceptionFilter([](\_In\_ struct \_EXCEPTION\_POINTERS \*ExceptionInfo)->LONG

{

return FaultInjectionExceptionFilter(ExceptionInfo);

});

vectoredExceptionHandler = AddVectoredExceptionHandler(0, [](\_In\_ struct \_EXCEPTION\_POINTERS \*ExceptionInfo)->LONG

{

switch (ExceptionInfo->ExceptionRecord->ExceptionCode)

{

// selected fatal exceptions:

case STATUS\_ACCESS\_VIOLATION:

{

if (pfnHandleAV

&& pfnHandleAV(ExceptionInfo->ExceptionRecord->ExceptionCode, ExceptionInfo) == EXCEPTION\_CONTINUE\_EXECUTION)

{

return EXCEPTION\_CONTINUE\_EXECUTION;

}

}

case STATUS\_ASSERTION\_FAILURE:

case STATUS\_STACK\_OVERFLOW:

FaultInjectionExceptionFilter(ExceptionInfo);

TerminateProcess(::GetCurrentProcess(), ExceptionInfo->ExceptionRecord->ExceptionCode);

default:

return EXCEPTION\_CONTINUE\_SEARCH;

}

});

return true;

}

return false;

}

void FaultInjection::RemoveExceptionFilters()

{

//C28725: Use Watson instead of this SetUnhandledExceptionFilter.

#pragma prefast(suppress: 28725)

SetUnhandledExceptionFilter(nullptr);

if (vectoredExceptionHandler != nullptr)

{

RemoveVectoredExceptionHandler(vectoredExceptionHandler);

exceptionFilterRemovalLastError = GetLastError(); // looks sometimes the removal fails

vectoredExceptionHandler = nullptr;

}

}

// Calculate stack hash by adding the addresses (only jscript9 frames)

UINT\_PTR FaultInjection::CalculateStackHash(void\* frames[], WORD frameCount, WORD framesToSkip)

{

UINT\_PTR hash = 0;

for (int i = framesToSkip; i < frameCount; i++)

{

if (AutoSystemInfo::Data.IsJscriptModulePointer(frames[i]))

{

hash += (UINT\_PTR)frames[i] - AutoSystemInfo::Data.dllLoadAddress;

}

}

return hash;

}

// save the stack data for dump debugging use

// to get list of fault injection points:

// !list -t jscript9test!Js::FaultInjection::InjectionRecord.next -e -x "dps @$extret @$extret+0x128" poi(@@c++(&jscript9test!Js::FaultInjection::Global.InjectionFirstRecord))

// to rebuild the stack (locals are available)

// .cxr @@C++(&jscript9test!Js::FaultInjection::Global.InjectionFirstRecord->Context)

\_\_declspec(noinline) void FaultInjection::dumpCurrentStackData(LPCWSTR name /\*= nullptr\*/, size\_t size /\*= 0\*/)

{

#if !defined(\_M\_ARM32\_OR\_ARM64)

static bool keepBreak = true; // for disabling following breakpoint by editing the value

if (keepBreak && IsDebuggerPresent())

{

DebugBreak();

}

InjectionRecord\* record = (InjectionRecord\*)malloc(sizeof(InjectionRecord));

if (record == nullptr) return;

ZeroMemory(record, sizeof(InjectionRecord));

auto \_stackbasepointer = ((PNT\_TIB)NtCurrentTeb())->StackBase;

// context

RtlCaptureContext(&record->Context);

#if \_M\_X64

auto& \_stackpointer = record->Context.Rsp;

auto& \_basepointer = record->Context.Rbp;

#elif \_M\_IX86

auto& \_stackpointer = record->Context.Esp;

auto& \_basepointer = record->Context.Ebp;

#endif

typedef decltype(\_stackpointer) spType;

record->StackDataLength = (spType)\_stackbasepointer - \_stackpointer;

record->StackData = malloc(record->StackDataLength);

if (record->StackData)

{

memcpy(record->StackData, (void\*)\_stackpointer, record->StackDataLength);

\_basepointer = \_basepointer + (spType)record->StackData - \_stackpointer;

\_stackpointer = (spType)record->StackData; // for .cxr switching to this state

}

if (name)

{

wcscpy\_s(record->name, name);

}

record->allocSize = size;

// stack frames

record->FrameCount = CaptureStack(0, MAX\_FRAME\_COUNT, record->StackFrames, 0);

// hash

record->hash = CalculateStackHash(record->StackFrames, record->FrameCount, 2);

fwprintf(stderr, L"\*\*\*FI: Fault Injected, StackHash:%p\n", (void\*)record->hash);

fflush(stderr);

\*InjectionLastRecordRef = record;

InjectionLastRecordRef = &record->next;

InjectionRecordsCount++;

#endif // \_M\_ARM || \_M\_ARM64

}

bool FaultInjection::ShouldInjectFault(FaultType fType, LPCWSTR name, size\_t size)

{

bool shouldInjectionFault = ShouldInjectFaultHelper(fType, name, size);

if (shouldInjectionFault && fType != FaultType::ScriptTerminationOnDispose)

{

dumpCurrentStackData(name, size);

}

return shouldInjectionFault;

}

bool FaultInjection::ShouldInjectFaultHelper(FaultType fType, LPCWSTR name, size\_t size)

{

if (globalFlags.FaultInjection < 0)

{

return false; // no -FaultInjection switch

}

if (globalFlags.FaultInjectionFilter && \_wcsicmp(globalFlags.FaultInjectionFilter, name) != 0)

{

return false;

}

if (globalFlags.FaultInjectionAllocSize >= 0 && size != (size\_t)globalFlags.FaultInjectionAllocSize)

{

return false;

}

// install exception filter to smart dump for faultinjection

// when reproing in debugger, only let debugger catch the exception

// can't do this in ctor because the global flags are not initialized yet

static auto dummy = InstallExceptionFilters();

bool validInjectionPoint = IsFaultEnabled(fType);

if (!validInjectionPoint)

{

return false;

}

bool shouldInjectionFault = false;

switch (globalFlags.FaultInjection)

{

case CountEquals:

//Fault inject on count only when equal

if (countOfInjectionPoints == (uint)globalFlags.FaultInjectionCount)

{

shouldInjectionFault = true;

}

break;

case CountEqualsOrAbove:

//Fault inject on count greater than or equal

if (countOfInjectionPoints >= (uint)globalFlags.FaultInjectionCount)

{

shouldInjectionFault = true;

}

break;

case StackMatch:

// We don't care about the fault if we already passed in terms of count, or the stack doesn't match

if (countOfInjectionPoints > (uint)globalFlags.FaultInjectionCount || !IsCurrentStackMatch())

{

validInjectionPoint = false;

}

else // otherwise determine if we will be injecting this time around

{

shouldInjectionFault = countOfInjectionPoints == (uint)globalFlags.FaultInjectionCount || globalFlags.FaultInjectionCount == -1;

}

break;

case StackMatchCountOnly:

validInjectionPoint = IsCurrentStackMatch();

break;

case StackHashCountOnly:

{

// extend the storage when necessary

if (countOfInjectionPoints > stackHashOfAllInjectionPointsSize)

{

stackHashOfAllInjectionPointsSize += 1024;

auto extended = (ULONG\_PTR\*)realloc(stackHashOfAllInjectionPoints,

stackHashOfAllInjectionPointsSize\*sizeof(ULONG\_PTR));

AssertMsg(extended, "OOM in FaultInjection Infra");

stackHashOfAllInjectionPoints = extended;

}

void\* StackFrames[MAX\_FRAME\_COUNT];

auto FrameCount = CaptureStack(0, MAX\_FRAME\_COUNT, StackFrames, 0);

UINT\_PTR hash = CalculateStackHash(StackFrames, FrameCount, 2);

stackHashOfAllInjectionPoints[countOfInjectionPoints] = hash;

break;

}

case CountOnly:

break;

default:

AssertMsg(false, "Invalid FaultInjection mode");

break;

}

if (validInjectionPoint)

{

countOfInjectionPoints++;

}

// try to lookup stack hash, to see if it matches

if (!shouldInjectionFault)

{

const static UINT\_PTR expectedHash = HexStrToAddress((LPCWSTR)globalFlags.FaultInjectionStackHash);

if (expectedHash != 0)

{

void\* StackFrames[MAX\_FRAME\_COUNT];

auto FrameCount = CaptureStack(0, MAX\_FRAME\_COUNT, StackFrames, 0);

UINT\_PTR hash = CalculateStackHash(StackFrames, FrameCount, 2);

if (hash == expectedHash)

{

shouldInjectionFault = true;

}

}

}

return shouldInjectionFault;

}

// For faster fault injection test run, filter out the AVs on same IP/hash

void FaultInjection::FaultInjetionAnalyzeException(\_EXCEPTION\_POINTERS \*ep)

{

#if !defined(\_M\_ARM32\_OR\_ARM64) // not support ARM for now, add support in case we run fault injection on ARM

AutoCriticalSection autocs(&cs\_Sym);

CONTEXT\* pContext = ep->ContextRecord;

#if \_M\_X64

auto ip = pContext->Rip;

#elif \_M\_IX86

auto ip = pContext->Eip;

#endif

bool needDump = true;

typedef decltype(ip) ipType;

ipType offset = 0;

// static to not use local stack space since stack space might be low at this point

\_\_declspec(thread) static wchar\_t modulePath[MAX\_PATH + 1];

\_\_declspec(thread) static WCHAR filename[MAX\_PATH + 1];

HMODULE mod = nullptr;

GetModuleHandleEx(GET\_MODULE\_HANDLE\_EX\_FLAG\_FROM\_ADDRESS, reinterpret\_cast<LPCTSTR>(ip), &mod);

offset = ip - (ipType)mod;

auto& faultModule = modulePath;

GetModuleFileName(mod, faultModule, MAX\_PATH);

fwprintf(stderr, L"\*\*\*FI: Exception: %08x, module: %s, offset: 0x%p\n",

ep->ExceptionRecord->ExceptionCode, faultModule, (void\*)offset);

//analyze duplication

ipType savedOffset = 0;

auto& mainModule = modulePath;

GetModuleFileName(NULL, mainModule, MAX\_PATH);

// multiple session of Fault Injection run shares the single crash offset recording file

\_snwprintf\_s(filename, \_TRUNCATE, L"%s.FICrashes.txt", mainModule);

auto fp = \_wfsopen(filename, L"a+t", \_SH\_DENYNO);

if (fp != nullptr)

{

HANDLE hFile = (HANDLE)\_get\_osfhandle(\_fileno(fp));

OVERLAPPED overlapped;

memset(&overlapped, 0, sizeof(overlapped));

const int lockSize = 1024 \* 64;

if (!LockFileEx(hFile, LOCKFILE\_EXCLUSIVE\_LOCK, 0, lockSize, 0, &overlapped))

{

fwprintf(stderr, L"LockFileEx(%ls) Failed when saving offset to file, gle=%8x\n", filename, GetLastError());

fclose(fp);

}

else

{ // file locked

wchar\_t content[32] = { 0 };

while (fgetws(content, 31, fp))

{

savedOffset = HexStrToAddress(content);

if (offset == savedOffset)

{

// found duplicate so not creating dump

needDump = false;

}

}

if (needDump)

{

fwprintf(stderr, L"This is new Exception\n");

fwprintf(fp, L"0x%p\n", (void\*)offset);

}

else

{

fwprintf(stderr, L"This is not a new Exception\n");

}

fflush(fp);

// save the hit count to a file, for bug prioritizing

\_snwprintf\_s(filename, \_TRUNCATE, L"%s.HitCount\_%llx.txt", mainModule, (long long)offset);

auto hcfp = \_wfsopen(filename, L"r+", \_SH\_DENYNO);

if (!hcfp)

{

hcfp = \_wfsopen(filename, L"w+", \_SH\_DENYNO);

}

if (hcfp)

{

auto count = 0;

fscanf\_s(hcfp, "%d", &count);

count++;

fseek(hcfp, -ftell(hcfp), SEEK\_CUR);

fwprintf(hcfp, L"%d", count);

fclose(hcfp);

}

fclose(fp);

UnlockFileEx(hFile, 0, lockSize, 0, &overlapped);

}

fflush(stderr);

}

// create dump for this crash

if (needDump)

{

\_\_declspec(thread) static wchar\_t dumpName[MAX\_PATH + 1];

wcscpy\_s(filename, globalFlags.Filename);

wchar\_t\* jsFile = filename;

wchar\_t \*pch = jsFile;

// remove path and keep only alphabet and number to make a valid filename

while (\*pch)

{

if (\*pch == L':' || \*pch == L'\\')

{

jsFile = pch + 1;

}

else if (!isalnum(\*pch))

{

\*pch = L'\_';

}

pch++;

}

// get dump file name

int suffix = 1;

const wchar\_t\* fiType = L"undefined";

if (globalFlags.FaultInjectionType != nullptr)

{

fiType = (LPCWSTR)globalFlags.FaultInjectionType;

}

while (true)

{

\_snwprintf\_s(dumpName, \_TRUNCATE, L"%s\_%s\_M%d\_T%s\_C%d\_%llx\_%llx\_%d.dmp",

mainModule, jsFile,

globalFlags.FaultInjection, fiType, globalFlags.FaultInjectionCount,

(ULONGLONG)offset, (ULONGLONG)ep->ExceptionRecord->ExceptionCode, suffix);

WIN32\_FIND\_DATAW data;

HANDLE hExist = FindFirstFile(dumpName, &data);

if (hExist == INVALID\_HANDLE\_VALUE)

{

FindClose(hExist);

break;

}

FindClose(hExist);

suffix++;

}

// writing the dump file

HANDLE hFile = CreateFile(dumpName, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if ((hFile == NULL) || (hFile == INVALID\_HANDLE\_VALUE))

{

fwprintf(stderr, L"CreateFile <%s> failed. gle=0x%08x\n", dumpName, GetLastError());

}

else

{

MINIDUMP\_EXCEPTION\_INFORMATION mdei;

mdei.ThreadId = GetCurrentThreadId();

mdei.ExceptionPointers = ep;

mdei.ClientPointers = FALSE;

MINIDUMP\_TYPE mdt = (MINIDUMP\_TYPE)(MiniDumpNormal

| MiniDumpWithFullMemory

| MiniDumpWithPrivateReadWriteMemory

| MiniDumpWithIndirectlyReferencedMemory

| MiniDumpWithThreadInfo);

// removing extension for windbg module name style

auto& jscript9Path = modulePath;

wcsncpy\_s(jscript9Path, AutoSystemInfo::Data.GetJscriptDllFileName(),

wcslen(AutoSystemInfo::Data.GetJscriptDllFileName()) - 4);

wchar\_t\* jscript9Name = jscript9Path + wcslen(jscript9Path);

while (\*(jscript9Name - 1) != L'\\' && jscript9Name > jscript9Path)

{

jscript9Name--;

}

// This buffer will be written to a dump stream when creating the minidump file.

// It contains windbg debugging instructions on how to figure out the injected faults,

// And the message will be showing in windbg while loading the minidump.

// If you need to add more instructions please increase the buffer capacity accordingly

\_\_declspec(thread) static wchar\_t dbgTip[1024];

if (InjectionFirstRecord == nullptr)

{

wcsncpy\_s(dbgTip,

L"\n"

L"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

L"\* The dump is made by FaultInjection framework, however, the fault is not actually injected yet.\n"

L"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", \_TRUNCATE);

}

else

{

\_snwprintf\_s(dbgTip, \_TRUNCATE, L"\n"

L"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

L"\* To find the Fault Injecting points run following command: \n"

L"\* !list -t %s!Js::FaultInjection::InjectionRecord.next -e -x \"dps @$extret @$extret+0x128\" poi(@@c++(&%s!Js::FaultInjection::Global.InjectionFirstRecord))\n"

L"\* To rebuild the stack (locals are available):\n"

L"\* .cxr @@C++(&%s!Js::FaultInjection::Global.InjectionFirstRecord->Context)\n"

L"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", jscript9Name, jscript9Name, jscript9Name);

}

MINIDUMP\_USER\_STREAM UserStreams[1];

UserStreams[0].Type = CommentStreamW;

UserStreams[0].Buffer = dbgTip;

UserStreams[0].BufferSize = (ULONG)wcslen(dbgTip)\*sizeof(wchar\_t);

MINIDUMP\_USER\_STREAM\_INFORMATION musi;

musi.UserStreamCount = 1;

musi.UserStreamArray = UserStreams;

BOOL rv = pfnMiniDumpWriteDump(GetCurrentProcess(), GetCurrentProcessId(), hFile, mdt, (ep != 0) ? &mdei : 0, &musi, 0);

if (rv)

{

fwprintf(stderr, L"Minidump created: %s\n", dumpName);

}

else

{

fwprintf(stderr, L"MiniDumpWriteDump failed. gle=0x%08x\n", GetLastError());

}

CloseHandle(hFile);

}

}

// always show stack for crash and fault injection points in console,

// this can be used for additional stack matching repro

auto printFrame = [&](LPVOID addr)

{

HANDLE hProcess = GetCurrentProcess();

DWORD64 dwSymDisplacement = 0;

sip.Init();

if (pfnSymFromAddrW(hProcess, (DWORD64)addr, &dwSymDisplacement, &sip.si))

{

mi.Init();

pfnSymGetModuleInfoW64(hProcess, (DWORD64)addr, &mi);

fwprintf(stderr, L"%s!%s+0x%llx\n", mi.ModuleName, sip.si.Name, (ULONGLONG)dwSymDisplacement);

}

else

{

fwprintf(stderr, L"0x%p\n", addr);

}

};

LPVOID backTrace[MAX\_FRAME\_COUNT];

#if \_M\_IX86

WORD nStackCount = StackTrace86(0, MAX\_FRAME\_COUNT, backTrace, 0, pContext);

#elif \_M\_X64

WORD nStackCount = StackTrace64(0, MAX\_FRAME\_COUNT, backTrace, 0, pContext);

#else

WORD nStackCount = CaptureStack(0, MAX\_FRAME\_COUNT, backTrace, 0);

#endif

// Print current crash stacks

fwprintf(stderr, crashStackStart);

//bool foundFaultIP = false;

for (int i = 0; i< nStackCount; i++)

{

printFrame(backTrace[i]);

}

fwprintf(stderr, crashStackEnd);

// Print fault injecting point stacks

auto record = InjectionFirstRecord;

while (record)

{

if (record->StackFrames)

{

fwprintf(stderr, injectionStackStart);

for (int i = 0; i < record->FrameCount; i++)

{

printFrame(record->StackFrames[i]);

}

fwprintf(stderr, injectionStackEnd);

}

record = record->next;

}

fflush(stderr);

#endif //\_M\_ARM and \_M\_ARM64

}

LONG WINAPI FaultInjection::FaultInjectionExceptionFilter(\_In\_ struct \_EXCEPTION\_POINTERS \*ExceptionInfo)

{

RemoveExceptionFilters();

// for debugging, can't hit here in windbg because of using vectored exception handling

if (faultInjectionDebug)

{

DebugBreak();

}

FaultInjection::Global.FaultInjetionAnalyzeException(ExceptionInfo);

return EXCEPTION\_EXECUTE\_HANDLER;

}

} //namespace Js

#endif //FAULT\_INJECTION

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

#ifdef FAULT\_INJECTION

class FaultInjection

{

public:

static const unsigned int MAX\_FRAME\_COUNT = 64;

// Fault types

#define FAULT\_TYPE(x) x, \

enum FaultType

{

#include "FaultTypes.h"

FaultTypeCount,

InvalidFaultType

};

#undef FAULT\_TYPE

// use bit array to save the enabled type

class FaultInjectionTypes

{

private:

char faultTypeBitArray[InvalidFaultType/8+1];

char getBit(int index) {

return (faultTypeBitArray[index/8] >> (7-(index & 0x7))) & 0x1;

}

void setBit(int index, int value) {

faultTypeBitArray[index/8] = faultTypeBitArray[index/8] | (value & 0x1) << (7-(index & 0x7));

}

public:

FaultInjectionTypes(){

memset(&faultTypeBitArray, 0, sizeof(faultTypeBitArray));

}

void EnableAll(){

memset(&faultTypeBitArray, ~0, sizeof(faultTypeBitArray));

}

void EnableType(FaultType type);

void EnableType(int type){

EnableType((FaultType)type);

}

bool IsEnabled(FaultType type);

bool IsEnabled(const wchar\_t\* name);

};

static wchar\_t \*FaultTypeNames[];

void ParseFaultTypes(const wchar\_t\* szFaultTypes);

public:

enum FaultMode

{

CountOnly = 0,

CountEquals = 1,

CountEqualsOrAbove = 2,

StackMatch = 3,

StackMatchCountOnly = 4,

StackHashCountOnly = 5,

DisplayAvailableFaultTypes = 6,

};

uint countOfInjectionPoints;

int FaultInjectionCookie;

enum StackMatchInitializationState

{

Uninitialized = 0,

FailedToInitialize = 1,

Succeeded = 2

};

StackMatchInitializationState stackMatchInitialized; // tri-state: 0-uninitialized, 1-tried to init, 2-initialized

private:

HMODULE hDbgHelp;

bool InitializeSym();

FaultInjectionTypes\* faultInjectionTypes;

bool IsCurrentStackMatch();

bool EnsureStackMatchInfraInitialized();

uint baselineFrameCount;

wchar\_t \*baselineStack[MAX\_FRAME\_COUNT];

UINT\_PTR baselineAddresses[MAX\_FRAME\_COUNT];

ULONG\_PTR\* stackHashOfAllInjectionPoints;

UINT stackHashOfAllInjectionPointsSize;

public:

static FaultInjection Global;

FaultInjection();

~FaultInjection();

bool IsFaultEnabled(FaultType faultType);

bool IsFaultInjectionOn(FaultType faultType);

bool ShouldInjectFault(FaultType fType, LPCWSTR name = nullptr, size\_t size = 0);// name and size are used for OOM only

// sample for customized fault type

template<class Pred>

bool ShouldInjectFault(FaultType fType, Pred p) {

bool shouldInjectionFault = Js::Configuration::Global.flags.FaultInjectionCount == 0

|| ShouldInjectFaultHelper(fType);

if (shouldInjectionFault && p()) {

if(IsDebuggerPresent()) {

DebugBreak();

}

dumpCurrentStackData();

}

return shouldInjectionFault;

}

private:

bool ShouldInjectFaultHelper(FaultType fType, LPCWSTR name = nullptr, size\_t size = 0);

private:

// for reconstruction stack of the fault injection points in postmortem debugging

struct InjectionRecord{

void\* StackFrames[MAX\_FRAME\_COUNT];

UINT\_PTR hash;

WORD FrameCount;

void\* StackData;

size\_t StackDataLength;

CONTEXT Context;

WCHAR name[32];

size\_t allocSize;

InjectionRecord\* next;

};

public:

InjectionRecord\* InjectionFirstRecord;

InjectionRecord\*\* InjectionLastRecordRef;

int InjectionRecordsCount;

void dumpCurrentStackData(LPCWSTR name = nullptr, size\_t size = 0);

static \_\_declspec(thread) int(\*pfnHandleAV)(int, PEXCEPTION\_POINTERS);

private:

bool symInitialized;

static PVOID vectoredExceptionHandler;

static DWORD exceptionFilterRemovalLastError;

static bool InstallExceptionFilters();

static void RemoveExceptionFilters();

static UINT\_PTR CalculateStackHash(void\* frames[], WORD frameCount, WORD framesToSkip);

static LONG WINAPI FaultInjectionExceptionFilter(\_In\_ struct \_EXCEPTION\_POINTERS \*ExceptionInfo);

void FaultInjetionAnalyzeException(\_EXCEPTION\_POINTERS \*ep);

};

#endif

} // namespace Js

#ifdef FAULT\_INJECTION

#define IS\_FAULTINJECT\_NO\_THROW\_ON \

Js::FaultInjection::Global.IsFaultInjectionOn(Js::FaultInjection::Global.NoThrow)

#define FAULTINJECT\_MEMORY\_NOTHROW(name, size) \

if(Js::FaultInjection::Global.ShouldInjectFault(Js::FaultInjection::Global.NoThrow, name, size)) \

return NULL;

#define FAULTINJECT\_MEMORY\_THROW(name, size) \

if(Js::FaultInjection::Global.ShouldInjectFault(Js::FaultInjection::Global.Throw, name, size)) \

Js::Throw::OutOfMemory();

#define FAULTINJECT\_MEMORY\_MARK\_THROW(name, size) \

if(Js::FaultInjection::Global.ShouldInjectFault(Js::FaultInjection::Global.MarkThrow, name, size)) { \

Js::Throw::OutOfMemory(); \

}

#define FAULTINJECT\_MEMORY\_MARK\_NOTHROW(name, size) \

if(Js::FaultInjection::Global.ShouldInjectFault(Js::FaultInjection::Global.MarkNoThrow, name, size)) { \

return false; \

}

#define FAULTINJECT\_SCRIPT\_TERMINATION \

if((this->threadContextFlags & ThreadContextFlagCanDisableExecution) != 0){ \

if( Js::FaultInjection::Global.ShouldInjectFault(Js::FaultInjection::Global.ScriptTermination)){ \

this->stackLimitForCurrentThread = Js::Constants::StackLimitForScriptInterrupt; \

}\

}

#define FAULTINJECT\_STACK\_PROBE \

if( Js::FaultInjection::Global.ShouldInjectFault(Js::FaultInjection::Global.StackProbe)){ \

stackAvailable = false; \

}

#define IS\_FAULTINJECT\_STACK\_PROBE\_ON \

Js::FaultInjection::Global.IsFaultInjectionOn(Js::FaultInjection::Global.StackProbe)

#define FAULTINJECT\_SCRIPT\_TERMINATION\_ON\_DISPOSE \

Js::FaultInjection::Global.ShouldInjectFault(Js::FaultInjection::Global.ScriptTerminationOnDispose)

// A general implementation of customized fault type injection

#define INJECT\_FAULT(type, condition, execution) \

do{\

if(Js::FaultInjection::Global.ShouldInjectFault(type, condition)) {\

##execution##\

};\

}while(0)

#else

#define IS\_FAULTINJECT\_NO\_THROW\_ON false

#define FAULTINJECT\_MEMORY\_NOTHROW(name, size)

#define FAULTINJECT\_MEMORY\_THROW(name, size)

#define FAULTINJECT\_MEMORY\_MARK\_THROW(name, size)

#define FAULTINJECT\_MEMORY\_MARK\_NOTHROW(name, size)

#define FAULTINJECT\_SCRIPT\_TERMINATION

#define FAULTINJECT\_SCRIPT\_TERMINATION\_ON\_DISPOSE false

#define FAULTINJECT\_STACK\_PROBE

#define IS\_FAULTINJECT\_STACK\_PROBE\_ON false

#define INJECT\_FAULT(type, condition, execution)

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#ifndef FAULT\_TYPE

#error FAULT\_TYPE not defined

#endif

FAULT\_TYPE(Throw)

FAULT\_TYPE(NoThrow)

FAULT\_TYPE(MarkThrow)

FAULT\_TYPE(MarkNoThrow)

FAULT\_TYPE(StackProbe)

FAULT\_TYPE(ScriptTermination)

FAULT\_TYPE(ScriptTerminationOnDispose)

FAULT\_TYPE(FaultInjectioSelfTest)

// custom fault types

// e.g. FaultInterpretThunk,

FAULT\_TYPE(EnumFields\_Fail)

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

class FinalizableObject

{

public:

// Called right after finish marking and this object is determined to be dead.

// Should contain only simple clean up code.

// Can't run another script

// Can't cause a re-entrant collection

virtual void Finalize(bool isShutdown) = 0;

// Call after sweeping is done.

// Can call other script or cause another collection.

virtual void Dispose(bool isShutdown) = 0;

// Used only by TrackableObjects (created with TrackedBit on by RecyclerNew\*Tracked)

virtual void Mark(Recycler \* recycler) = 0;

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

interface ICmdLineArgsParser

{

virtual BSTR GetCurrentString() = 0;

virtual bool GetCurrentBoolean() = 0;

virtual int GetCurrentInt() = 0;

};

interface ICustomConfigFlags

{

virtual void PrintUsage() = 0;

virtual bool ParseFlag(LPCWSTR flagsString, ICmdLineArgsParser \* parser) = 0;

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#include <string.h>

#include <stdarg.h>

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAM")

bool Output::s\_useDebuggerWindow = false;

CriticalSection Output::s\_critsect;

AutoFILE Output::s\_outputFile; // Create a separate output file that is not thread-local.

#ifdef ENABLE\_TRACE

Js::ILogger\* Output::s\_inMemoryLogger = nullptr;

Js::IStackTraceHelper\* Output::s\_stackTraceHelper = nullptr;

unsigned int Output::s\_traceEntryId = 0;

#endif

THREAD\_ST FILE\* Output::s\_file = nullptr;

THREAD\_ST wchar\_t\* Output::buffer = nullptr;

THREAD\_ST size\_t Output::bufferAllocSize = 0;

THREAD\_ST size\_t Output::bufferFreeSize = 0;

THREAD\_ST size\_t Output::s\_Column = 0;

THREAD\_ST WORD Output::s\_color = 0;

THREAD\_ST bool Output::s\_hasColor = false;

THREAD\_ST bool Output::s\_capture = false;

#define MAX\_OUTPUT\_BUFFER\_SIZE 10 \* 1024 \* 1024 // 10 MB maximum before we force a flush

size\_t \_\_cdecl

Output::VerboseNote(const wchar\_t \* format, ...)

{

#ifdef ENABLE\_TRACE

if (Js::Configuration::Global.flags.Verbose)

{

AutoCriticalSection autocs(&s\_critsect);

va\_list argptr;

va\_start(argptr, format);

size\_t size = vfwprintf(stdout, format, argptr);

va\_end(argptr);

return size;

}

#endif

return 0;

}

#ifdef ENABLE\_TRACE

size\_t \_\_cdecl

Output::Trace(Js::Phase phase, const wchar\_t \*form, ...)

{

size\_t retValue = 0;

if(Js::Configuration::Global.flags.Trace.IsEnabled(phase))

{

va\_list argptr;

va\_start(argptr, form);

retValue += Output::VTrace(L"%s: ", Js::PhaseNames[static\_cast<int>(phase)], form, argptr);

}

return retValue;

}

size\_t \_\_cdecl

Output::Trace2(Js::Phase phase, const wchar\_t \*form, ...)

{

size\_t retValue = 0;

if (Js::Configuration::Global.flags.Trace.IsEnabled(phase))

{

va\_list argptr;

va\_start(argptr, form);

retValue += Output::VPrint(form, argptr);

}

return retValue;

}

size\_t \_\_cdecl

Output::TraceWithPrefix(Js::Phase phase, const wchar\_t prefix[], const wchar\_t \*form, ...)

{

size\_t retValue = 0;

if (Js::Configuration::Global.flags.Trace.IsEnabled(phase))

{

va\_list argptr;

va\_start(argptr, form);

WCHAR prefixValue[512];

swprintf\_s(prefixValue, L"%s: %s: ", Js::PhaseNames[static\_cast<int>(phase)], prefix);

retValue += Output::VTrace(L"%s", prefixValue, form, argptr);

}

return retValue;

}

size\_t \_\_cdecl

Output::TraceWithFlush(Js::Phase phase, const wchar\_t \*form, ...)

{

size\_t retValue = 0;

if(Js::Configuration::Global.flags.Trace.IsEnabled(phase))

{

va\_list argptr;

va\_start(argptr, form);

retValue += Output::VTrace(L"%s:", Js::PhaseNames[static\_cast<int>(phase)], form, argptr);

Output::Flush();

}

return retValue;

}

size\_t \_\_cdecl

Output::TraceWithFlush(Js::Flag flag, const wchar\_t \*form, ...)

{

size\_t retValue = 0;

if (Js::Configuration::Global.flags.IsEnabled(flag))

{

va\_list argptr;

va\_start(argptr, form);

retValue += Output::VTrace(L"[-%s]::", Js::FlagNames[static\_cast<int>(flag)], form, argptr);

Output::Flush();

}

return retValue;

}

size\_t

Output::VTrace(const wchar\_t\* shortPrefixFormat, const wchar\_t\* prefix, const wchar\_t \*form, va\_list argptr)

{

size\_t retValue = 0;

if (CONFIG\_FLAG(RichTraceFormat))

{

InterlockedIncrement(&s\_traceEntryId);

retValue += Output::Print(L"[%d ~%d %s] ", s\_traceEntryId, ::GetCurrentThreadId(), prefix);

}

else

{

retValue += Output::Print(shortPrefixFormat, prefix);

}

retValue += Output::VPrint(form, argptr);

// Print stack trace.

if (s\_stackTraceHelper)

{

const ULONG c\_framesToSkip = 2; // Skip 2 frames -- Output::VTrace and Output::Trace.

const ULONG c\_frameCount = 10; // TODO: make it configurable.

const wchar\_t callStackPrefix[] = L"call stack:";

if (s\_inMemoryLogger)

{

// Trace just addresses of functions, avoid symbol info as it takes too much memory.

// One line for whole stack trace for easier parsing on the jd side.

const size\_t c\_msgCharCount = \_countof(callStackPrefix) + (1 + sizeof(void\*) \* 2) \* c\_frameCount; // 2 hexadecimal digits per byte + 1 for space.

wchar\_t callStackMsg[c\_msgCharCount];

void\* frames[c\_frameCount];

size\_t start = 0;

size\_t temp;

temp = \_snwprintf\_s(callStackMsg, \_countof(callStackMsg), \_TRUNCATE, L"%s", callStackPrefix);

Assert(temp != -1);

start += temp;

ULONG framesObtained = s\_stackTraceHelper->GetStackTrace(c\_framesToSkip, c\_frameCount, frames);

Assert(framesObtained <= c\_frameCount);

for (ULONG i = 0; i < framesObtained && i < c\_frameCount; ++i)

{

Assert(\_countof(callStackMsg) >= start);

temp = \_snwprintf\_s(callStackMsg + start, \_countof(callStackMsg) - start, \_TRUNCATE, L" %p", frames[i]);

Assert(temp != -1);

start += temp;

}

retValue += Output::Print(L"%s\n", callStackMsg);

}

else

{

// Trace with full symbol info.

retValue += Output::Print(L"%s\n", callStackPrefix);

retValue += s\_stackTraceHelper->PrintStackTrace(c\_framesToSkip, c\_frameCount);

}

}

return retValue;

}

#ifdef BGJIT\_STATS

size\_t \_\_cdecl

Output::TraceStats(Js::Phase phase, const wchar\_t \*form, ...)

{

if(PHASE\_STATS1(phase))

{

va\_list argptr;

va\_start(argptr, form);

return Output::VPrint(form, argptr);

}

return 0;

}

#endif

#endif ENABLE\_TRACE

///----------------------------------------------------------------------------

///

/// Output::Print

///

/// Print the given format string.

///

///

///----------------------------------------------------------------------------

size\_t \_\_cdecl

Output::Print(const wchar\_t \*form, ...)

{

va\_list argptr;

va\_start(argptr, form);

return Output::VPrint(form, argptr);

}

size\_t \_\_cdecl

Output::Print(int column, const wchar\_t \*form, ...)

{

Output::SkipToColumn(column);

va\_list argptr;

va\_start(argptr, form);

return Output::VPrint(form, argptr);

}

size\_t \_\_cdecl

Output::VPrint(const wchar\_t \*form, va\_list argptr)

{

wchar\_t buf[2048];

size\_t size;

size = \_vsnwprintf\_s(buf, \_countof(buf), \_TRUNCATE, form, argptr);

if(size == -1)

{

size = 2048;

}

return Output::PrintBuffer(buf, size);

}

size\_t \_\_cdecl

Output::PrintBuffer(const wchar\_t \* buf, size\_t size)

{

Output::s\_Column += size;

const wchar\_t \* endbuf = wcschr(buf, '\n');

while (endbuf != nullptr)

{

Output::s\_Column = size - (endbuf - buf) - 1;

endbuf = wcschr(endbuf + 1, '\n');

}

bool useConsoleOrFile = true;

if (!Output::s\_capture)

{

if (Output::s\_useDebuggerWindow)

{

OutputDebugStringW(buf);

useConsoleOrFile = false;

}

#ifdef ENABLE\_TRACE

if (Output::s\_inMemoryLogger)

{

s\_inMemoryLogger->Write(buf);

useConsoleOrFile = false;

}

#endif

}

if (useConsoleOrFile)

{

if (s\_file == nullptr || Output::s\_capture)

{

bool addToBuffer = true;

if (Output::bufferFreeSize < size + 1)

{

if (Output::bufferAllocSize > MAX\_OUTPUT\_BUFFER\_SIZE && !Output::s\_capture)

{

Output::Flush();

if (Output::bufferFreeSize < size + 1)

{

DirectPrint(buf);

addToBuffer = false;

}

}

else

{

size\_t oldBufferSize = bufferAllocSize - bufferFreeSize;

size\_t newBufferAllocSize = (bufferAllocSize + size + 1) \* 4 / 3;

wchar\_t \* newBuffer = (wchar\_t \*)realloc(buffer, (newBufferAllocSize \* sizeof(wchar\_t)));

if (newBuffer == nullptr)

{

// See if I can just flush it and print directly

Output::Flush();

// Reset the buffer

free(Output::buffer);

Output::buffer = nullptr;

Output::bufferAllocSize = 0;

Output::bufferFreeSize = 0;

// Print it directly

DirectPrint(buf);

addToBuffer = false;

}

else

{

bufferAllocSize = newBufferAllocSize;

buffer = newBuffer;

bufferFreeSize = bufferAllocSize - oldBufferSize;

}

}

}

if (addToBuffer)

{

Assert(Output::bufferFreeSize >= size + 1);

memcpy\_s(Output::buffer + Output::bufferAllocSize - Output::bufferFreeSize, Output::bufferFreeSize \* sizeof(wchar\_t),

buf, (size + 1) \* sizeof(wchar\_t));

bufferFreeSize -= size;

}

}

else

{

fwprintf\_s(Output::s\_file, L"%s", buf);

}

if(s\_outputFile != nullptr && !Output::s\_capture)

{

fwprintf\_s(s\_outputFile, L"%s", buf);

}

}

if (IsDebuggerPresent())

{

Output::Flush();

}

return size;

}

void Output::Flush()

{

if (s\_capture)

{

return;

}

if (bufferFreeSize != bufferAllocSize)

{

DirectPrint(Output::buffer);

bufferFreeSize = bufferAllocSize;

}

if(s\_outputFile != nullptr)

{

fflush(s\_outputFile);

}

\_flushall();

}

void Output::DirectPrint(wchar\_t const \* string)

{

AutoCriticalSection autocs(&s\_critsect);

WORD oldValue = 0;

BOOL restoreColor = FALSE;

HANDLE hConsole = NULL;

if (Output::s\_hasColor)

{

\_CONSOLE\_SCREEN\_BUFFER\_INFO info;

hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (hConsole && GetConsoleScreenBufferInfo(hConsole, &info))

{

oldValue = info.wAttributes;

restoreColor = SetConsoleTextAttribute(hConsole, Output::s\_color);

}

}

fwprintf(stdout, L"%s", string);

if (restoreColor)

{

SetConsoleTextAttribute(hConsole, oldValue);

}

}

///----------------------------------------------------------------------------

///

/// Output::SkipToColumn

///

/// Inserts spaces up to the column passed in.

///

///----------------------------------------------------------------------------

void

Output::SkipToColumn(size\_t column)

{

if (column <= Output::s\_Column)

{

Output::Print(L" ");

return;

}

// compute distance to our destination

size\_t dist = column - Output::s\_Column;

// Print at least one space

while (dist > 0)

{

Output::Print(L" ");

dist--;

}

}

FILE\*

Output::GetFile()

{

return Output::s\_file;

}

FILE\*

Output::SetFile(FILE \*file)

{

Output::Flush();

FILE \*oldfile = Output::s\_file;

Output::s\_file = file;

return oldfile;

}

void

Output::SetOutputFile(FILE\* file)

{

if(s\_outputFile != nullptr)

{

AssertMsg(false, "Output file is being set twice.");

}

else

{

s\_outputFile = file;

}

}

FILE\*

Output::GetOutputFile()

{

return s\_outputFile;

}

#ifdef ENABLE\_TRACE

void

Output::SetInMemoryLogger(Js::ILogger\* logger)

{

AssertMsg(s\_inMemoryLogger == nullptr, "This cannot be called more than once.");

s\_inMemoryLogger = logger;

}

void

Output::SetStackTraceHelper(Js::IStackTraceHelper\* helper)

{

AssertMsg(s\_stackTraceHelper == nullptr, "This cannot be called more than once.");

#ifndef STACK\_BACK\_TRACE

AssertMsg("STACK\_BACK\_TRACE must be defined");

#endif

s\_stackTraceHelper = helper;

}

#endif ENABLE\_TRACE

//

// Sets the foreground color and returns the old color. Returns 0 on failure

//

WORD

Output::SetConsoleForeground(WORD color)

{

AutoCriticalSection autocs(&s\_critsect);

\_CONSOLE\_SCREEN\_BUFFER\_INFO info;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (hConsole && GetConsoleScreenBufferInfo(hConsole, &info))

{

Output::Flush();

Output::s\_color = color | (info.wAttributes & ~15);

Output::s\_hasColor = Output::s\_color != info.wAttributes;

return info.wAttributes;

}

return 0;

}

void

Output::CaptureStart()

{

Assert(!s\_capture);

Output::Flush();

s\_capture = true;

}

wchar\_t \*

Output::CaptureEnd()

{

Assert(s\_capture);

s\_capture = false;

bufferFreeSize = 0;

bufferAllocSize = 0;

wchar\_t \* returnBuffer = buffer;

buffer = nullptr;

return returnBuffer;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

enum Flag;

enum Phase;

};

#if defined(ENABLE\_DEBUG\_CONFIG\_OPTIONS) && defined(BGJIT\_STATS)

#define OUTPUT\_TRACE(Phase, ...) Output::Trace((Phase), \_\_VA\_ARGS\_\_)

#define OUTPUT\_TRACE\_2(Phase, ...) Output::Trace2((Phase), \_\_VA\_ARGS\_\_)

#define OUTPUT\_VERBOSE\_TRACE(Phase, ...) \

if(Js::Configuration::Global.flags.Verbose) \

{ \

OUTPUT\_TRACE((Phase), \_\_VA\_ARGS\_\_); \

}

#define OUTPUT\_STATS(Phase, ...) Output::TraceStats((Phase), \_\_VA\_ARGS\_\_)

#define OUTPUT\_VERBOSE\_STATS(Phase, ...) \

if(Js::Configuration::Global.flags.Verbose) \

{ \

OUTPUT\_STATS((Phase), \_\_VA\_ARGS\_\_); \

}

#define OUTPUT\_FLUSH() Output::Flush()

#else

#define OUTPUT\_TRACE(Phase, ...)

#define OUTPUT\_TRACE\_2(Phase, ...)

#define OUTPUT\_VERBOSE\_TRACE(Phase, ...)

#define OUTPUT\_STATS(Phase, ...)

#define OUTPUT\_VERBOSE\_STATS(Phase, ...)

#define OUTPUT\_FLUSH()

#endif

#if DBG

#define OUTPUT\_TRACE\_DEBUGONLY\_ENABLED 1

#define OUTPUT\_TRACE\_DEBUGONLY(Phase, ...) Output::TraceWithFlush((Phase), \_\_VA\_ARGS\_\_)

#define OUTPUT\_TRACE\_DEBUGONLY(Flag, ...) Output::TraceWithFlush((Flag), \_\_VA\_ARGS\_\_)

#else

#define OUTPUT\_TRACE\_DEBUGONLY(Phase, ...)

#define OUTPUT\_TRACE\_FLAG\_DEBUGONLY(Flag, ...)

#endif

namespace Js

{

// Logging interfaces:

// decouple implementation so that in common.lib we don't have dependency on memory.lib

struct ILogger

{

virtual void Write(const wchar\_t\* msg) = 0;

};

struct IStackTraceHelper

{

virtual size\_t PrintStackTrace(ULONG framesToSkip, ULONG framesToCapture) = 0; // Returns # of chars printed.

virtual ULONG GetStackTrace(ULONG framesToSkip, ULONG framesToCapture, void\*\* stackFrames) = 0; // Returns # of frames captured.

};

} // namespace Js.

class Output

{

public:

static size\_t \_\_cdecl VerboseNote(const wchar\_t \* format, ...);

#ifdef ENABLE\_TRACE

static size\_t \_\_cdecl Trace(Js::Phase phase, const wchar\_t \*form, ...);

static size\_t \_\_cdecl Trace2(Js::Phase phase, const wchar\_t \*form, ...);

static size\_t \_\_cdecl TraceWithPrefix(Js::Phase phase, const wchar\_t prefix[], const wchar\_t \*form, ...);

static size\_t \_\_cdecl TraceWithFlush(Js::Phase phase, const wchar\_t \*form, ...);

static size\_t \_\_cdecl TraceWithFlush(Js::Flag flag, const wchar\_t \*form, ...);

static size\_t \_\_cdecl TraceStats(Js::Phase phase, const wchar\_t \*form, ...);

template<class Fn>

static size\_t \_\_cdecl

TraceWithCallback(Js::Phase phase, Fn callback, const wchar\_t \*form, ...)

{

size\_t retValue = 0;

if(Js::Configuration::Global.flags.Trace.IsEnabled(phase))

{

va\_list argptr;

va\_start(argptr, form);

retValue = Output::Print(L"%s:", Js::PhaseNames[static\_cast<int>(phase)]);

retValue += Output::VPrint(form, argptr);

retValue += Output::Print(L"%s", callback());

}

return retValue;

}

static void SetInMemoryLogger(Js::ILogger\* logger);

static void SetStackTraceHelper(Js::IStackTraceHelper\* helper);

#endif ENABLE\_TRACE

static size\_t \_\_cdecl Print(const wchar\_t \*form, ...);

static size\_t \_\_cdecl Print(int column, const wchar\_t \*form, ...);

static size\_t \_\_cdecl PrintBuffer(const wchar\_t \* buffer, size\_t size);

static size\_t \_\_cdecl VPrint(const wchar\_t \*form, va\_list argptr);

static void SkipToColumn(size\_t column);

static FILE\* SetFile(FILE \*);

static FILE\* GetFile();

static void SetOutputFile(FILE \*);

static FILE\* GetOutputFile();

static void UseDebuggerWindow() { s\_useDebuggerWindow = true; }

static void Flush();

static WORD SetConsoleForeground(WORD color);

static void CaptureStart();

static wchar\_t\* CaptureEnd();

private:

static void DirectPrint(const wchar\_t \* string);

static AutoFILE s\_outputFile;

static bool s\_useDebuggerWindow;

static CriticalSection s\_critsect;

#ifdef ENABLE\_TRACE

static Js::ILogger\* s\_inMemoryLogger; // Used to trace into memory so that when process crashes, you can see tracing in crash dump file.

static unsigned int s\_traceEntryId; // Sequential id of trace entry for rich output format.

#ifdef STACK\_BACK\_TRACE

static Js::IStackTraceHelper\* s\_stackTraceHelper; // Used for capturing stack trace.

#endif

static size\_t VTrace(const wchar\_t\* shortPrefixFormat, const wchar\_t\* prefix, const wchar\_t \*form, va\_list argptr);

#endif ENABLE\_TRACE

#define THREAD\_ST \_\_declspec(thread)

THREAD\_ST static bool s\_capture;

THREAD\_ST static FILE \* s\_file;

THREAD\_ST static wchar\_t \* buffer;

THREAD\_ST static size\_t bufferFreeSize;

THREAD\_ST static size\_t bufferAllocSize;

THREAD\_ST static size\_t s\_Column;

THREAD\_ST static WORD s\_color;

THREAD\_ST static bool s\_hasColor;

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#include "PerfCounterImpl.cpp"

#ifdef PERF\_COUNTERS

namespace PerfCounter

{

Counter& Counter::operator+=(size\_t value)

{

Assert(count);

::InterlockedExchangeAdd(count, (DWORD)value);

return \*this;

}

Counter& Counter::operator-=(size\_t value)

{

Assert(count);

::InterlockedExchangeSubtract(count, (DWORD)value);

return \*this;

}

Counter& Counter::operator++()

{

Assert(count);

::InterlockedIncrement(count);

return \*this;

}

Counter& Counter::operator--()

{

Assert(count);

::InterlockedDecrement(count);

return \*this;

}

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef PERF\_COUNTERS

// Forward declaration from perflib.h

struct \_PERF\_COUNTERSET\_INSTANCE;

typedef struct \_PERF\_COUNTERSET\_INSTANCE \*PPERF\_COUNTERSET\_INSTANCE;

enum PageAllocatorType;

#define MAX\_OBJECT\_NAME\_PREFIX 1024

namespace PerfCounter

{

class Provider;

class InstanceBase

{

protected:

InstanceBase(Provider& provider, GUID const& guid);

~InstanceBase();

bool IsProviderInitialized() const;

bool Initialize(wchar\_t const \* wszInstanceName, DWORD id);

DWORD \* InitializeSharedMemory(DWORD numCounter, HANDLE& handle);

DWORD \* OpenSharedMemory(\_\_in\_ecount(MAX\_OBJECT\_NAME\_PREFIX) wchar\_t const wszObjectNamePrefix[MAX\_OBJECT\_NAME\_PREFIX], DWORD pid, DWORD numCounter, HANDLE& handle);

void UninitializeSharedMemory(DWORD \* data, HANDLE handle);

bool IsEnabled() const;

private:

Provider& GetProvider() { return provider; }

PPERF\_COUNTERSET\_INSTANCE GetData() { return instanceData; }

Provider& provider;

GUID const& guid;

PPERF\_COUNTERSET\_INSTANCE instanceData;

friend class Counter;

};

class Counter

{

public:

Counter() : count(NULL) {};

void Initialize(InstanceBase& instance, DWORD id, DWORD \* count);

void Uninitialize(InstanceBase& instance, DWORD id);

Counter& operator+=(size\_t value);

Counter& operator-=(size\_t value);

Counter& operator++();

Counter& operator--();

DWORD GetValue() { return \*count; }

private:

/\* TODO: 64-bit \*/

DWORD \* count;

};

class PageAllocatorCounterSetDefinition

{

public:

static DWORD const MaxCounter = 24;

static GUID const& GetGuid();

static Provider& GetProvider();

static uint GetReservedCounterId(PageAllocatorType type);

static uint GetCommittedCounterId(PageAllocatorType type);

static uint GetUsedCounterId(PageAllocatorType type);

};

class BasicCounterSetDefinition

{

public:

static DWORD const MaxCounter = 4;

static GUID const& GetGuid();

static Provider& GetProvider();

};

class CodeCounterSetDefinition

{

public:

static DWORD const MaxCounter = 17;

static GUID const& GetGuid();

static Provider& GetProvider();

};

#ifdef HEAP\_PERF\_COUNTERS

class HeapCounterSetDefinition

{

public:

static DWORD const MaxCounter = 2;

static GUID const& GetGuid();

static Provider& GetProvider();

};

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

class RecyclerCounterSetDefinition

{

public:

static DWORD const MaxCounter = 14;

static GUID const& GetGuid();

static Provider& GetProvider();

};

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

#define RECYCLER\_TRACKER\_PERF\_COUNTER\_TYPE(MACRO) \

MACRO(JavascriptNumber); \

MACRO(ConcatString); \

MACRO(LiteralString); \

MACRO(SubString); \

MACRO(PropertyString); \

MACRO(PropertyRecord); \

MACRO(DynamicObject); \

MACRO(CustomExternalObject); \

MACRO(DynamicType); \

MACRO(JavascriptFunction); \

MACRO(JavascriptArray); \

MACRO(SingleCharString); \

MACRO(FrameDisplay); \

MACRO(CompoundString); \

MACRO(RecyclerWeakReferenceBase); \

MACRO(ProjectionObjectInstance); \

#define RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER\_TYPE(MACRO) \

MACRO(Var); \

MACRO(wchar\_t); \

#define RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER\_TYPE(MACRO) \

MACRO(PropertyRecord); \

MACRO(DynamicType); \

MACRO(PropertyString); \

MACRO(DynamicObject); \

MACRO(Type); \

#define DECLARE\_RECYCLER\_TRACKER\_PERF\_COUNTER\_INDEX(type) \

static uint const type##CounterIndex; \

static uint const type##SizeCounterIndex;

#define DECLARE\_RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER\_INDEX(type) \

static uint const type##ArrayCounterIndex; \

static uint const type##ArraySizeCounterIndex;

#define DECLARE\_RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER\_INDEX(type) \

static uint const type##WeakRefCounterIndex;

class RecyclerTrackerCounterSetDefinition

{

public:

static DWORD const MaxCounter = 46;

static GUID const& GetGuid();

static Provider& GetProvider();

RECYCLER\_TRACKER\_PERF\_COUNTER\_TYPE(DECLARE\_RECYCLER\_TRACKER\_PERF\_COUNTER\_INDEX);

RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER\_TYPE(DECLARE\_RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER\_INDEX);

RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER\_TYPE(DECLARE\_RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER\_INDEX);

};

#undef DECLARE\_RECYCLER\_TRACKER\_PERF\_COUNTER\_INDEX

#undef DECLARE\_RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER\_INDEX

#undef DECLARE\_RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER\_INDEX

#endif

};

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#ifdef PERF\_COUNTERS

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAD")

#include "Microsoft-Scripting-Jscript9.InternalCounters.h"

namespace PerfCounter

{

class Provider

{

public:

static Provider InternalCounter;

#ifdef ENABLE\_COUNTER\_NOTIFICATION\_CALLBACK

void SetNotificationCallBack(PERFLIBREQUEST pfn) { pfnNotificationCallBack = pfn; }

#endif

private:

Provider(HANDLE& handle);

~Provider();

bool IsInitialized() const { return isInitialized; }

HANDLE GetHandler() { return handle; }

#ifdef ENABLE\_COUNTER\_NOTIFICATION\_CALLBACK

PERFLIBREQUEST pfnNotificationCallBack;

static ULONG WINAPI NotificationCallBack(ULONG RequestCode, PVOID Buffer, ULONG BufferSize);

#endif

HANDLE& handle;

bool isInitialized;

friend class InstanceBase;

friend class Counter;

};

Provider Provider::InternalCounter(JS9InternalCounterProvider);

#ifdef ENABLE\_COUNTER\_NOTIFICATION\_CALLBACK

ULONG WINAPI

Provider::NotificationCallBack(ULONG RequestCode, PVOID Buffer, ULONG BufferSize)

{

if (Provider::InternalCounter.pfnNotificationCallBack != NULL)

{

return Provider::InternalCounter.pfnNotificationCallBack(RequestCode, Buffer, BufferSize);

}

return ERROR\_SUCCESS;

}

#endif

Provider::Provider(HANDLE& handle) :

handle(handle), isInitialized(false)

{

PERFLIBREQUEST callback = NULL;

#ifdef ENABLE\_COUNTER\_NOTIFICATION\_CALLBACK

callback = &NotificationCallBack;

#endif

if (ERROR\_SUCCESS == CounterInitialize(callback, NULL, NULL, NULL))

{

isInitialized = true;

}

}

Provider::~Provider()

{

if (IsInitialized())

{

CounterCleanup();

}

}

InstanceBase::InstanceBase(Provider& provider, GUID const& guid) : provider(provider), guid(guid), instanceData(NULL)

{

}

InstanceBase::~InstanceBase()

{

if (IsEnabled())

{

::PerfDeleteInstance(provider.GetHandler(), instanceData);

}

}

bool

InstanceBase::IsProviderInitialized() const

{

return provider.IsInitialized();

}

bool

InstanceBase::IsEnabled() const

{

return instanceData != NULL;

}

static const size\_t GUID\_LEN = 37; // includes null

static const wchar\_t s\_wszObjectNamePrefix[] = L"jscript9\_perf\_counter\_";

static const size\_t OBJECT\_NAME\_LEN = GUID\_LEN + \_countof(s\_wszObjectNamePrefix) + 11;

static

void GetSharedMemoryObjectName(\_\_inout\_ecount(OBJECT\_NAME\_LEN) wchar\_t wszObjectName[OBJECT\_NAME\_LEN], DWORD pid, GUID const& guid)

{

swprintf\_s(wszObjectName, OBJECT\_NAME\_LEN, L"%s%d\_%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x",

s\_wszObjectNamePrefix, pid,

guid.Data1,

guid.Data2,

guid.Data3,

guid.Data4[0], guid.Data4[1],

guid.Data4[2], guid.Data4[3], guid.Data4[4], guid.Data4[5], guid.Data4[6], guid.Data4[7]);

}

bool

InstanceBase::Initialize(wchar\_t const \* wszInstanceName, DWORD processId)

{

if (provider.IsInitialized())

{

instanceData = PerfCreateInstance(provider.GetHandler(), &guid,

wszInstanceName, processId);

return instanceData != NULL;

}

return false;

}

DWORD \*

InstanceBase::InitializeSharedMemory(DWORD numCounter, HANDLE& handle)

{

Assert(!IsEnabled());

DWORD size = numCounter \* sizeof(DWORD);

wchar\_t wszObjectName[OBJECT\_NAME\_LEN];

GetSharedMemoryObjectName(wszObjectName, GetCurrentProcessId(), guid);

handle = ::CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, size, wszObjectName);

if (handle == NULL)

{

return NULL;

}

DWORD \* data = (DWORD \*)MapViewOfFile(handle, FILE\_MAP\_WRITE, 0, 0, size);

if (data == NULL)

{

CloseHandle(handle);

handle = NULL;

}

return data;

}

DWORD \*

InstanceBase::OpenSharedMemory(\_\_in\_ecount(MAX\_OBJECT\_NAME\_PREFIX) wchar\_t const wszObjectNamePrefix[MAX\_OBJECT\_NAME\_PREFIX],

DWORD pid, DWORD numCounter, HANDLE& handle)

{

DWORD size = numCounter \* sizeof(DWORD);

wchar\_t wszObjectName[OBJECT\_NAME\_LEN];

GetSharedMemoryObjectName(wszObjectName, pid, guid);

wchar\_t wszObjectNameFull[MAX\_OBJECT\_NAME\_PREFIX + OBJECT\_NAME\_LEN];

swprintf\_s(wszObjectNameFull, L"%s\\%s", wszObjectNamePrefix, wszObjectName);

handle = ::OpenFileMapping(FILE\_MAP\_READ, FALSE, wszObjectNameFull);

if (handle == NULL)

{

return NULL;

}

DWORD \* data = (DWORD \*)MapViewOfFile(handle, FILE\_MAP\_READ, 0, 0, size);

if (data == NULL)

{

CloseHandle(handle);

handle = NULL;

}

return data;

}

void

InstanceBase::UninitializeSharedMemory(DWORD \* data, HANDLE handle)

{

UnmapViewOfFile(data);

CloseHandle(handle);

}

void

Counter::Initialize(InstanceBase& instance, DWORD id, DWORD \* count)

{

this->count = count;

if (instance.IsEnabled())

{

::PerfSetCounterRefValue(instance.GetProvider().GetHandler(), instance.GetData(), id, count);

}

}

void

Counter::Uninitialize(InstanceBase& instance, DWORD id)

{

if (instance.IsEnabled())

{

::PerfSetCounterRefValue(instance.GetProvider().GetHandler(), instance.GetData(), id, NULL);

}

}

#define DEFINE\_PAGE\_ALLOCATOR\_COUNTER\_ID(type) JS9InternalCounter\_PageAllocCounterSet\_##type##ReservedSize,

static uint ReservedCounterId[PageAllocatorType\_Max + 1] =

{

PAGE\_ALLOCATOR\_TYPE(DEFINE\_PAGE\_ALLOCATOR\_COUNTER\_ID)

JS9InternalCounter\_PageAllocCounterSet\_TotalReservedSize

};

#undef DEFINE\_PAGE\_ALLOCATOR\_COUNTER\_ID

#define DEFINE\_PAGE\_ALLOCATOR\_COUNTER\_ID(type) JS9InternalCounter\_PageAllocCounterSet\_##type##CommittedSize,

static uint CommittedCounterId[PageAllocatorType\_Max + 1] =

{

PAGE\_ALLOCATOR\_TYPE(DEFINE\_PAGE\_ALLOCATOR\_COUNTER\_ID)

JS9InternalCounter\_PageAllocCounterSet\_TotalCommittedSize

};

#undef DEFINE\_PAGE\_ALLOCATOR\_COUNTER\_ID

#define DEFINE\_PAGE\_ALLOCATOR\_COUNTER\_ID(type) JS9InternalCounter\_PageAllocCounterSet\_##type##UsedSize,

static uint UsedCounterId[PageAllocatorType\_Max + 1] =

{

PAGE\_ALLOCATOR\_TYPE(DEFINE\_PAGE\_ALLOCATOR\_COUNTER\_ID)

JS9InternalCounter\_PageAllocCounterSet\_TotalUsedSize

};

#undef DEFINE\_PAGE\_ALLOCATOR\_COUNTER\_ID

uint

PageAllocatorCounterSetDefinition::GetReservedCounterId(PageAllocatorType type)

{

return ReservedCounterId[type];

}

uint

PageAllocatorCounterSetDefinition::GetCommittedCounterId(PageAllocatorType type)

{

return CommittedCounterId[type];

}

uint

PageAllocatorCounterSetDefinition::GetUsedCounterId(PageAllocatorType type)

{

return UsedCounterId[type];

}

GUID const& PageAllocatorCounterSetDefinition::GetGuid() { return JS9InternalCounter\_PageAllocCounterSetGuid; }

Provider& PageAllocatorCounterSetDefinition::GetProvider() { return Provider::InternalCounter; }

GUID const& BasicCounterSetDefinition::GetGuid() { return JS9InternalCounter\_BasicCounterSetGuid; }

Provider& BasicCounterSetDefinition::GetProvider() { return Provider::InternalCounter; }

GUID const& CodeCounterSetDefinition::GetGuid() { return JS9InternalCounter\_CodeCounterSetGuid; }

Provider& CodeCounterSetDefinition::GetProvider() { return Provider::InternalCounter; }

#ifdef HEAP\_PERF\_COUNTERS

GUID const& HeapCounterSetDefinition::GetGuid() { return JS9InternalCounter\_HeapCounterSetGuid; }

Provider& HeapCounterSetDefinition::GetProvider() { return Provider::InternalCounter; }

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

GUID const& RecyclerCounterSetDefinition::GetGuid() { return JS9InternalCounter\_RecyclerCounterSetGuid; }

Provider& RecyclerCounterSetDefinition::GetProvider() { return Provider::InternalCounter; }

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

GUID const& RecyclerTrackerCounterSetDefinition::GetGuid() { return JS9InternalCounter\_RecyclerTrackerCounterSetGuid; }

Provider& RecyclerTrackerCounterSetDefinition::GetProvider() { return Provider::InternalCounter; }

#define DEFINE\_RECYCLER\_TRACKER\_PERF\_COUNTER\_INDEX(type) \

uint const RecyclerTrackerCounterSetDefinition::##type##CounterIndex = JS9InternalCounter\_RecyclerTrackerCounterSet\_##type##Count; \

uint const RecyclerTrackerCounterSetDefinition::##type##SizeCounterIndex = JS9InternalCounter\_RecyclerTrackerCounterSet\_##type##Size;

#define DEFINE\_RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER\_INDEX(type) \

uint const RecyclerTrackerCounterSetDefinition::##type##ArrayCounterIndex = JS9InternalCounter\_RecyclerTrackerCounterSet\_##type##ArrayCount; \

uint const RecyclerTrackerCounterSetDefinition::##type##ArraySizeCounterIndex = JS9InternalCounter\_RecyclerTrackerCounterSet\_##type##ArraySize;

#define DEFINE\_RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER\_INDEX(type) \

uint const RecyclerTrackerCounterSetDefinition::##type##WeakRefCounterIndex = JS9InternalCounter\_RecyclerTrackerCounterSet\_##type##WeakRefCount;

RECYCLER\_TRACKER\_PERF\_COUNTER\_TYPE(DEFINE\_RECYCLER\_TRACKER\_PERF\_COUNTER\_INDEX);

RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER\_TYPE(DEFINE\_RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER\_INDEX);

RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER\_TYPE(DEFINE\_RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER\_INDEX);

#endif

};

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#ifdef PERF\_COUNTERS

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAE")

namespace PerfCounter

{

//===================================================================================

// PageAllocatorCounterSet

//===================================================================================

Counter&

PageAllocatorCounterSet::GetTotalReservedSizeCounter()

{

return instance.GetCounter(PageAllocatorCounterSetDefinition::GetReservedCounterId(PageAllocatorType\_Max));

}

Counter&

PageAllocatorCounterSet::GetTotalCommittedSizeCounter()

{

return instance.GetCounter(PageAllocatorCounterSetDefinition::GetCommittedCounterId(PageAllocatorType\_Max));

}

Counter&

PageAllocatorCounterSet::GetTotalUsedSizeCounter()

{

return instance.GetCounter(PageAllocatorCounterSetDefinition::GetUsedCounterId(PageAllocatorType\_Max));

}

DefaultCounterSetInstance<PageAllocatorCounterSetDefinition> PageAllocatorCounterSet::instance;

DefaultCounterSetInstance<BasicCounterSetDefinition> BasicCounterSet::instance;

DefaultCounterSetInstance<CodeCounterSetDefinition> CodeCounterSet::instance;

#ifdef HEAP\_PERF\_COUNTERS

DefaultCounterSetInstance<HeapCounterSetDefinition> HeapCounterSet::instance;

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

DefaultCounterSetInstance<RecyclerCounterSetDefinition> RecyclerCounterSet::instance;

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

DefaultCounterSetInstance<RecyclerTrackerCounterSetDefinition> RecyclerTrackerCounterSet::instance;

type\_info const \* RecyclerTrackerCounterSet::CountIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - RecyclerTrackerCounterSet::NumUnknownCounters];

type\_info const \* RecyclerTrackerCounterSet::SizeIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - RecyclerTrackerCounterSet::NumUnknownCounters];

type\_info const \* RecyclerTrackerCounterSet::ArrayCountIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - RecyclerTrackerCounterSet::NumUnknownCounters];

type\_info const \* RecyclerTrackerCounterSet::ArraySizeIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - RecyclerTrackerCounterSet::NumUnknownCounters];

type\_info const \* RecyclerTrackerCounterSet::WeakRefIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - RecyclerTrackerCounterSet::NumUnknownCounters];

RecyclerTrackerCounterSet::Map::Map(type\_info const \* type, bool isArray,uint counterIndex, uint sizeCounterIndex)

{

Assert(counterIndex >= NumUnknownCounters && counterIndex < RecyclerTrackerCounterSetDefinition::MaxCounter);

Assert(sizeCounterIndex >= NumUnknownCounters && sizeCounterIndex < RecyclerTrackerCounterSetDefinition::MaxCounter);

\_\_analysis\_assume(counterIndex >= NumUnknownCounters && counterIndex < RecyclerTrackerCounterSetDefinition::MaxCounter);

\_\_analysis\_assume(sizeCounterIndex >= NumUnknownCounters && sizeCounterIndex < RecyclerTrackerCounterSetDefinition::MaxCounter);

if (isArray)

{

Assert(ArrayCountIndexTypeInfoMap[counterIndex - NumUnknownCounters] == nullptr);

Assert(ArraySizeIndexTypeInfoMap[sizeCounterIndex - NumUnknownCounters] == nullptr);

ArrayCountIndexTypeInfoMap[counterIndex - NumUnknownCounters] = type;

ArraySizeIndexTypeInfoMap[sizeCounterIndex - NumUnknownCounters] = type;

}

else

{

Assert(CountIndexTypeInfoMap[counterIndex - NumUnknownCounters] == nullptr);

Assert(SizeIndexTypeInfoMap[sizeCounterIndex - NumUnknownCounters] == nullptr);

CountIndexTypeInfoMap[counterIndex - NumUnknownCounters] = type;

SizeIndexTypeInfoMap[sizeCounterIndex - NumUnknownCounters] = type;

}

}

RecyclerTrackerCounterSet::Map::Map(type\_info const \* type, uint weakRefCounterIndex)

{

\_\_analysis\_assume(weakRefCounterIndex >= NumUnknownCounters && weakRefCounterIndex < RecyclerTrackerCounterSetDefinition::MaxCounter);

Assert(weakRefCounterIndex >= NumUnknownCounters && weakRefCounterIndex < RecyclerTrackerCounterSetDefinition::MaxCounter);

Assert(WeakRefIndexTypeInfoMap[weakRefCounterIndex - NumUnknownCounters] == nullptr);

WeakRefIndexTypeInfoMap[weakRefCounterIndex - NumUnknownCounters] = type;

}

Counter&

RecyclerTrackerCounterSet::GetPerfCounter(type\_info const \* typeinfo, bool isArray)

{

Counter& unknownCounter = isArray? GetUnknownArrayCounter() : GetUnknownCounter();

type\_info const \*\*counters = isArray? ArrayCountIndexTypeInfoMap : CountIndexTypeInfoMap;

for (uint i = 0; i < RecyclerTrackerCounterSetDefinition::MaxCounter - NumUnknownCounters; i++)

{

if (typeinfo == counters[i])

{

return instance.GetCounter(i + NumUnknownCounters);

}

}

return unknownCounter;

}

Counter&

RecyclerTrackerCounterSet::GetPerfSizeCounter(type\_info const \* typeinfo, bool isArray)

{

Counter& unknownCounter = isArray? GetUnknownArraySizeCounter() : GetUnknownSizeCounter();

type\_info const \*\*counters = isArray? ArraySizeIndexTypeInfoMap : SizeIndexTypeInfoMap;

for (uint i = 0; i < RecyclerTrackerCounterSetDefinition::MaxCounter - NumUnknownCounters; i++)

{

if (typeinfo == counters[i])

{

return instance.GetCounter(i + NumUnknownCounters);

}

}

return unknownCounter;

}

Counter&

RecyclerTrackerCounterSet::GetWeakRefPerfCounter(type\_info const \* typeinfo)

{

Counter& unknownCounter = GetUnknownWeakRefCounter();

for (uint i = 0; i < RecyclerTrackerCounterSetDefinition::MaxCounter - NumUnknownCounters; i++)

{

if (typeinfo == WeakRefIndexTypeInfoMap[i])

{

return instance.GetCounter(i + NumUnknownCounters);

}

}

return unknownCounter;

}

#endif

};

#endif

DEFINE\_RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER(wchar\_t);

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef PERF\_COUNTERS

namespace PerfCounter

{

template <typename TCounter>

class DefaultCounterSetInstance : public InstanceBase

{

public:

DefaultCounterSetInstance() : InstanceBase(TCounter::GetProvider(), TCounter::GetGuid())

{

if (Initialize())

{

data = defaultData;

}

else

{

// if for any reason perf counter failed to initialize, try to create

// shared memory instead. This will happen for sure if running under

// Win8 AppContainer because they don't support v2 perf counters.

// See comments in WWAHostJSCounterProvider for details.

data = \_\_super::InitializeSharedMemory(TCounter::MaxCounter, handle);

if (data == nullptr)

{

data = defaultData;

}

}

for (uint i = 0; i < TCounter::MaxCounter; i++)

{

data[i] = 0;

counters[i].Initialize(\*this, i, &data[i]);

}

}

~DefaultCounterSetInstance()

{

for (uint i = 0; i < TCounter::MaxCounter; i++)

{

counters[i].Uninitialize(\*this, i);

}

if (data != defaultData)

{

\_\_super::UninitializeSharedMemory(data, handle);

}

}

Counter& GetCounter(uint id) { Assert(id < TCounter::MaxCounter); return counters[id]; }

bool Initialize()

{

if (IsProviderInitialized())

{

wchar\_t wszModuleName[\_MAX\_PATH];

if (!GetModuleFileName(NULL, wszModuleName, \_MAX\_PATH))

{

return false;

}

wchar\_t wszFilename[\_MAX\_FNAME];

\_wsplitpath\_s(wszModuleName, NULL, 0, NULL, 0, wszFilename, \_MAX\_FNAME, NULL, 0);

return \_\_super::Initialize(wszFilename, GetCurrentProcessId());

}

return false;

}

private:

DWORD defaultData[TCounter::MaxCounter];

DWORD \* data;

HANDLE handle;

Counter counters[TCounter::MaxCounter];

};

class PageAllocatorCounterSet

{

public:

static Counter& GetReservedSizeCounter(PageAllocatorType type)

{ return instance.GetCounter(PageAllocatorCounterSetDefinition::GetReservedCounterId(type)); }

static Counter& GetTotalReservedSizeCounter();

static Counter& GetCommittedSizeCounter(PageAllocatorType type)

{ return instance.GetCounter(PageAllocatorCounterSetDefinition::GetCommittedCounterId(type)); }

static Counter& GetTotalCommittedSizeCounter();

static Counter& GetUsedSizeCounter(PageAllocatorType type)

{ return instance.GetCounter(PageAllocatorCounterSetDefinition::GetUsedCounterId(type)); }

static Counter& GetTotalUsedSizeCounter();

private:

static DefaultCounterSetInstance<PageAllocatorCounterSetDefinition> instance;

};

class BasicCounterSet

{

public:

static Counter& GetThreadContextCounter() { return instance.GetCounter(0); }

static Counter& GetScriptContextCounter() { return instance.GetCounter(1); }

static Counter& GetScriptContextActiveCounter() { return instance.GetCounter(2); }

static Counter& GetScriptCodeBufferCountCounter() { return instance.GetCounter(3); }

private:

static DefaultCounterSetInstance<BasicCounterSetDefinition> instance;

};

class CodeCounterSet

{

public:

static Counter& GetTotalByteCodeSizeCounter() { return instance.GetCounter(0); }

static Counter& GetTotalNativeCodeSizeCounter() { return instance.GetCounter(1); }

static Counter& GetTotalNativeCodeDataSizeCounter() { return instance.GetCounter(2); }

static Counter& GetStaticByteCodeSizeCounter() { return instance.GetCounter(3); }

static Counter& GetStaticNativeCodeSizeCounter() { return instance.GetCounter(4); }

static Counter& GetStaticNativeCodeDataSizeCounter() { return instance.GetCounter(5); }

static Counter& GetDynamicByteCodeSizeCounter() { return instance.GetCounter(6); }

static Counter& GetDynamicNativeCodeSizeCounter() { return instance.GetCounter(7); }

static Counter& GetDynamicNativeCodeDataSizeCounter() { return instance.GetCounter(8); }

static Counter& GetTotalFunctionCounter() { return instance.GetCounter(9); }

static Counter& GetStaticFunctionCounter() { return instance.GetCounter(10); }

static Counter& GetDynamicFunctionCounter() { return instance.GetCounter(11); }

static Counter& GetLoopNativeCodeSizeCounter() { return instance.GetCounter(12); }

static Counter& GetFunctionNativeCodeSizeCounter() { return instance.GetCounter(13); }

static Counter& GetDeferDeserializeFunctionProxyCounter() { return instance.GetCounter(14); }

static Counter& GetDeserializedFunctionBodyCounter() { return instance.GetCounter(15); }

static Counter& GetDeferedFunctionCounter() { return instance.GetCounter(16); }

private:

static DefaultCounterSetInstance<CodeCounterSetDefinition> instance;

};

#ifdef HEAP\_PERF\_COUNTERS

class HeapCounterSet

{

public:

static Counter& GetLiveObjectCounter() { return instance.GetCounter(0); }

static Counter& GetLiveObjectSizeCounter() { return instance.GetCounter(1); }

private:

static DefaultCounterSetInstance<HeapCounterSetDefinition> instance;

};

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

class RecyclerCounterSet

{

public:

static Counter& GetLiveObjectSizeCounter() { return instance.GetCounter(0); }

static Counter& GetLiveObjectCounter() { return instance.GetCounter(1); }

static Counter& GetFreeObjectSizeCounter() { return instance.GetCounter(2); }

static Counter& GetPinnedObjectCounter() { return instance.GetCounter(3); }

static Counter& GetBindReferenceCounter() { return instance.GetCounter(4); }

static Counter& GetPropertyRecordBindReferenceCounter() { return instance.GetCounter(5); }

static Counter& GetLargeHeapBlockLiveObjectSizeCounter() { return instance.GetCounter(6); }

static Counter& GetLargeHeapBlockLiveObjectCounter() { return instance.GetCounter(7); }

static Counter& GetLargeHeapBlockFreeObjectSizeCounter() { return instance.GetCounter(8); }

static Counter& GetSmallHeapBlockLiveObjectSizeCounter() { return instance.GetCounter(9); }

static Counter& GetSmallHeapBlockLiveObjectCounter() { return instance.GetCounter(10); }

static Counter& GetSmallHeapBlockFreeObjectSizeCounter() { return instance.GetCounter(11); }

static Counter& GetLargeHeapBlockCountCounter() { return instance.GetCounter(12); }

static Counter& GetLargeHeapBlockPageSizeCounter() { return instance.GetCounter(13); }

private:

static DefaultCounterSetInstance<RecyclerCounterSetDefinition> instance;

};

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

class RecyclerTrackerCounterSet

{

public:

static Counter& GetPerfCounter(type\_info const \* typeinfo, bool isArray);

static Counter& GetPerfSizeCounter(type\_info const \* typeinfo, bool isArray);

static Counter& GetWeakRefPerfCounter(type\_info const \* typeinfo);

class Map

{

public:

Map(type\_info const \* type, bool isArray, uint counterIndex, uint sizeCounterIndex);

Map(type\_info const \* type, uint weakRefCounterIndex);

};

private:

static Counter& GetUnknownCounter() { return instance.GetCounter(0); }

static Counter& GetUnknownSizeCounter() { return instance.GetCounter(1); }

static Counter& GetUnknownArrayCounter() { return instance.GetCounter(2); }

static Counter& GetUnknownArraySizeCounter() { return instance.GetCounter(3); }

static Counter& GetUnknownWeakRefCounter() { return instance.GetCounter(4); }

static DefaultCounterSetInstance<RecyclerTrackerCounterSetDefinition> instance;

static uint const NumUnknownCounters = 5;

static type\_info const \* CountIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - NumUnknownCounters];

static type\_info const \* SizeIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - NumUnknownCounters];

static type\_info const \* ArrayCountIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - NumUnknownCounters];

static type\_info const \* ArraySizeIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - NumUnknownCounters];

static type\_info const \* WeakRefIndexTypeInfoMap[RecyclerTrackerCounterSetDefinition::MaxCounter - NumUnknownCounters];

};

#define DEFINE\_RECYCLER\_TRACKER\_PERF\_COUNTER(type) \

static PerfCounter::RecyclerTrackerCounterSet::Map RecyclerTrackerCounter##id(&typeid(type), false, \

PerfCounter::RecyclerTrackerCounterSetDefinition::##type##CounterIndex, \

PerfCounter::RecyclerTrackerCounterSetDefinition::##type##SizeCounterIndex)

#define DEFINE\_RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER(type) \

static PerfCounter::RecyclerTrackerCounterSet::Map RecyclerTrackerArrayCounter##id(&typeid(type), true, \

PerfCounter::RecyclerTrackerCounterSetDefinition::##type##ArrayCounterIndex, \

PerfCounter::RecyclerTrackerCounterSetDefinition::##type##ArraySizeCounterIndex)

#define DEFINE\_RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER(type) \

static PerfCounter::RecyclerTrackerCounterSet::Map RecyclerTrackerWeakRefCounter##id(&typeid(type), \

PerfCounter::RecyclerTrackerCounterSetDefinition::##type##WeakRefCounterIndex);

#else

#define DEFINE\_RECYCLER\_TRACKER\_PERF\_COUNTER(type)

#define DEFINE\_RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER(type)

#define DEFINE\_RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER(type)

#endif

};

#define PERF\_COUNTER\_INC(CounterSetName, CounterName) ++PerfCounter::CounterSetName##CounterSet::Get##CounterName##Counter()

#define PERF\_COUNTER\_DEC(CounterSetName, CounterName) --PerfCounter::CounterSetName##CounterSet::Get##CounterName##Counter()

#define PERF\_COUNTER\_ADD(CounterSetName, CounterName, value) PerfCounter::CounterSetName##CounterSet::Get##CounterName##Counter() += value

#define PERF\_COUNTER\_SUB(CounterSetName, CounterName, value) PerfCounter::CounterSetName##CounterSet::Get##CounterName##Counter() -= value

#else

#define PERF\_COUNTER\_INC(CounterSetName, CounterName)

#define PERF\_COUNTER\_DEC(CounterSetName, CounterName)

#define PERF\_COUNTER\_ADD(CounterSetName, CounterName, value)

#define PERF\_COUNTER\_SUB(CounterSetName, CounterName, value)

#define DEFINE\_RECYCLER\_TRACKER\_PERF\_COUNTER(type)

#define DEFINE\_RECYCLER\_TRACKER\_ARRAY\_PERF\_COUNTER(type)

#define DEFINE\_RECYCLER\_TRACKER\_WEAKREF\_PERF\_COUNTER(type)

#endif

#ifdef HEAP\_PERF\_COUNTERS

#define HEAP\_PERF\_COUNTER\_INC(CounterName) PERF\_COUNTER\_INC(Heap, CounterName)

#define HEAP\_PERF\_COUNTER\_DEC(CounterName) PERF\_COUNTER\_DEC(Heap, CounterName)

#define HEAP\_PERF\_COUNTER\_ADD(CounterName, value) PERF\_COUNTER\_ADD(Heap, CounterName, value)

#define HEAP\_PERF\_COUNTER\_SUB(CounterName, value) PERF\_COUNTER\_SUB(Heap, CounterName, value)

#else

#define HEAP\_PERF\_COUNTER\_INC(CounterName)

#define HEAP\_PERF\_COUNTER\_DEC(CounterName)

#define HEAP\_PERF\_COUNTER\_ADD(CounterName, value)

#define HEAP\_PERF\_COUNTER\_SUB(CounterName, value)

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

#define RECYCLER\_PERF\_COUNTER\_INC(CounterName) PERF\_COUNTER\_INC(Recycler, CounterName)

#define RECYCLER\_PERF\_COUNTER\_DEC(CounterName) PERF\_COUNTER\_DEC(Recycler, CounterName)

#define RECYCLER\_PERF\_COUNTER\_ADD(CounterName, value) PERF\_COUNTER\_ADD(Recycler, CounterName, value)

#define RECYCLER\_PERF\_COUNTER\_SUB(CounterName, value) PERF\_COUNTER\_SUB(Recycler, CounterName, value)

#else

#define RECYCLER\_PERF\_COUNTER\_INC(CounterName)

#define RECYCLER\_PERF\_COUNTER\_DEC(CounterName)

#define RECYCLER\_PERF\_COUNTER\_ADD(CounterName, value)

#define RECYCLER\_PERF\_COUNTER\_SUB(CounterName, value)

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#ifdef PROFILE\_EXEC

#include "core\ProfileInstrument.h"

#define HIRES\_PROFILER

namespace Js

{

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class Profiler::UnitData

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// UnitData::UnitData

///

/// Constructor

///

///----------------------------------------------------------------------------

UnitData::UnitData()

{

this->incl = 0;

this->excl = 0;

this->max = 0;

this->count = 0;

}

///----------------------------------------------------------------------------

///

/// UnitData::Add

///

///----------------------------------------------------------------------------

void

UnitData::Add(TimeStamp incl, TimeStamp excl)

{

this->incl += incl;

this->excl += excl;

this->count++;

if (incl > this->max)

{

this->max = incl;

}

}

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class Profiler

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// Profiler::Profiler

///

/// Constructor

///

///----------------------------------------------------------------------------

Profiler::Profiler(ArenaAllocator \* allocator) :

alloc(allocator),

rootNode(NULL)

{

this->curNode = &this->rootNode;

for(int i = 0; i < PhaseCount; i++)

{

this->inclSumAtLevel[i] = 0;

}

}

///----------------------------------------------------------------------------

///

/// Profiler::Begin

///

///----------------------------------------------------------------------------

void

Profiler::Begin(Phase tag)

{

Push(TimeEntry(tag, GetTime()));

}

///----------------------------------------------------------------------------

///

/// Profiler::End

///

///----------------------------------------------------------------------------

void

Profiler::End(Phase tag)

{

Pop(TimeEntry(tag, GetTime()));

}

void

Profiler::Suspend(Phase tag, SuspendRecord \* suspendRecord)

{

suspendRecord->count = 0;

Phase topTag;

do

{

topTag = timeStack.Peek()->tag;

Pop(TimeEntry(topTag, GetTime()));

suspendRecord->phase[suspendRecord->count++] = topTag;

} while(topTag != tag);

}

void

Profiler::Resume(SuspendRecord \* suspendRecord)

{

while (suspendRecord->count)

{

suspendRecord->count--;

Begin(suspendRecord->phase[suspendRecord->count]);

}

}

///----------------------------------------------------------------------------

///

/// Profiler::EndAllUpTo

///

/// Ends all phases up to the specified phase. Useful for catching exceptions

/// after a phase was started, and ending all intermediate phases until the

/// first phase that was started.

///

///----------------------------------------------------------------------------

void

Profiler::EndAllUpTo(Phase tag)

{

Phase topTag;

do

{

topTag = timeStack.Peek()->tag;

Pop(TimeEntry(topTag, GetTime()));

} while(topTag != tag);

}

///----------------------------------------------------------------------------

///

/// Profiler::Push

///

/// 1. Push entry on stack.

/// 2. Update curNode

///

///----------------------------------------------------------------------------

void

Profiler::Push(TimeEntry entry)

{

AssertMsg(NULL != curNode, "Profiler Stack Corruption");

this->timeStack.Push(entry);

if(!curNode->ChildExistsAt(entry.tag))

{

TypeNode \* node = AnewNoThrow(this->alloc, TypeNode, curNode);

// We crash if we run out of memory here and we don't care

curNode->SetChildAt(entry.tag, node);

}

curNode = curNode->GetChildAt(entry.tag);

}

///----------------------------------------------------------------------------

///

/// Profiler::Pop

///

/// Core logic for the timer. Calculated the exclusive, inclusive times.

/// There is a list inclSumAtLevel which stores accumulates the inclusive sum

/// of all the tags that where 'pushed' after this tag.

///

/// Consider the following calls. fx indicates Push and fx', the corresponding Pop

///

/// f1

/// f2

/// f3

/// f3'

/// f2'

/// f4

/// f5

/// f5'

/// f4'

/// f1'

///

/// calculating the inclusive times are trivial. Let us calculate the exclusive

/// time for f1. That would be

/// excl(f1) = incl(f1) - [incl(f2) + incl(f4)]

///

/// Basically if a function is at level 'x' then we need to deduct from its

/// exclusive times, the inclusive times of all the functions at level 'x + 1'

/// We don't care about deeper levels. Hence 'inclSumAtLevel' array which accumulates

/// the sum of variables at different levels.

///

/// Reseting the next level is also required. In the above example, f3 and f5 are

/// at the same level. if we don't reset level 3 when popping f2, then we will

/// have wrong sums for f4. So once a tag has been popped, all sums at its higher

/// levels is set to zero. (Of course we just need to reset the next level and

/// all above levels will invariably remain zero)

///

///----------------------------------------------------------------------------

void

Profiler::Pop(TimeEntry curEntry)

{

int curLevel = this->timeStack.Count();

TimeEntry \*entry = this->timeStack.Pop();

AssertMsg(entry->tag == curEntry.tag, "Profiler Stack corruption, push pop entries do not correspond to the same tag");

TimeStamp inclusive = curEntry.time - entry->time;

TimeStamp exclusive = inclusive - this->inclSumAtLevel[curLevel +1];

Assert(inclusive >= 0);

Assert(exclusive >= 0);

this->inclSumAtLevel[curLevel + 1] = 0;

this->inclSumAtLevel[curLevel] += inclusive;

curNode->GetValue()->Add(inclusive, exclusive);

curNode = curNode->GetParent();

AssertMsg(curNode != NULL, "Profiler stack corruption");

}

void

Profiler::Merge(Profiler \* profiler)

{

MergeTree(&rootNode, &profiler->rootNode);

if (profiler->timeStack.Count() > 1)

{

FixedStack<TimeEntry, MaxStackDepth> reverseStack;

do

{

reverseStack.Push(\*profiler->timeStack.Pop());

}

while (profiler->timeStack.Count() > 1);

do

{

TimeEntry \* entry = reverseStack.Pop();

this->Push(\*entry);

profiler->timeStack.Push(\*entry);

}

while (reverseStack.Count() != 0);

}

}

void

Profiler::MergeTree(TypeNode \* toNode, TypeNode \* fromNode)

{

UnitData \* toData = toNode->GetValue();

const UnitData \* fromData = fromNode->GetValue();

toData->count += fromData->count;

toData->incl += fromData->incl;

toData->excl += fromData->excl;

if (fromData->max > toData->max)

{

toData->max = fromData->max;

}

for (int i = 0; i < PhaseCount; i++)

{

if (fromNode->ChildExistsAt(i))

{

TypeNode \* fromChild = fromNode->GetChildAt(i);

TypeNode \* toChild;

if (!toNode->ChildExistsAt(i))

{

toChild = Anew(this->alloc, TypeNode, toNode);

toNode->SetChildAt(i, toChild);

}

else

{

toChild = toNode->GetChildAt(i);

}

MergeTree(toChild, fromChild);

}

}

}

///----------------------------------------------------------------------------

///

/// Profiler::PrintTree

///

/// Private method that walks the tree and prints it recursively.

///

///----------------------------------------------------------------------------

void

Profiler::PrintTree(TypeNode \*node, TypeNode \*baseNode, int column, TimeStamp freq)

{

const UnitData \*base = baseNode->GetValue();

for(int i = 0; i < PhaseCount; i++)

{

if(node->ChildExistsAt(i))

{

UnitData \*data = node->GetChildAt(i)->GetValue();

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if( int(data->incl \* 100 / base->incl) >= Configuration::Global.flags.ProfileThreshold) // threshold

#endif

{

Output::SkipToColumn(column);

Output::Print(L"%-\*s %7.1f %5d %7.1f %5d %7.1f %7.1f %5d\n",

(Profiler::PhaseNameWidth-column), PhaseNames[i],

(double)data->incl / freq , // incl

int(data->incl \* 100 / base->incl ), // incl %

(double)data->excl / freq , // excl

int(data->excl \* 100 / base->incl ), // excl %

(double)data->max / freq , // max

(double)data->incl / ( freq \* data->count ), // mean

int(data->count) // count

);

}

PrintTree(node->GetChildAt(i), baseNode, column + Profiler::TabWidth, freq);

}

}

}

///----------------------------------------------------------------------------

///

/// Profiler::Print

///

/// Pretty printer

///

///----------------------------------------------------------------------------

void

Profiler::Print(Phase baseTag)

{

if (baseTag == InvalidPhase)

{

baseTag = AllPhase; // default to all phase

}

const TimeStamp freq = this->GetFrequency();

bool foundNode = false;

ForEachNode(baseTag, &rootNode, [&](TypeNode \*const baseNode, const Phase parentTag)

{

if(!foundNode)

{

foundNode = true;

Output::Print(L"%-\*s:%7s %5s %7s %5s %7s %7s %5s\n",

(Profiler::PhaseNameWidth-0),

L"Profiler Report",

L"Incl",

L"(%)",

L"Excl",

L"(%)",

L"Max",

L"Mean",

L"Count"

);

Output::Print(L"-------------------------------------------------------------------------------\n");

}

UnitData \*data = baseNode->GetValue();

if(0 == data->count)

{

Output::Print(L"The phase : %s was never started", PhaseNames[baseTag]);

return;

}

int indent = 0;

if(parentTag != InvalidPhase)

{

TypeNode \*const parentNode = baseNode->GetParent();

Assert(parentNode);

Output::Print(L"%-\*s\n", (Profiler::PhaseNameWidth-0), PhaseNames[parentTag]);

indent += Profiler::TabWidth;

}

if(indent)

{

Output::SkipToColumn(indent);

}

Output::Print(L"%-\*s %7.1f %5d %7.1f %5d %7.1f %7.1f %5d\n",

(Profiler::PhaseNameWidth-indent),

PhaseNames[baseTag],

(double)data->incl / freq , // incl

int(100), // incl %

(double)data->excl / freq , // excl

int(data->excl \* 100 / data->incl ), // excl %

(double)data->max / freq , // max

(double)data->incl / ( freq \* data->count ),// mean

int(data->count) // count

);

indent += Profiler::TabWidth;

PrintTree(baseNode, baseNode, indent, freq);

});

if(foundNode)

{

Output::Print(L"-------------------------------------------------------------------------------\n");

Output::Flush();

}

}

///----------------------------------------------------------------------------

///

/// Profiler::FindNode

///

/// Does a tree traversal(DFS) and finds the first occurrence of the 'tag'

///

///----------------------------------------------------------------------------

template<class FVisit>

void

Profiler::ForEachNode(Phase tag, TypeNode \*node, FVisit visit, Phase parentTag)

{

AssertMsg(node != NULL, "Invalid usage: node must always be non null");

for(int i = 0; i < PhaseCount; i++)

{

if(node->ChildExistsAt(i))

{

TypeNode \* child = node->GetChildAt(i);

if(i == tag)

{

visit(child, parentTag);

}

else

{

ForEachNode(tag, child, visit, static\_cast<Phase>(i));

}

}

}

}

///----------------------------------------------------------------------------

///

/// Profiler::GetTime

///

///----------------------------------------------------------------------------

TimeStamp

Profiler::GetTime()

{

#if !defined HIRES\_PROFILER && (defined(\_M\_IX86) || defined(\_M\_X64))

return \_\_rdtsc();

#else

LARGE\_INTEGER tmp;

if(QueryPerformanceCounter(&tmp))

{

return tmp.QuadPart;

}

else

{

AssertMsg(0, "Could not get time. Don't know what to do");

return 0;

}

#endif

}

///----------------------------------------------------------------------------

///

/// Profiler::GetFrequency

///

///----------------------------------------------------------------------------

TimeStamp

Profiler::GetFrequency()

{

#if !defined HIRES\_PROFILER && (defined(\_M\_IX86) || defined(\_M\_X64))

long long start, end;

int CPUInfo[4];

// Flush pipeline

\_\_cpuid(CPUInfo, 0);

// Measure 1 second / 5

start = GetTime();

Sleep(1000/5);

end = GetTime();

return ((end - start) \* 5) / FrequencyScale;

#else

LARGE\_INTEGER tmp;

if(QueryPerformanceFrequency(&tmp))

{

return tmp.QuadPart / FrequencyScale;

}

else

{

AssertMsg(0, "Could not get time. Don't know what to do");

return 0;

}

#endif

}

} //namespace Js

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef PROFILE\_EXEC

#include "DataStructures\FixedStack.h"

#include "DataStructures\Tree.h"

namespace Js

{

//

// Type of each timestamp.

//

typedef long long TimeStamp;

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// struct TimeEntry

///

/// Simple pair of (tag, timestamp).

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

struct TimeEntry

{

Phase tag;

TimeStamp time;

explicit TimeEntry(Phase profileTag = InvalidPhase, TimeStamp curTime = 0) :

tag(profileTag),

time(curTime)

{}

};

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// struct UnitData

///

/// Stores the elemental data for individual calls. Accumulates the inclusive,

/// exclusive sums, count, and maximum.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

struct UnitData

{

// Data

public:

uint count;

TimeStamp incl;

TimeStamp excl;

TimeStamp max;

// Constructor

public:

UnitData();

// Methods

public:

void Add(TimeStamp, TimeStamp);

};

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class Profiler

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

class Profiler

{

friend class ScriptContextProfiler;

// Local types

private:

typedef TreeNode<UnitData, PhaseCount> TypeNode;

// Data

private:

//

// Maximum depth of the stack. This should be some safe limit

//

static const int MaxStackDepth = 20;

//

// Controls the scale of the measuring clock. a value of 1000 indicates

// millisecond, 1000,000 implies micro-seconds.

//

static const int FrequencyScale = 1000;

//

// Used for printing the report

//

static const int TabWidth = 1;

static const int PhaseNameWidth = 27;

FixedStack<TimeEntry, MaxStackDepth>

timeStack;

TimeStamp inclSumAtLevel[PhaseCount];

TypeNode rootNode;

TypeNode \*curNode;

ArenaAllocator \*alloc;

// Constructor

public:

Profiler(ArenaAllocator \*alloc);

// Implementation

private:

static TimeStamp GetTime();

static TimeStamp GetFrequency();

template<class FVisit>

static void ForEachNode(Phase tag, TypeNode \*curNode, FVisit visit, Phase parentTag = InvalidPhase);

void Push(TimeEntry entry);

void Pop(TimeEntry entry);

void PrintTree(TypeNode \*curNode, TypeNode \*baseNode, int, TimeStamp freq);

void MergeTree(TypeNode \*toNode, TypeNode \*fromNode);

// Methods

public:

class SuspendRecord

{

private:

Phase phase[MaxStackDepth];

uint count;

friend class Profiler;

};

void Begin(Phase tag);

void End(Phase tag);

void EndAllUpTo(Phase tag);

void Print(Phase baseTag);

void Merge(Profiler \* profiler);

void Suspend(Phase tag, SuspendRecord \* suspendRecord);

void Resume(SuspendRecord \* suspendRecord);

};

} //namespace Js

#define ASYNC\_HOST\_OPERATION\_START(threadContext) {Js::Profiler::SuspendRecord \_\_suspendRecord; bool wasInAsync = threadContext->AsyncHostOperationStart(&\_\_suspendRecord)

#define ASYNC\_HOST\_OPERATION\_END(threadContext) threadContext->AsyncHostOperationEnd(wasInAsync, &\_\_suspendRecord); }

#elif DBG

#define ASYNC\_HOST\_OPERATION\_START(threadContext) { bool wasInAsync = threadContext->AsyncHostOperationStart(null)

#define ASYNC\_HOST\_OPERATION\_END(threadContext) threadContext->AsyncHostOperationEnd(wasInAsync, null)

#else

#define ASYNC\_HOST\_OPERATION\_START(threadContext)

#define ASYNC\_HOST\_OPERATION\_END(threadContext)

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#ifdef PROFILE\_MEM

#include "DataStructures\QuickSort.h"

#include "Memory\AutoPtr.h"

#include "core\ProfileMemory.h"

\_\_declspec(thread) MemoryProfiler \* MemoryProfiler::Instance = nullptr;

CriticalSection MemoryProfiler::s\_cs;

AutoPtr<MemoryProfiler, NoCheckHeapAllocator> MemoryProfiler::profilers(nullptr);

MemoryProfiler::MemoryProfiler() :

pageAllocator(nullptr, Js::Configuration::Global.flags, PageAllocatorType\_Max, 0, false, nullptr),

alloc(L"MemoryProfiler", &pageAllocator, Js::Throw::OutOfMemory),

arenaDataMap(&alloc, 10)

{

threadId = ::GetCurrentThreadId();

memset(&pageMemoryData, 0, sizeof(pageMemoryData));

memset(&recyclerMemoryData, 0, sizeof(recyclerMemoryData));

}

MemoryProfiler::~MemoryProfiler()

{

#if DBG

pageAllocator.SetDisableThreadAccessCheck();

#endif

if (next != nullptr)

{

NoCheckHeapDelete(next);

}

}

MemoryProfiler \*

MemoryProfiler::EnsureMemoryProfiler()

{

MemoryProfiler \* memoryProfiler = MemoryProfiler::Instance;

if (memoryProfiler == nullptr)

{

memoryProfiler = NoCheckHeapNew(MemoryProfiler);

{

AutoCriticalSection autocs(&s\_cs);

memoryProfiler->next = MemoryProfiler::profilers.Detach();

MemoryProfiler::profilers = memoryProfiler;

}

MemoryProfiler::Instance = memoryProfiler;

}

return memoryProfiler;

}

PageMemoryData \*

MemoryProfiler::GetPageMemoryData(PageAllocatorType type)

{

if (!Js::Configuration::Global.flags.IsEnabled(Js::TraceMemoryFlag))

{

return nullptr;

}

if (type == PageAllocatorType\_Max)

{

return nullptr;

}

MemoryProfiler \* memoryProfiler = EnsureMemoryProfiler();

return &memoryProfiler->pageMemoryData[type];

}

RecyclerMemoryData \*

MemoryProfiler::GetRecyclerMemoryData()

{

if (!Js::Configuration::Global.flags.IsEnabled(Js::TraceMemoryFlag))

{

return nullptr;

}

MemoryProfiler \* memoryProfiler = EnsureMemoryProfiler();

return &memoryProfiler->recyclerMemoryData;

}

ArenaMemoryData \*

MemoryProfiler::Begin(LPCWSTR name)

{

if (!Js::Configuration::Global.flags.IsEnabled(Js::TraceMemoryFlag))

{

return nullptr;

}

Assert(name != nullptr);

if (wcscmp(name, L"MemoryProfiler") == 0)

{

// Don't profile memory profiler itself

return nullptr;

}

// This is debug only code, we don't care if we catch the right exception

AUTO\_NESTED\_HANDLED\_EXCEPTION\_TYPE(ExceptionType\_DisableCheck);

MemoryProfiler \* memoryProfiler = EnsureMemoryProfiler();

ArenaMemoryDataSummary \* arenaTotalMemoryData;

if (!memoryProfiler->arenaDataMap.TryGetValue((LPWSTR)name, &arenaTotalMemoryData))

{

arenaTotalMemoryData = AnewStructZ(&memoryProfiler->alloc, ArenaMemoryDataSummary);

memoryProfiler->arenaDataMap.Add((LPWSTR)name, arenaTotalMemoryData);

}

arenaTotalMemoryData->arenaCount++;

ArenaMemoryData \* memoryData = AnewStructZ(&memoryProfiler->alloc, ArenaMemoryData);

if (arenaTotalMemoryData->data == nullptr)

{

arenaTotalMemoryData->data = memoryData;

}

else

{

memoryData->next = arenaTotalMemoryData->data;

arenaTotalMemoryData->data->prev = memoryData;

arenaTotalMemoryData->data = memoryData;

}

memoryData->profiler = memoryProfiler;

return memoryData;

}

void

MemoryProfiler::Reset(LPCWSTR name, ArenaMemoryData \* memoryData)

{

MemoryProfiler \* memoryProfiler = memoryData->profiler;

ArenaMemoryDataSummary \* arenaMemoryDataSummary;

bool hasItem = memoryProfiler->arenaDataMap.TryGetValue((LPWSTR)name, &arenaMemoryDataSummary);

Assert(hasItem);

AccumulateData(arenaMemoryDataSummary, memoryData, true);

memoryData->allocatedBytes = 0;

memoryData->alignmentBytes = 0;

memoryData->requestBytes = 0;

memoryData->requestCount = 0;

memoryData->reuseBytes = 0;

memoryData->reuseCount = 0;

memoryData->freelistBytes = 0;

memoryData->freelistCount = 0;

memoryData->resetCount++;

}

void

MemoryProfiler::End(LPCWSTR name, ArenaMemoryData \* memoryData)

{

MemoryProfiler \* memoryProfiler = memoryData->profiler;

ArenaMemoryDataSummary \* arenaMemoryDataSummary;

bool hasItem = memoryProfiler->arenaDataMap.TryGetValue((LPWSTR)name, &arenaMemoryDataSummary);

Assert(hasItem);

if (memoryData->next != nullptr)

{

memoryData->next->prev = memoryData->prev;

}

if (memoryData->prev != nullptr)

{

memoryData->prev->next = memoryData->next;

}

else

{

Assert(arenaMemoryDataSummary->data == memoryData);

arenaMemoryDataSummary->data = memoryData->next;

}

AccumulateData(arenaMemoryDataSummary, memoryData);

}

void

MemoryProfiler::AccumulateData(ArenaMemoryDataSummary \* arenaMemoryDataSummary, ArenaMemoryData \* memoryData, bool reset)

{

arenaMemoryDataSummary->total.alignmentBytes += memoryData->alignmentBytes;

arenaMemoryDataSummary->total.allocatedBytes += memoryData->allocatedBytes;

arenaMemoryDataSummary->total.freelistBytes += memoryData->freelistBytes;

arenaMemoryDataSummary->total.freelistCount += memoryData->freelistCount;

arenaMemoryDataSummary->total.requestBytes += memoryData->requestBytes;

arenaMemoryDataSummary->total.requestCount += memoryData->requestCount;

arenaMemoryDataSummary->total.reuseCount += memoryData->reuseCount;

arenaMemoryDataSummary->total.reuseBytes += memoryData->reuseBytes;

if (!reset)

{

arenaMemoryDataSummary->total.resetCount += memoryData->resetCount;

}

arenaMemoryDataSummary->max.alignmentBytes = max(arenaMemoryDataSummary->max.alignmentBytes, memoryData->alignmentBytes);

arenaMemoryDataSummary->max.allocatedBytes = max(arenaMemoryDataSummary->max.allocatedBytes, memoryData->allocatedBytes);

arenaMemoryDataSummary->max.freelistBytes = max(arenaMemoryDataSummary->max.freelistBytes, memoryData->freelistBytes);

arenaMemoryDataSummary->max.freelistCount = max(arenaMemoryDataSummary->max.freelistCount, memoryData->freelistCount);

arenaMemoryDataSummary->max.requestBytes = max(arenaMemoryDataSummary->max.requestBytes, memoryData->requestBytes);

arenaMemoryDataSummary->max.requestCount = max(arenaMemoryDataSummary->max.requestCount, memoryData->requestCount);

arenaMemoryDataSummary->max.reuseCount = max(arenaMemoryDataSummary->max.reuseCount, memoryData->reuseCount);

arenaMemoryDataSummary->max.reuseBytes = max(arenaMemoryDataSummary->max.reuseBytes, memoryData->reuseBytes);

if (!reset)

{

arenaMemoryDataSummary->max.resetCount = max(arenaMemoryDataSummary->max.resetCount, memoryData->resetCount);

}

}

void

MemoryProfiler::PrintPageMemoryData(PageMemoryData const& pageMemoryData, char const \* title)

{

if (pageMemoryData.allocSegmentCount != 0)

{

Output::Print(L"%-10S:%9d %10d | %4d %10d | %4d %10d | %10d | %10d | %10d | %10d\n", title,

pageMemoryData.currentCommittedPageCount \* AutoSystemInfo::PageSize, pageMemoryData.peakCommittedPageCount \* AutoSystemInfo::PageSize,

pageMemoryData.allocSegmentCount, pageMemoryData.allocSegmentBytes,

pageMemoryData.releaseSegmentCount, pageMemoryData.releaseSegmentBytes,

pageMemoryData.allocPageCount \* AutoSystemInfo::PageSize,

pageMemoryData.releasePageCount \* AutoSystemInfo::PageSize,

pageMemoryData.decommitPageCount \* AutoSystemInfo::PageSize,

pageMemoryData.recommitPageCount \* AutoSystemInfo::PageSize);

}

}

void

MemoryProfiler::Print()

{

Output::Print(L"-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=\n");

Output::Print(L"Allocation for thread 0x%08X\n", threadId);

Output::Print(L"-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=\n");

bool hasData = false;

for (int i = 0; i < PageAllocatorType\_Max; i++)

{

if (pageMemoryData[i].allocSegmentCount != 0)

{

hasData = true;

break;

}

}

if (hasData)

{

Output::Print(L"%-10s:%-20s | %-15s | %-15s | %10s | %10s | %11s | %11s\n", L"", L" Current", L" Alloc Seg", L" Free Seg",

L"Request", L"Released", L"Decommitted", L"Recommitted");

Output::Print(L"%-10s:%9s %10s | %4s %10s | %4s %10s | %10s | %10s | %10s | %10s\n", L"", L"Bytes", L"Peak", L"#", L"Bytes", L"#", L"Bytes",

L"Bytes", L"Bytes", L"Bytes", L"Bytes");

Output::Print(L"------------------------------------------------------------------------------------------------------------------\n");

#define PAGEALLOCATOR\_PRINT(i) PrintPageMemoryData(pageMemoryData[PageAllocatorType\_ ## i], STRINGIZE(i));

PAGE\_ALLOCATOR\_TYPE(PAGEALLOCATOR\_PRINT);

Output::Print(L"------------------------------------------------------------------------------------------------------------------\n");

}

if (recyclerMemoryData.requestCount != 0)

{

Output::Print(L"%-10s:%7s %10s %10s %10s\n",

L"Recycler",

L"#Alloc",

L"AllocBytes",

L"ReqBytes",

L"AlignByte");

Output::Print(L"--------------------------------------------------------------------------------------------------------\n");

Output::Print(L"%-10s:%7d %10d %10d %10d\n",

L"",

recyclerMemoryData.requestCount,

recyclerMemoryData.requestBytes + recyclerMemoryData.alignmentBytes,

recyclerMemoryData.requestBytes,

recyclerMemoryData.alignmentBytes);

Output::Print(L"--------------------------------------------------------------------------------------------------------\n");

}

if (Js::Configuration::Global.flags.TraceMemory.IsEnabled(Js::AllPhase))

{

PrintArena(false);

}

PrintArena(true);

}

void

MemoryProfiler::PrintArenaHeader(wchar\_t const \* title)

{

Output::Print(L"--------------------------------------------------------------------------------------------------------\n");

Output::Print(L"%-20s:%7s %9s %9s %9s %6s %9s %6s %9s %5s | %5s\n",

title,

L"#Alloc",

L"AllocByte",

L"ReqBytes",

L"AlignByte",

L"#Reuse",

L"ReuseByte",

L"#Free",

L"FreeBytes",

L"Reset",

L"Count");

Output::Print(L"--------------------------------------------------------------------------------------------------------\n");

}

int MemoryProfiler::CreateArenaUsageSummary(ArenaAllocator \* alloc, bool liveOnly,

\_Outptr\_result\_buffer\_(return) LPWSTR \*\* name\_ptr, \_Outptr\_result\_buffer\_(return) ArenaMemoryDataSummary \*\*\* summaries\_ptr)

{

Assert(alloc);

LPWSTR \*& name = \*name\_ptr;

ArenaMemoryDataSummary \*\*& summaries = \*summaries\_ptr;

int count = arenaDataMap.Count();

name = AnewArray(alloc, LPWSTR, count);

int i = 0;

arenaDataMap.Map([&i, name](LPWSTR key, ArenaMemoryDataSummary\*)

{

name[i++] = key;

});

JsUtil::QuickSort<LPWSTR, DefaultComparer<LPWSTR>>::Sort(name, name + (count - 1));

summaries = AnewArray(alloc, ArenaMemoryDataSummary \*, count);

for (int i = 0; i < count; i++)

{

ArenaMemoryDataSummary \* summary = arenaDataMap.Item(name[i]);

ArenaMemoryData \* data = summary->data;

ArenaMemoryDataSummary \* localSummary;

if (liveOnly)

{

if (data == nullptr)

{

summaries[i] = nullptr;

continue;

}

localSummary = AnewStructZ(alloc, ArenaMemoryDataSummary);

}

else

{

localSummary = Anew(alloc, ArenaMemoryDataSummary, \*summary);

}

while (data != nullptr)

{

localSummary->outstandingCount++;

AccumulateData(localSummary, data);

data = data->next;

}

if (liveOnly)

{

localSummary->arenaCount = localSummary->outstandingCount;

}

summaries[i] = localSummary;

}

return count;

}

void

MemoryProfiler::PrintArena(bool liveOnly)

{

WithArenaUsageSummary(liveOnly, [&] (int count, \_In\_reads\_(count) LPWSTR \* name, \_In\_reads\_(count) ArenaMemoryDataSummary \*\* summaries)

{

int i = 0;

if (liveOnly)

{

Output::Print(L"Arena usage summary (live)\n");

}

else

{

Output::Print(L"Arena usage summary (all)\n");

}

bool header = false;

for (i = 0; i < count; i++)

{

ArenaMemoryDataSummary \* data = summaries[i];

if (data == nullptr)

{

continue;

}

if (!header)

{

header = true;

PrintArenaHeader(L"Arena Size");

}

Output::Print(L"%-20s %7d %9d %9d %9d %6d %9d %6d %9d %5d | %5d\n",

name[i],

data->total.requestCount,

data->total.allocatedBytes,

data->total.requestBytes,

data->total.alignmentBytes,

data->total.reuseCount,

data->total.reuseBytes,

data->total.freelistCount,

data->total.freelistBytes,

data->total.resetCount,

data->arenaCount);

}

header = false;

for (i = 0; i < count; i++)

{

ArenaMemoryDataSummary \* data = summaries[i];

if (data == nullptr)

{

continue;

}

if (!header)

{

header = true;

PrintArenaHeader(L"Arena Max");

}

Output::Print(L"%-20s %7d %9d %9d %9d %6d %9d %6d %9d %5d | %5d\n",

name[i],

data->max.requestCount,

data->max.allocatedBytes,

data->max.requestBytes,

data->max.alignmentBytes,

data->max.reuseCount,

data->max.reuseBytes,

data->max.freelistCount, data->max.freelistBytes,

data->max.resetCount, data->outstandingCount);

}

header = false;

for (i = 0; i < count; i++)

{

ArenaMemoryDataSummary \* data = summaries[i];

if (data == nullptr)

{

continue;

}

if (!header)

{

header = true;

PrintArenaHeader(L"Arena Average");

}

Output::Print(L"%-20s %7d %9d %9d %9d %6d %9d %6d %9d %5d\n", name[i],

data->total.requestCount / data->arenaCount,

data->total.allocatedBytes / data->arenaCount,

data->total.requestBytes / data->arenaCount,

data->total.alignmentBytes / data->arenaCount,

data->total.reuseCount / data->arenaCount,

data->total.reuseBytes / data->arenaCount,

data->total.freelistCount / data->arenaCount,

data->total.freelistBytes / data->arenaCount,

data->total.resetCount / data->arenaCount);

}

Output::Print(L"--------------------------------------------------------------------------------------------------------\n");

});

}

void

MemoryProfiler::PrintCurrentThread()

{

MemoryProfiler\* instance = NULL;

instance = MemoryProfiler::Instance;

Output::Print(L"========================================================================================================\n");

Output::Print(L"Memory Profile (Current thread)\n");

if (instance != nullptr)

{

instance->Print();

}

Output::Flush();

}

void

MemoryProfiler::PrintAll()

{

Output::Print(L"========================================================================================================\n");

Output::Print(L"Memory Profile (All threads)\n");

ForEachProfiler([] (MemoryProfiler \* memoryProfiler)

{

memoryProfiler->Print();

});

Output::Flush();

}

bool

MemoryProfiler::IsTraceEnabled(bool isRecycler)

{

if (!Js::Configuration::Global.flags.IsEnabled(Js::TraceMemoryFlag))

{

return false;

}

if (Js::Configuration::Global.flags.TraceMemory.IsEnabled(Js::AllPhase))

{

return true;

}

if (!isRecycler)

{

return (Js::Configuration::Global.flags.TraceMemory.IsEnabled(Js::RunPhase)

|| Js::Configuration::Global.flags.TraceMemory.GetFirstPhase() == Js::InvalidPhase);

}

return Js::Configuration::Global.flags.TraceMemory.IsEnabled(Js::RecyclerPhase);

}

bool

MemoryProfiler::IsEnabled()

{

return Js::Configuration::Global.flags.IsEnabled(Js::ProfileMemoryFlag);

}

bool

MemoryProfiler::DoTrackRecyclerAllocation()

{

return MemoryProfiler::IsEnabled() || MemoryProfiler::IsTraceEnabled(true) || MemoryProfiler::IsTraceEnabled(false);

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef PROFILE\_MEM

class MemoryProfiler;

struct ArenaMemoryData

{

size\_t allocatedBytes; // memory allocated

size\_t alignmentBytes; // memory for alignment

size\_t requestBytes; // memory asked to be allocated

size\_t requestCount; // count of alloc

size\_t reuseBytes;

size\_t reuseCount;

size\_t freelistBytes; // memory asked to be freed.

size\_t freelistCount; // count of freed

size\_t resetCount;

ArenaMemoryData \* prev;

ArenaMemoryData \* next;

MemoryProfiler \* profiler;

};

struct ArenaMemoryDataSummary

{

size\_t arenaCount;

size\_t outstandingCount;

ArenaMemoryData total;

ArenaMemoryData max;

ArenaMemoryData \* data;

};

struct PageMemoryData

{

size\_t allocSegmentCount;

size\_t allocSegmentBytes;

size\_t releaseSegmentCount;

size\_t releaseSegmentBytes;

size\_t allocPageCount;

size\_t releasePageCount;

size\_t decommitPageCount;

size\_t recommitPageCount;

size\_t currentCommittedPageCount;

size\_t peakCommittedPageCount;

};

struct RecyclerMemoryData

{

size\_t allocatedBytes;

size\_t alignmentBytes;

size\_t requestBytes;

size\_t requestCount;

};

class MemoryProfiler

{

public:

MemoryProfiler();

~MemoryProfiler();

static ArenaMemoryData \* Begin(LPCWSTR name);

static RecyclerMemoryData \* GetRecyclerMemoryData();

static PageMemoryData \* GetPageMemoryData(PageAllocatorType type);

static void Reset(LPCWSTR name, ArenaMemoryData \* arena);

static void End(LPCWSTR name, ArenaMemoryData \* arena);

static void PrintAll();

static void PrintCurrentThread();

static bool IsTraceEnabled(bool isRecycler = false);

static bool IsEnabled();

static bool DoTrackRecyclerAllocation();

template<typename THandler>

static void GetArenaMemoryUsage(THandler handler)

{

ForEachProfiler([&] (MemoryProfiler \* memoryProfiler)

{

memoryProfiler->WithArenaUsageSummary(true, [&] (int count, \_In\_reads\_(count) LPWSTR \* name, \_In\_reads\_(count) ArenaMemoryDataSummary \* summaries)

{

for (int i = 0; i < count; i++)

{

ArenaMemoryDataSummary \* data = summaries[i];

if (data == nullptr)

{

continue;

}

handler(name[i], data);

}

});

});

}

private:

int MemoryProfiler::CreateArenaUsageSummary(ArenaAllocator \* alloc, bool liveOnly,

\_Outptr\_result\_buffer\_(return) LPWSTR \*\* name\_ptr, \_Outptr\_result\_buffer\_(return) ArenaMemoryDataSummary \*\*\* summaries\_ptr);

template<typename THandler>

void WithArenaUsageSummary(bool liveOnly, THandler handler);

template<typename THandler>

static void ForEachProfiler(THandler handler)

{

MemoryProfiler \* memoryProfiler = MemoryProfiler::profilers;

while (memoryProfiler != nullptr)

{

handler(memoryProfiler);

memoryProfiler = memoryProfiler->next;

}

}

static MemoryProfiler \* EnsureMemoryProfiler();

static void AccumulateData(ArenaMemoryDataSummary \*, ArenaMemoryData \*, bool reset = false);

void Print();

void PrintArenaHeader(wchar\_t const \* title);

static void PrintPageMemoryData(PageMemoryData const& pageMemoryData, char const \* title);

private:

void PrintArena(bool liveOnly);

static \_\_declspec(thread) MemoryProfiler \* Instance;

static CriticalSection s\_cs;

static AutoPtr<MemoryProfiler, NoCheckHeapAllocator> profilers;

PageAllocator pageAllocator;

ArenaAllocator alloc;

JsUtil::BaseDictionary<LPWSTR, ArenaMemoryDataSummary \*, ArenaAllocator, PrimeSizePolicy> arenaDataMap;

PageMemoryData pageMemoryData[PageAllocatorType\_Max];

RecyclerMemoryData recyclerMemoryData;

MemoryProfiler \* next;

DWORD threadId;

};

template<typename THandler>

void

MemoryProfiler::WithArenaUsageSummary(bool liveOnly, THandler handler)

{

// This is debug only code, we don't care if we catch the right exception

AUTO\_NESTED\_HANDLED\_EXCEPTION\_TYPE(ExceptionType\_DisableCheck);

PageAllocator tempPageAlloc(nullptr, Js::Configuration::Global.flags);

ArenaAllocator tempAlloc(L"MemoryProfiler", &tempPageAlloc, Js::Throw::OutOfMemory);

LPWSTR \* name = nullptr;

ArenaMemoryDataSummary \*\* summaries = nullptr;

int count = CreateArenaUsageSummary(&tempAlloc, liveOnly, &name, &summaries);

handler(count, name, summaries);

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#ifdef STACK\_BACK\_TRACE

#include "core\StackBackTrace.h"

#include "core\DbgHelpSymbolManager.h"

StackBackTrace::StackBackTrace(ULONG framesToSkip, ULONG framesToCapture) : requestedFramesToCapture(framesToCapture)

{

this->Capture(framesToSkip);

}

// Don't capture, just remember requestedFramesToCapture/allocate buffer for them.

StackBackTrace::StackBackTrace(ULONG framesToCaptureLater) : requestedFramesToCapture(framesToCaptureLater), framesCount(0)

{

}

// This can be called multiple times, together with Create, in which case we will use (overwrite) same buffer.

ULONG StackBackTrace::Capture(ULONG framesToSkip)

{

this->framesCount = CaptureStackBackTrace(framesToSkip + BaseFramesToSkip, this->requestedFramesToCapture, this->stackBackTrace, NULL);

return this->framesCount;

}

size\_t

StackBackTrace::Print()

{

DbgHelpSymbolManager::EnsureInitialized();

size\_t retValue = 0;

for(ULONG i = 0; i < this->framesCount; i++)

{

PVOID address = this->stackBackTrace[i];

retValue += Output::Print(L" ");

retValue += DbgHelpSymbolManager::PrintSymbol(address);

retValue += Output::Print(L"\n");

}

retValue += Output::Print(L"\n");

return retValue;

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef STACK\_BACK\_TRACE

class StackBackTrace

{

public:

static const ULONG DefaultFramesToCapture = 30;

template <typename TAllocator> \_\_declspec(noinline)

static StackBackTrace \* Capture(TAllocator \* alloc, ULONG framesToSkip = 0, ULONG framesToCapture = DefaultFramesToCapture);

template <typename TAllocator> \_\_declspec(noinline)

static StackBackTrace \* Create(TAllocator \* alloc, ULONG framesToCaptureLater = DefaultFramesToCapture);

size\_t Print();

template<typename Fn>void Map(Fn fn); // The Fn is expected to be: void Fn(void\*).

ULONG Capture(ULONG framesToSkip);

ULONG GetRequestedFrameCount() { return this->requestedFramesToCapture; }

template <typename TAllocator>

void Delete(TAllocator \* alloc);

private:

// We want to skip at lease the StackBackTrace::Capture and the constructor frames

static const ULONG BaseFramesToSkip = 2;

\_\_declspec(noinline) StackBackTrace(ULONG framesToSkip, ULONG framesToCapture);

\_\_declspec(noinline) StackBackTrace(ULONG framesToCapture);

ULONG requestedFramesToCapture;

ULONG framesCount;

PVOID stackBackTrace[];

};

template <typename TAllocator>

StackBackTrace \*

StackBackTrace::Capture(TAllocator \* alloc, ULONG framesToSkip, ULONG framesToCapture)

{

return AllocatorNewPlusZ(TAllocator, alloc, sizeof(PVOID) \* framesToCapture, StackBackTrace, framesToSkip, framesToCapture);

}

template <typename TAllocator>

StackBackTrace\* StackBackTrace::Create(TAllocator \* alloc, ULONG framesToCaptureLater)

{

return AllocatorNewPlusZ(TAllocator, alloc, sizeof(PVOID)\* framesToCaptureLater, StackBackTrace, framesToCaptureLater);

}

template <typename TAllocator>

void StackBackTrace::Delete(TAllocator \* alloc)

{

AllocatorDeletePlus(TAllocator, alloc, sizeof(PVOID)\* requestedFramesToCapture, this);

}

template <typename Fn>

void StackBackTrace::Map(Fn fn)

{

for (ULONG i = 0; i < this->framesCount; ++i)

{

fn(this->stackBackTrace[i]);

}

}

class StackBackTraceNode

{

public:

template <typename TAllocator>

static void Prepend(TAllocator \* allocator, StackBackTraceNode \*& head, StackBackTrace \* stackBackTrace)

{

head = AllocatorNew(TAllocator, allocator, StackBackTraceNode, stackBackTrace, head);

}

template <typename TAllocator>

static void DeleteAll(TAllocator \* allocator, StackBackTraceNode \*& head)

{

StackBackTraceNode \* curr = head;

while (curr != nullptr)

{

StackBackTraceNode \* next = curr->next;

curr->stackBackTrace->Delete(allocator);

AllocatorDelete(TAllocator, allocator, curr);

curr = next;

}

head = nullptr;

}

static void PrintAll(StackBackTraceNode \* head)

{

// We want to print them tail first because that is the first stack trace we added

// Reverse the list

StackBackTraceNode \* curr = head;

StackBackTraceNode \* prev = nullptr;

while (curr != nullptr)

{

StackBackTraceNode \* next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

// print and reverse again.

curr = prev;

prev = nullptr;

while (curr != nullptr)

{

curr->stackBackTrace->Print();

StackBackTraceNode \* next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

Assert(prev == head);

}

private:

StackBackTraceNode(StackBackTrace \* stackBackTrace, StackBackTraceNode \* next) : stackBackTrace(stackBackTrace), next(next) {};

StackBackTrace \* stackBackTrace;

StackBackTraceNode \* next;

};

//

// Capture multiple call stack traces using a in-memory ring buffer. Useful for instrumenting source

// code to track calls.

//

// BUFFERS: Number of stack traces to keep. When all the buffers are filled up, capture will start

// over from the beginning and overwrite older traces.

// HEADER: Number of pointer-sized data reserved in the header of each trace. You can save runtime

// data in the header of each trace to record runtime state of the stack trace.

// FRAMES: Number of stack frames for each trace.

// SKIPFRAMES: Top frames to skip for each capture. e.g., at least StackBackTraceRing::Capture frame is useless.

//

// Usage: Following captures the last 100 stacks that changes scriptContext->debuggerMode:

// Declare an instance: StackBackTraceRing<100> s\_debuggerMode;

// Call at every debuggerMode change point: s\_debugModeTrace.Capture(scriptContext, debuggerMode);

//

// x86:

// Debug a dump in windbg: ?? chakra!Js::s\_debuggerMode

// Inspect trace 0: dds [buf]

// Inspect trace N: dds [buf]+0n32\*4\*N

// Inspect last trace: dds [buf]+0n32\*4\*[cur-1]

//

template <ULONG BUFFERS, ULONG HEADER = 2, ULONG FRAMES = 30, ULONG SKIPFRAMES = 1>

class StackBackTraceRing

{

static const ULONG ONE\_TRACE = HEADER + FRAMES;

private:

LPVOID\* buf;

ULONG cur;

public:

StackBackTraceRing()

{

buf = new LPVOID[ONE\_TRACE \* BUFFERS];

cur = 0;

}

~StackBackTraceRing()

{

delete[] buf;

}

template <class HeaderFunc>

void CaptureWithHeader(HeaderFunc writeHeader)

{

LPVOID\* buffer = &buf[ONE\_TRACE \* cur++];

cur = cur % BUFFERS;

memset(buffer, 0, sizeof(LPVOID) \* ONE\_TRACE);

writeHeader(buffer);

LPVOID\* frames = &buffer[HEADER];

CaptureStackBackTrace(SKIPFRAMES, FRAMES, frames, nullptr);

}

// Capture a stack trace

void Capture()

{

CaptureWithHeader([](LPVOID\* buffer)

{

});

}

// Capture a stack trace and save data0 in header

template <class T0>

void Capture(T0 data0)

{

C\_ASSERT(HEADER >= 1);

CaptureWithHeader([=](\_Out\_writes\_(HEADER) LPVOID\* buffer)

{

buffer[0] = reinterpret\_cast<LPVOID>(data0);

});

}

// Capture a stack trace and save data0 and data1 in header

template <class T0, class T1>

void Capture(T0 data0, T1 data1)

{

C\_ASSERT(HEADER >= 2);

CaptureWithHeader([=](\_Out\_writes\_(HEADER) LPVOID\* buffer)

{

buffer[0] = reinterpret\_cast<LPVOID>(data0);

buffer[1] = reinterpret\_cast<LPVOID>(data1);

});

}

};

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonCorePch.h"

#include <psapi.h>

#include <Wincrypt.h>

#include <VersionHelpers.h>

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAB")

EXTERN\_C IMAGE\_DOS\_HEADER \_\_ImageBase;

AutoSystemInfo AutoSystemInfo::Data;

void

AutoSystemInfo::Initialize()

{

Assert(!initialized);

processHandle = GetCurrentProcess();

GetSystemInfo(this);

// Make the page size constant so calculation are faster.

Assert(this->dwPageSize == AutoSystemInfo::PageSize);

#if defined(\_M\_IX86) || defined(\_M\_X64)

\_\_cpuid(CPUInfo, 1);

isAtom = CheckForAtom();

#endif

#if defined(\_M\_ARM32\_OR\_ARM64)

armDivAvailable = IsProcessorFeaturePresent(PF\_ARM\_DIVIDE\_INSTRUCTION\_AVAILABLE) ? true : false;

#endif

allocationGranularityPageCount = dwAllocationGranularity / dwPageSize;

isWindows8OrGreater = IsWindows8OrGreater();

binaryName[0] = L'\0';

dllLoadAddress = (UINT\_PTR)&\_\_ImageBase;

dllHighAddress = (UINT\_PTR)&\_\_ImageBase +

((PIMAGE\_NT\_HEADERS)(((char \*)&\_\_ImageBase) + \_\_ImageBase.e\_lfanew))->OptionalHeader.SizeOfImage;

InitPhysicalProcessorCount();

#if DBG

initialized = true;

#endif

WCHAR DisableDebugScopeCaptureFlag[MAX\_PATH];

if (::GetEnvironmentVariable(L"JS\_DEBUG\_SCOPE", DisableDebugScopeCaptureFlag, \_countof(DisableDebugScopeCaptureFlag)) != 0)

{

disableDebugScopeCapture = true;

}

else

{

disableDebugScopeCapture = false;

}

this->shouldQCMoreFrequently = false;

this->supportsOnlyMultiThreadedCOM = false;

this->isLowMemoryDevice = false;

// 0 indicates we haven't retrieved the available commit. We get it lazily.

this->availableCommit = 0;

::ChakraBinaryAutoSystemInfoInit(this);

}

bool

AutoSystemInfo::InitPhysicalProcessorCount()

{

DWORD size = 0;

DWORD countPhysicalProcessor = 0;

PSYSTEM\_LOGICAL\_PROCESSOR\_INFORMATION pBufferCurrent;

PSYSTEM\_LOGICAL\_PROCESSOR\_INFORMATION pBufferStart;

BOOL bResult;

Assert(!this->initialized);

// Initialize physical processor to number of logical processors.

// If anything below fails, we still need an approximate value

this->dwNumberOfPhyscialProcessors = this->dwNumberOfProcessors;

bResult = GetLogicalProcessorInformation(NULL, &size);

if (bResult || GetLastError() != ERROR\_INSUFFICIENT\_BUFFER || !size)

{

return false;

}

DWORD count = (size) / sizeof(SYSTEM\_LOGICAL\_PROCESSOR\_INFORMATION);

if (size != count \* sizeof(SYSTEM\_LOGICAL\_PROCESSOR\_INFORMATION))

{

Assert(false);

return false;

}

pBufferCurrent = pBufferStart = NoCheckHeapNewArray(SYSTEM\_LOGICAL\_PROCESSOR\_INFORMATION, (size\_t)count);

if (!pBufferCurrent)

{

return false;

}

bResult = GetLogicalProcessorInformation(pBufferCurrent, &size);

if (!bResult)

{

return false;

}

while (pBufferCurrent < (pBufferStart + count))

{

if (pBufferCurrent->Relationship == RelationProcessorCore)

{

countPhysicalProcessor++;

}

pBufferCurrent++;

}

NoCheckHeapDeleteArray(count, pBufferStart);

this->dwNumberOfPhyscialProcessors = countPhysicalProcessor;

return true;

}

bool

AutoSystemInfo::IsJscriptModulePointer(void \* ptr)

{

return ((UINT\_PTR)ptr >= Data.dllLoadAddress && (UINT\_PTR)ptr < Data.dllHighAddress);

}

uint

AutoSystemInfo::GetAllocationGranularityPageCount() const

{

Assert(initialized);

return allocationGranularityPageCount;

}

uint

AutoSystemInfo::GetAllocationGranularityPageSize() const

{

Assert(initialized);

return allocationGranularityPageCount \* PageSize;

}

#if defined(\_M\_IX86) || defined(\_M\_X64)

bool

AutoSystemInfo::VirtualSseAvailable(const int sseLevel) const

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

return CONFIG\_FLAG(Sse) < 0 || CONFIG\_FLAG(Sse) >= sseLevel;

#else

return true;

#endif

}

#endif

BOOL

AutoSystemInfo::SSE2Available() const

{

Assert(initialized);

#if defined(\_M\_X64) || defined(\_M\_ARM32\_OR\_ARM64)

return true;

#elif defined(\_M\_IX86)

return VirtualSseAvailable(2) && (CPUInfo[3] & (1 << 26));

#else

#error Unsupported platform.

#endif

}

#if defined(\_M\_IX86) || defined(\_M\_X64)

BOOL

AutoSystemInfo::SSE3Available() const

{

Assert(initialized);

return VirtualSseAvailable(3) && (CPUInfo[2] & 0x1);

}

BOOL

AutoSystemInfo::SSE4\_1Available() const

{

Assert(initialized);

return VirtualSseAvailable(4) && (CPUInfo[2] & (0x1 << 19));

}

BOOL

AutoSystemInfo::PopCntAvailable() const

{

Assert(initialized);

return VirtualSseAvailable(4) && (CPUInfo[2] & (1 << 23));

}

BOOL

AutoSystemInfo::LZCntAvailable() const

{

Assert(initialized);

int CPUInfo[4];

\_\_cpuid(CPUInfo, 0x80000001);

return VirtualSseAvailable(4) && (CPUInfo[2] & (1 << 5));

}

bool

AutoSystemInfo::IsAtomPlatform() const

{

return isAtom;

}

bool

AutoSystemInfo::CheckForAtom() const

{

int CPUInfo[4];

const int GENUINE\_INTEL\_0 = 0x756e6547,

GENUINE\_INTEL\_1 = 0x49656e69,

GENUINE\_INTEL\_2 = 0x6c65746e;

const int PLATFORM\_MASK = 0x0fff3ff0;

const int ATOM\_PLATFORM\_A = 0x0106c0, /\* bonnell - extended model 1c, type 0, family code 6 \*/

ATOM\_PLATFORM\_B = 0x020660, /\* lincroft - extended model 26, type 0, family code 6 \*/

ATOM\_PLATFORM\_C = 0x020670, /\* saltwell - extended model 27, type 0, family code 6 \*/

ATOM\_PLATFORM\_D = 0x030650, /\* tbd - extended model 35, type 0, family code 6 \*/

ATOM\_PLATFORM\_E = 0x030660, /\* tbd - extended model 36, type 0, family code 6 \*/

ATOM\_PLATFORM\_F = 0x030670; /\* tbd - extended model 37, type 0, family code 6 \*/

int platformSignature;

\_\_cpuid(CPUInfo, 0);

// See if CPU is ATOM HW. First check if CPU is genuine Intel.

if( CPUInfo[1]==GENUINE\_INTEL\_0 &&

CPUInfo[3]==GENUINE\_INTEL\_1 &&

CPUInfo[2]==GENUINE\_INTEL\_2)

{

\_\_cpuid(CPUInfo, 1);

// get platform signature

platformSignature = CPUInfo[0];

if((( PLATFORM\_MASK & platformSignature) == ATOM\_PLATFORM\_A) ||

((PLATFORM\_MASK & platformSignature) == ATOM\_PLATFORM\_B) ||

((PLATFORM\_MASK & platformSignature) == ATOM\_PLATFORM\_C) ||

((PLATFORM\_MASK & platformSignature) == ATOM\_PLATFORM\_D) ||

((PLATFORM\_MASK & platformSignature) == ATOM\_PLATFORM\_E) ||

((PLATFORM\_MASK & platformSignature) == ATOM\_PLATFORM\_F))

{

return true;

}

}

return false;

}

#endif

bool

AutoSystemInfo::IsCFGEnabled()

{

#if defined(\_CONTROL\_FLOW\_GUARD)

return true

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

&& IsWinThresholdOrLater() && !PHASE\_OFF1(Js::CFGPhase)

#endif //ENABLE\_DEBUG\_CONFIG\_OPTIONS

;

#else

return false;

#endif //\_CONTROL\_FLOW\_GUARD

}

bool

AutoSystemInfo::IsWin8OrLater()

{

return isWindows8OrGreater;

}

#if defined(\_CONTROL\_FLOW\_GUARD)

bool

AutoSystemInfo::IsWinThresholdOrLater()

{

return IsWindowsThresholdOrGreater();

}

#endif

DWORD AutoSystemInfo::SaveModuleFileName(HANDLE hMod)

{

return ::GetModuleFileNameW((HMODULE)hMod, Data.binaryName, MAX\_PATH);

}

LPCWSTR AutoSystemInfo::GetJscriptDllFileName()

{

return (LPCWSTR)Data.binaryName;

}

bool AutoSystemInfo::IsLowMemoryProcess()

{

ULONG64 commit = ULONG64(-1);

this->GetAvailableCommit(&commit);

return commit <= CONFIG\_FLAG(LowMemoryCap);

}

BOOL AutoSystemInfo::GetAvailableCommit(ULONG64 \*pCommit)

{

Assert(initialized);

// Non-zero value indicates we've been here before.

if (this->availableCommit == 0)

{

return false;

}

\*pCommit = this->availableCommit;

return true;

}

void AutoSystemInfo::SetAvailableCommit(ULONG64 commit)

{

::InterlockedCompareExchange64((volatile LONG64 \*)&this->availableCommit, commit, 0);

}

//

// Returns the major and minor version of the loaded binary. If the version info has been fetched once, it will be cached

// and returned without any system calls to find the version number.

//

HRESULT AutoSystemInfo::GetJscriptFileVersion(DWORD\* majorVersion, DWORD\* minorVersion, DWORD \*buildDateHash, DWORD \*buildTimeHash)

{

HRESULT hr = E\_FAIL;

if(AutoSystemInfo::Data.majorVersion == 0 && AutoSystemInfo::Data.minorVersion == 0)

{

// uninitialized state - call the system API to get the version info.

LPCWSTR jscriptDllName = GetJscriptDllFileName();

hr = GetVersionInfo(jscriptDllName, majorVersion, minorVersion);

AutoSystemInfo::Data.majorVersion = \*majorVersion;

AutoSystemInfo::Data.minorVersion = \*minorVersion;

}

else if(AutoSystemInfo::Data.majorVersion != INVALID\_VERSION)

{

// if the cached copy is valid, use it and return S\_OK.

\*majorVersion = AutoSystemInfo::Data.majorVersion;

\*minorVersion = AutoSystemInfo::Data.minorVersion;

hr = S\_OK;

}

if (buildDateHash)

{

\*buildDateHash = AutoSystemInfo::Data.buildDateHash;

}

if (buildTimeHash)

{

\*buildTimeHash = AutoSystemInfo::Data.buildTimeHash;

}

return hr;

}

//

// Returns the major and minor version of the binary passed as argument.

//

HRESULT AutoSystemInfo::GetVersionInfo(\_\_in LPCWSTR pszPath, DWORD\* majorVersion, DWORD\* minorVersion)

{

DWORD dwTemp;

DWORD cbVersionSz;

HRESULT hr = E\_FAIL;

BYTE\* pVerBuffer = NULL;

VS\_FIXEDFILEINFO\* pFileInfo = NULL;

cbVersionSz = GetFileVersionInfoSizeEx(FILE\_VER\_GET\_LOCALISED, pszPath, &dwTemp);

if(cbVersionSz > 0)

{

pVerBuffer = NoCheckHeapNewArray(BYTE, cbVersionSz);

if(pVerBuffer)

{

if(GetFileVersionInfoEx(FILE\_VER\_GET\_LOCALISED|FILE\_VER\_GET\_NEUTRAL, pszPath, 0, cbVersionSz, pVerBuffer))

{

UINT uiSz = sizeof(VS\_FIXEDFILEINFO);

if(!VerQueryValue(pVerBuffer, L"\\", (LPVOID\*)&pFileInfo, &uiSz))

{

hr = HRESULT\_FROM\_WIN32(GetLastError());

}

else

{

hr = S\_OK;

}

}

else

{

hr = HRESULT\_FROM\_WIN32(GetLastError());

}

}

else

{

hr = E\_OUTOFMEMORY;

}

}

if(SUCCEEDED(hr))

{

\*majorVersion = pFileInfo->dwFileVersionMS;

\*minorVersion = pFileInfo->dwFileVersionLS;

}

else

{

\*majorVersion = INVALID\_VERSION;

\*minorVersion = INVALID\_VERSION;

}

if(pVerBuffer)

{

NoCheckHeapDeleteArray(cbVersionSz, pVerBuffer);

}

return hr;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

//----------------------------------------------------------------------------

// Automatic system info getter at startup

//----------------------------------------------------------------------------

class AutoSystemInfo : public SYSTEM\_INFO

{

friend void ChakraBinaryAutoSystemInfoInit(AutoSystemInfo \*); // The hosting DLL provides the implementation of this function.

public:

static AutoSystemInfo Data;

uint GetAllocationGranularityPageCount() const;

uint GetAllocationGranularityPageSize() const;

HANDLE GetProcessHandle() { return this->processHandle; }

bool DisableDebugScopeCapture() const { return this->disableDebugScopeCapture; }

bool IsCFGEnabled();

bool IsWin8OrLater();

#if defined(\_CONTROL\_FLOW\_GUARD)

bool IsWinThresholdOrLater();

#endif

#if defined(\_M\_IX86) || defined(\_M\_X64)

bool VirtualSseAvailable(const int sseLevel) const;

#endif

BOOL SSE2Available() const;

#if defined(\_M\_IX86) || defined(\_M\_X64)

BOOL SSE3Available() const;

BOOL SSE4\_1Available() const;

BOOL PopCntAvailable() const;

BOOL LZCntAvailable() const;

bool IsAtomPlatform() const;

#endif

bool IsLowMemoryProcess();

BOOL GetAvailableCommit(ULONG64 \*pCommit);

void SetAvailableCommit(ULONG64 commit);

DWORD GetNumberOfLogicalProcessors() const { return this->dwNumberOfProcessors; }

DWORD GetNumberOfPhysicalProcessors() const { return this->dwNumberOfPhyscialProcessors; }

#if defined(\_M\_ARM32\_OR\_ARM64)

bool ArmDivAvailable() const { return this->armDivAvailable; }

#endif

static DWORD SaveModuleFileName(HANDLE hMod);

static LPCWSTR GetJscriptDllFileName();

static HRESULT GetJscriptFileVersion(DWORD\* majorVersion, DWORD\* minorVersion, DWORD \*buildDateHash = nullptr, DWORD \*buildTimeHash = nullptr);

#if DBG

static bool IsInitialized() { return AutoSystemInfo::Data.initialized; }

#endif

static bool IsJscriptModulePointer(void \* ptr);

static DWORD const PageSize = 4096;

#ifdef STACK\_ALIGN

static DWORD const StackAlign = STACK\_ALIGN;

#else

# if defined(\_WIN64)

static DWORD const StackAlign = 16;

# elif defined(\_M\_ARM)

static DWORD const StackAlign = 8;

# elif defined(\_M\_IX86)

static DWORD const StackAlign = 4;

# else

# error\_missing\_target

# endif

#endif

UINT\_PTR dllLoadAddress;

UINT\_PTR dllHighAddress;

private:

AutoSystemInfo() : majorVersion(0), minorVersion(0), buildDateHash(0), buildTimeHash(0) { Initialize(); }

void Initialize();

bool isWindows8OrGreater;

uint allocationGranularityPageCount;

HANDLE processHandle;

#if defined(\_M\_IX86) || defined(\_M\_X64)

int CPUInfo[4];

#endif

#if defined(\_M\_ARM32\_OR\_ARM64)

bool armDivAvailable;

#endif

DWORD dwNumberOfPhyscialProcessors;

bool disableDebugScopeCapture;

#if DBG

bool initialized;

#endif

private:

#if defined(\_M\_IX86) || defined(\_M\_X64)

bool isAtom;

bool CheckForAtom() const;

#endif

bool InitPhysicalProcessorCount();

WCHAR binaryName[MAX\_PATH + 1];

DWORD majorVersion;

DWORD minorVersion;

DWORD buildDateHash;

DWORD buildTimeHash;

static HRESULT GetVersionInfo(\_\_in LPCWSTR pszPath, DWORD\* majorVersion, DWORD\* minorVersion);

static const DWORD INVALID\_VERSION = (DWORD)-1;

ULONG64 availableCommit;

bool shouldQCMoreFrequently;

bool supportsOnlyMultiThreadedCOM;

bool isLowMemoryDevice;

public:

static bool ShouldQCMoreFrequently()

{

return Data.shouldQCMoreFrequently;

}

static bool SupportsOnlyMultiThreadedCOM()

{

return Data.supportsOnlyMultiThreadedCOM;

}

static bool IsLowMemoryDevice()

{

return Data.isLowMemoryDevice;

}

};

// For Prefast where it doesn't like symbolic constants

CompileAssert(AutoSystemInfo::PageSize == 4096);

#define \_\_in\_ecount\_pagesize \_\_in\_ecount(4096)

#define \_\_in\_ecount\_twopagesize \_\_in\_ecount(8192)

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

//////////////////////////////////////////////////////////////////////////

// Template implementation of dictionary based on .NET BCL implementation.

//

// Buckets and entries are allocated as contiguous arrays to maintain good locality of reference.

//

// COLLISION STRATEGY

// This dictionary uses a chaining collision resolution strategy. Chains are implemented using indexes to the 'buckets' array.

//

// STORAGE (TAllocator)

// This dictionary works for both arena and recycler based allocation using TAllocator template parameter.

// It supports storing of both value and pointer types. Using template specialization, value types (built-in fundamental

// types and structs) are stored as leaf nodes by default.

//

// INITIAL SIZE and BUCKET MAPPING (SizePolicy)

// This can be specified using TSizePolicy template parameter. There are 2 implementations:

// - PrimeSizePolicy (Better distribution): Initial size is a prime number. Mapping to bucket is done using modulus operation (costlier).

// - PowerOf2SizePolicy (faster): Initial size is a power of 2. Mapping to bucket is done by a fast truncating the MSB bits up to the size of the table.

//

// COMPARISONS AND HASHCODE (Comparer)

// Enables custom comparisons for TKey and TValue. For example, for strings we use string comparison instead of comparing pointers.

//

#if PROFILE\_DICTIONARY

#include "DictionaryStats.h"

#endif

namespace Js

{

template <class TDictionary>

class RemoteDictionary;

}

namespace JsDiag

{

template <class TDictionary>

struct RemoteDictionary;

}

namespace JsUtil

{

class NoResizeLock

{

public:

void BeginResize() {}

void EndResize() {}

};

class AsymetricResizeLock

{

public:

void BeginResize() { cs.Enter(); }

void EndResize() { cs.Leave(); }

void LockResize() { cs.Enter(); }

void UnlockResize() { cs.Leave(); }

private:

CriticalSection cs;

};

template <class TKey, class TValue> class SimpleDictionaryEntry;

template <

class TKey,

class TValue,

class TAllocator,

class SizePolicy = PowerOf2SizePolicy,

template <typename ValueOrKey> class Comparer = DefaultComparer,

template <typename K, typename V> class Entry = SimpleDictionaryEntry,

typename Lock = NoResizeLock

>

class BaseDictionary : protected Lock

{

public:

typedef TKey KeyType;

typedef TValue ValueType;

typedef typename AllocatorInfo<TAllocator, TValue>::AllocatorType AllocatorType;

typedef SizePolicy CurrentSizePolicy;

typedef Entry<TKey, TValue> EntryType;

template<class TDictionary> class EntryIterator;

template<class TDictionary> class BucketEntryIterator;

protected:

typedef typename AllocatorInfo<TAllocator, TValue>::AllocatorFunc EntryAllocatorFuncType;

friend class Js::RemoteDictionary<BaseDictionary>;

template <typename ValueOrKey> struct ComparerType { typedef Comparer<ValueOrKey> Type; }; // Used by diagnostics to access Comparer type

int\* buckets;

EntryType\* entries;

AllocatorType\* alloc;

int size;

uint bucketCount;

int count;

int freeList;

int freeCount;

#if PROFILE\_DICTIONARY

DictionaryStats \*stats;

#endif

enum InsertOperations

{

Insert\_Add , // FatalInternalError if the item already exist in debug build

Insert\_AddNew, // Ignore add if the item already exist

Insert\_Item // Replace the item if it already exist

};

class AutoDoResize

{

public:

AutoDoResize(Lock& lock) : lock(lock) { lock.BeginResize(); };

~AutoDoResize() { lock.EndResize(); };

private:

Lock& lock;

};

public:

BaseDictionary(AllocatorType\* allocator, int capacity = 0)

: buckets (nullptr),

size(0),

bucketCount(0),

entries(nullptr),

count(0),

freeCount(0),

alloc(allocator)

{

Assert(allocator);

#if PROFILE\_DICTIONARY

stats = nullptr;

#endif

// If initial capacity is negative or 0, lazy initialization on

// the first insert operation is performed.

if (capacity > 0)

{

Initialize(capacity);

}

}

BaseDictionary(const BaseDictionary &other) : alloc(other.alloc)

{

if(other.Count() == 0)

{

size = 0;

bucketCount = 0;

buckets = nullptr;

entries = nullptr;

count = 0;

freeCount = 0;

#if PROFILE\_DICTIONARY

stats = nullptr;

#endif

return;

}

Assert(other.bucketCount != 0);

Assert(other.size != 0);

buckets = AllocateBuckets(other.bucketCount);

Assert(buckets); // no-throw allocators are currently not supported

try

{

entries = AllocateEntries(other.size, false /\* zeroAllocate \*/);

Assert(entries); // no-throw allocators are currently not supported

}

catch(...)

{

DeleteBuckets(buckets, other.bucketCount);

throw;

}

size = other.size;

bucketCount = other.bucketCount;

count = other.count;

freeList = other.freeList;

freeCount = other.freeCount;

size\_t copySize = bucketCount \* sizeof(buckets[0]);

js\_memcpy\_s(buckets, copySize, other.buckets, copySize);

copySize = size \* sizeof(entries[0]);

js\_memcpy\_s(entries, copySize, other.entries, copySize);

#if PROFILE\_DICTIONARY

stats = DictionaryStats::Create(typeid(this).name(), size);

#endif

}

~BaseDictionary()

{

if (buckets)

{

DeleteBuckets(buckets, bucketCount);

}

if (entries)

{

DeleteEntries(entries, size);

}

}

AllocatorType \*GetAllocator() const

{

return alloc;

}

inline int Capacity() const

{

return size;

}

inline int Count() const

{

return count - freeCount;

}

TValue Item(const TKey& key)

{

int i = FindEntry(key);

Assert(i >= 0);

return entries[i].Value();

}

int Add(const TKey& key, const TValue& value)

{

return Insert<Insert\_Add>(key, value);

}

int AddNew(const TKey& key, const TValue& value)

{

return Insert<Insert\_AddNew>(key, value);

}

int Item(const TKey& key, const TValue& value)

{

return Insert<Insert\_Item>(key, value);

}

bool Contains(KeyValuePair<TKey, TValue> keyValuePair)

{

int i = FindEntry(keyValuePair.Key());

if( i >= 0 && Comparer<TValue>::Equals(entries[i].Value(), keyValuePair.Value()))

{

return true;

}

return false;

}

bool Remove(KeyValuePair<TKey, TValue> keyValuePair)

{

int i, last;

uint targetBucket;

if(FindEntryWithKey(keyValuePair.Key(), &i, &last, &targetBucket))

{

const TValue &value = entries[i].Value();

if(Comparer<TValue>::Equals(value, keyValuePair.Value()))

{

RemoveAt(i, last, targetBucket);

return true;

}

}

return false;

}

void Clear()

{

if (count > 0)

{

memset(buckets, -1, bucketCount \* sizeof(buckets[0]));

memset(entries, 0, sizeof(EntryType) \* size);

count = 0;

freeCount = 0;

#if PROFILE\_DICTIONARY

// To not loose previously collected data, we will treat cleared dictionary as a separate instance for stats tracking purpose

stats = DictionaryStats::Create(typeid(this).name(), size);

#endif

}

}

void ResetNoDelete()

{

this->size = 0;

this->bucketCount = 0;

this->buckets = nullptr;

this->entries = nullptr;

this->count = 0;

this->freeCount = 0;

}

void Reset()

{

if(bucketCount != 0)

{

DeleteBuckets(buckets, bucketCount);

buckets = nullptr;

bucketCount = 0;

}

else

{

Assert(!buckets);

}

if(size != 0)

{

DeleteEntries(entries, size);

entries = nullptr;

freeCount = count = size = 0;

}

else

{

Assert(!entries);

Assert(count == 0);

Assert(freeCount == 0);

}

}

bool ContainsKey(const TKey& key) const

{

return FindEntry(key) >= 0;

}

template <typename TLookup>

inline const TValue& LookupWithKey(const TLookup& key, const TValue& defaultValue) const

{

int i = FindEntryWithKey(key);

if (i >= 0)

{

return entries[i].Value();

}

return defaultValue;

}

inline const TValue& Lookup(const TKey& key, const TValue& defaultValue)

{

return LookupWithKey<TKey>(key, defaultValue);

}

template <typename TLookup>

bool TryGetValue(const TLookup& key, TValue\* value) const

{

int i = FindEntryWithKey(key);

if (i >= 0)

{

\*value = entries[i].Value();

return true;

}

return false;

}

bool TryGetValueAndRemove(const TKey& key, TValue\* value)

{

int i, last;

uint targetBucket;

if (FindEntryWithKey(key, &i, &last, &targetBucket))

{

\*value = entries[i].Value();

RemoveAt(i, last, targetBucket);

return true;

}

return false;

}

template <typename TLookup>

bool TryGetReference(const TLookup& key, const TValue\*\* value) const

{

int i;

return TryGetReference(key, value, &i);

}

template <typename TLookup>

bool TryGetReference(const TLookup& key, TValue\*\* value) const

{

return TryGetReference(key, const\_cast<const TValue \*\*>(value));

}

template <typename TLookup>

bool TryGetReference(const TLookup& key, const TValue\*\* value, int\* index) const

{

int i = FindEntryWithKey(key);

if (i >= 0)

{

\*value = &entries[i].Value();

\*index = i;

return true;

}

return false;

}

template <typename TLookup>

bool TryGetReference(const TLookup& key, TValue\*\* value, int\* index) const

{

return TryGetReference(key, const\_cast<const TValue \*\*>(value), index);

}

const TValue& GetValueAt(const int index) const

{

Assert(index >= 0);

Assert(index < count);

return entries[index].Value();

}

TValue\* GetReferenceAt(const int index) const

{

Assert(index >= 0);

Assert(index < count);

return &entries[index].Value();

}

TKey const& GetKeyAt(const int index) const

{

Assert(index >= 0);

Assert(index < count);

return entries[index].Key();

}

bool TryGetValueAt(const int index, TValue const \*\* value) const

{

if (index >= 0 && index < count)

{

\*value = &entries[index].Value();

return true;

}

return false;

}

bool TryGetValueAt(int index, TValue \* value) const

{

if (index >= 0 && index < count)

{

\*value = entries[index].Value();

return true;

}

return false;

}

bool Remove(const TKey& key)

{

int i, last;

uint targetBucket;

if(FindEntryWithKey(key, &i, &last, &targetBucket))

{

RemoveAt(i, last, targetBucket);

return true;

}

return false;

}

EntryIterator<const BaseDictionary> GetIterator() const

{

return EntryIterator<const BaseDictionary>(\*this);

}

EntryIterator<BaseDictionary> GetIterator()

{

return EntryIterator<BaseDictionary>(\*this);

}

BucketEntryIterator<BaseDictionary> GetIteratorWithRemovalSupport()

{

return BucketEntryIterator<BaseDictionary>(\*this);

}

template<class Fn>

bool AnyValue(Fn fn) const

{

for (uint i = 0; i < bucketCount; i++)

{

if(buckets[i] != -1)

{

for (int currentIndex = buckets[i] ; currentIndex != -1 ; currentIndex = entries[currentIndex].next)

{

if (fn(entries[currentIndex].Value()))

{

return true;

}

}

}

}

return false;

}

template<class Fn>

void EachValue(Fn fn) const

{

for (uint i = 0; i < bucketCount; i++)

{

if(buckets[i] != -1)

{

for (int currentIndex = buckets[i] ; currentIndex != -1 ; currentIndex = entries[currentIndex].next)

{

fn(entries[currentIndex].Value());

}

}

}

}

template<class Fn>

void MapReference(Fn fn)

{

MapUntilReference([fn](TKey const& key, TValue& value)

{

fn(key, value);

return false;

});

}

template<class Fn>

bool MapUntilReference(Fn fn) const

{

return MapEntryUntil([fn](EntryType &entry) -> bool

{

return fn(entry.Key(), entry.Value());

});

}

template<class Fn>

void MapAddress(Fn fn) const

{

MapUntilAddress([fn](TKey const& key, TValue \* value) -> bool

{

fn(key, value);

return false;

});

}

template<class Fn>

bool MapUntilAddress(Fn fn) const

{

return MapEntryUntil([fn](EntryType &entry) -> bool

{

return fn(entry.Key(), &entry.Value());

});

}

template<class Fn>

void Map(Fn fn) const

{

MapUntil([fn](TKey const& key, TValue const& value) -> bool

{

fn(key, value);

return false;

});

}

template<class Fn>

bool MapUntil(Fn fn) const

{

return MapEntryUntil([fn](EntryType const& entry) -> bool

{

return fn(entry.Key(), entry.Value());

});

}

template<class Fn>

void MapAndRemoveIf(Fn fn)

{

for (uint i = 0; i < bucketCount; i++)

{

if (buckets[i] != -1)

{

for (int currentIndex = buckets[i], lastIndex = -1; currentIndex != -1;)

{

// If the predicate says we should remove this item

if (fn(entries[currentIndex]) == true)

{

const int nextIndex = entries[currentIndex].next;

RemoveAt(currentIndex, lastIndex, i);

currentIndex = nextIndex;

}

else

{

lastIndex = currentIndex;

currentIndex = entries[currentIndex].next;

}

}

}

}

}

template <class Fn>

bool RemoveIf(TKey const& key, Fn fn)

{

return RemoveIfWithKey<TKey>(key, fn);

}

template <typename LookupType, class Fn>

bool RemoveIfWithKey(LookupType const& lookupKey, Fn fn)

{

int i, last;

uint targetBucket;

if (FindEntryWithKey<LookupType>(lookupKey, &i, &last, &targetBucket))

{

if (fn(entries[i].Key(), entries[i].Value()))

{

RemoveAt(i, last, targetBucket);

return true;

}

}

return false;

}

// Returns whether the dictionary was resized or not

bool EnsureCapacity()

{

if (freeCount == 0 && count == size)

{

Resize();

return true;

}

return false;

}

int GetNextIndex()

{

if (freeCount != 0)

{

Assert(freeCount > 0);

Assert(freeList >= 0);

Assert(freeList < count);

return freeList;

}

return count;

}

int GetLastIndex()

{

return count - 1;

}

BaseDictionary \*Clone()

{

return AllocatorNew(AllocatorType, alloc, BaseDictionary, \*this);

}

void Copy(const BaseDictionary \*const other)

{

DoCopy(other);

}

void LockResize()

{

\_\_super::LockResize();

}

void UnlockResize()

{

\_\_super::UnlockResize();

}

protected:

template<class T>

void DoCopy(const T \*const other)

{

Assert(size == 0);

Assert(bucketCount == 0);

Assert(!buckets);

Assert(!entries);

Assert(count == 0);

Assert(freeCount == 0);

#if PROFILE\_DICTIONARY

Assert(!stats);

#endif

if(other->Count() == 0)

{

return;

}

Assert(other->bucketCount != 0);

Assert(other->size != 0);

buckets = AllocateBuckets(other->bucketCount);

Assert(buckets); // no-throw allocators are currently not supported

try

{

entries = AllocateEntries(other->size, false /\* zeroAllocate \*/);

Assert(entries); // no-throw allocators are currently not supported

}

catch(...)

{

DeleteBuckets(buckets, other->bucketCount);

buckets = nullptr;

throw;

}

size = other->size;

bucketCount = other->bucketCount;

count = other->count;

freeList = other->freeList;

freeCount = other->freeCount;

size\_t copySize = bucketCount \* sizeof(buckets[0]);

js\_memcpy\_s(buckets, copySize, other->buckets, copySize);

copySize = size \* sizeof(entries[0]);

js\_memcpy\_s(entries, copySize, other->entries, copySize);

#if PROFILE\_DICTIONARY

stats = DictionaryStats::Create(typeid(this).name(), size);

#endif

}

protected:

template<class Fn>

bool MapEntryUntil(Fn fn) const

{

for (uint i = 0; i < bucketCount; i++)

{

if(buckets[i] != -1)

{

for (int currentIndex = buckets[i] ; currentIndex != -1 ; currentIndex = entries[currentIndex].next)

{

if (fn(entries[currentIndex]))

{

return true; // fn condition succeeds

}

}

}

}

return false;

}

private:

template <typename TLookup>

static hash\_t GetHashCodeWithKey(const TLookup& key)

{

// set last bit to 1 to avoid false positive to make hash appears to be an valid recycler address.

// In the same line, 0 should be use to indicate a non-existing entry.

return TAGHASH(Comparer<TLookup>::GetHashCode(key));

}

static hash\_t GetHashCode(const TKey& key)

{

return GetHashCodeWithKey<TKey>(key);

}

static uint GetBucket(hash\_t hashCode, int bucketCount)

{

return SizePolicy::GetBucket(UNTAGHASH(hashCode), bucketCount);

}

uint GetBucket(uint hashCode) const

{

return GetBucket(hashCode, this->bucketCount);

}

static bool IsFreeEntry(const EntryType &entry)

{

// A free entry's next index will be (-2 - nextIndex), such that it is always <= -2, for fast entry iteration

// allowing for skipping over free entries. -1 is reserved for the end-of-chain marker for a used entry.

return entry.next <= -2;

}

void SetNextFreeEntryIndex(EntryType &freeEntry, const int nextFreeEntryIndex)

{

Assert(!IsFreeEntry(freeEntry));

Assert(nextFreeEntryIndex >= -1);

Assert(nextFreeEntryIndex < count);

// The last entry in the free list chain will have a next of -2 to indicate that it is a free entry. The end of the

// free list chain is identified using freeCount.

freeEntry.next = nextFreeEntryIndex >= 0 ? -2 - nextFreeEntryIndex : -2;

}

static int GetNextFreeEntryIndex(const EntryType &freeEntry)

{

Assert(IsFreeEntry(freeEntry));

return -2 - freeEntry.next;

}

template <typename LookupType>

\_\_inline int FindEntryWithKey(const LookupType& key) const

{

#if PROFILE\_DICTIONARY

uint depth = 0;

#endif

int \* localBuckets = buckets;

if (localBuckets != nullptr)

{

hash\_t hashCode = GetHashCodeWithKey<LookupType>(key);

uint targetBucket = this->GetBucket(hashCode);

EntryType \* localEntries = entries;

for (int i = localBuckets[targetBucket]; i >= 0; i = localEntries[i].next)

{

if (localEntries[i].KeyEquals<Comparer<TKey>>(key, hashCode))

{

#if PROFILE\_DICTIONARY

if (stats)

stats->Lookup(depth);

#endif

return i;

}

#if PROFILE\_DICTIONARY

depth += 1;

#endif

}

}

#if PROFILE\_DICTIONARY

if (stats)

stats->Lookup(depth);

#endif

return -1;

}

inline int FindEntry(const TKey& key) const

{

return FindEntryWithKey<TKey>(key);

}

template <typename LookupType>

\_\_inline bool FindEntryWithKey(const LookupType& key, int \*const i, int \*const last, uint \*const targetBucket)

{

#if PROFILE\_DICTIONARY

uint depth = 0;

#endif

int \* localBuckets = buckets;

if (localBuckets != nullptr)

{

uint hashCode = GetHashCodeWithKey<LookupType>(key);

\*targetBucket = this->GetBucket(hashCode);

\*last = -1;

EntryType \* localEntries = entries;

for (\*i = localBuckets[\*targetBucket]; \*i >= 0; \*last = \*i, \*i = localEntries[\*i].next)

{

if (localEntries[\*i].KeyEquals<Comparer<TKey>>(key, hashCode))

{

#if PROFILE\_DICTIONARY

if (stats)

stats->Lookup(depth);

#endif

return true;

}

#if PROFILE\_DICTIONARY

depth += 1;

#endif

}

}

#if PROFILE\_DICTIONARY

if (stats)

stats->Lookup(depth);

#endif

return false;

}

void Initialize(int capacity)

{

// minimum capacity is 4

int initSize = max(capacity, 4);

uint initBucketCount = SizePolicy::GetBucketSize(initSize);

AssertMsg(initBucketCount > 0, "Size returned by policy should be greater than 0");

Allocate(&buckets, &entries, initBucketCount, initSize);

// Allocation can throw - assign the size only after allocation has succeeded.

this->bucketCount = initBucketCount;

this->size = initSize;

Assert(this->freeCount == 0);

#if PROFILE\_DICTIONARY

stats = DictionaryStats::Create(typeid(this).name(), size);

#endif

}

template <InsertOperations op>

int Insert(TKey key, TValue value)

{

int \* localBuckets = buckets;

if (localBuckets == nullptr)

{

Initialize(0);

localBuckets = buckets;

}

#if DBG || PROFILE\_DICTIONARY

// Always search and verify

const bool needSearch = true;

#else

const bool needSearch = (op != Insert\_Add);

#endif

hash\_t hashCode = GetHashCode(key);

uint targetBucket = this->GetBucket(hashCode);

if (needSearch)

{

#if PROFILE\_DICTIONARY

uint depth = 0;

#endif

EntryType \* localEntries = entries;

for (int i = localBuckets[targetBucket]; i >= 0; i = localEntries[i].next)

{

if (localEntries[i].KeyEquals<Comparer<TKey>>(key, hashCode))

{

#if PROFILE\_DICTIONARY

if (stats)

stats->Lookup(depth);

#endif

Assert(op != Insert\_Add);

if (op == Insert\_Item)

{

localEntries[i].SetValue(value);

return i;

}

return -1;

}

#if PROFILE\_DICTIONARY

depth += 1;

#endif

}

#if PROFILE\_DICTIONARY

if (stats)

stats->Lookup(depth);

#endif

}

// Ideally we'd do cleanup only if weak references have been collected since the last resize

// but that would require us to use an additional field to store the last recycler cleanup id

// that we saw

// We can add that optimization later if we have to.

if (EntryType::SupportsCleanup() && freeCount == 0 && count == size)

{

this->MapAndRemoveIf([](EntryType& entry)

{

return EntryType::NeedsCleanup(entry);

});

}

int index;

if (freeCount != 0)

{

Assert(freeCount > 0);

Assert(freeList >= 0);

Assert(freeList < count);

index = freeList;

freeCount--;

if(freeCount != 0)

{

freeList = GetNextFreeEntryIndex(entries[index]);

}

}

else

{

// If there's nothing free, then in general, we set index to count, and increment count

// If we resize, we also need to recalculate the target

// However, if cleanup is supported, then before resize, we should try and clean up and see

// if something got freed, and if it did, reuse that index

if (count == size)

{

Resize();

targetBucket = this->GetBucket(hashCode);

index = count;

count++;

}

else

{

index = count;

count++;

}

Assert(count <= size);

Assert(index < size);

}

entries[index].Set(key, value, hashCode);

entries[index].next = buckets[targetBucket];

buckets[targetBucket] = index;

#if PROFILE\_DICTIONARY

int profileIndex = index;

uint depth = 1; // need to recalculate depth in case there was a resize (also 1-based for stats->Insert)

while(entries[profileIndex].next != -1)

{

profileIndex = entries[profileIndex].next;

++depth;

}

if (stats)

stats->Insert(depth);

#endif

return index;

}

void Resize()

{

AutoDoResize autoDoResize(\*this);

int newSize = SizePolicy::GetNextSize(count);

uint newBucketCount = SizePolicy::GetBucketSize(newSize);

\_\_analysis\_assume(newSize > count);

int\* newBuckets = nullptr;

EntryType\* newEntries = nullptr;

if (newBucketCount == bucketCount)

{

// no need to rehash

newEntries = AllocateEntries(newSize);

js\_memcpy\_s(newEntries, sizeof(EntryType) \* newSize, entries, sizeof(EntryType) \* count);

DeleteEntries(entries, size);

this->entries = newEntries;

this->size = newSize;

return;

}

Allocate(&newBuckets, &newEntries, newBucketCount, newSize);

js\_memcpy\_s(newEntries, sizeof(EntryType) \* newSize, entries, sizeof(EntryType) \* count);

// When TAllocator is of type Recycler, it is possible that the Allocate above causes a collection, which

// in turn can cause entries in the dictionary to be removed - i.e. the dictionary contains weak references

// that remove themselves when no longer valid. This means the free list might not be empty anymore.

for (int i = 0; i < count; i++)

{

\_\_analysis\_assume(i < newSize);

if (!IsFreeEntry(newEntries[i]))

{

uint hashCode = newEntries[i].GetHashCode<Comparer<TKey>>();

int bucket = GetBucket(hashCode, newBucketCount);

newEntries[i].next = newBuckets[bucket];

newBuckets[bucket] = i;

}

}

DeleteBuckets(buckets, bucketCount);

DeleteEntries(entries, size);

#if PROFILE\_DICTIONARY

if (stats)

stats->Resize(newSize, /\*emptyBuckets=\*/ newSize - size);

#endif

buckets = newBuckets;

bucketCount = newBucketCount;

size = newSize;

entries = newEntries;

}

\_\_ecount(bucketCount) int \*AllocateBuckets(const uint bucketCount)

{

return

AllocateArray<AllocatorType, int, false>(

TRACK\_ALLOC\_INFO(alloc, int, AllocatorType, 0, bucketCount),

TypeAllocatorFunc<AllocatorType, int>::GetAllocFunc(),

bucketCount);

}

\_\_ecount(size) EntryType \* AllocateEntries(int size, const bool zeroAllocate = true)

{

// Note that the choice of leaf/non-leaf node is decided for the EntryType on the basis of TValue. By default, if

// TValue is a pointer, a non-leaf allocation is done. This behavior can be overridden by specializing

// TypeAllocatorFunc for TValue.

return

AllocateArray<AllocatorType, EntryType, false>(

TRACK\_ALLOC\_INFO(alloc, EntryType, AllocatorType, 0, size),

zeroAllocate ? EntryAllocatorFuncType::GetAllocZeroFunc() : EntryAllocatorFuncType::GetAllocFunc(),

size);

}

void DeleteBuckets(\_\_in\_ecount(bucketCount) int \*const buckets, const uint bucketCount)

{

Assert(buckets);

Assert(bucketCount != 0);

AllocatorFree(alloc, (TypeAllocatorFunc<AllocatorType, int>::GetFreeFunc()), buckets, bucketCount \* sizeof(int));

}

void DeleteEntries(\_\_in\_ecount(size) EntryType \*const entries, const int size)

{

Assert(entries);

Assert(size != 0);

AllocatorFree(alloc, EntryAllocatorFuncType::GetFreeFunc(), entries, size \* sizeof(EntryType));

}

void Allocate(\_\_deref\_out\_ecount(bucketCount) int\*\* ppBuckets, \_\_deref\_out\_ecount(size) EntryType\*\* ppEntries, uint bucketCount, int size)

{

int \*const buckets = AllocateBuckets(bucketCount);

Assert(buckets); // no-throw allocators are currently not supported

EntryType \*entries;

try

{

entries = AllocateEntries(size);

Assert(entries); // no-throw allocators are currently not supported

}

catch(...)

{

DeleteBuckets(buckets, bucketCount);

throw;

}

memset(buckets, -1, bucketCount \* sizeof(buckets[0]));

\*ppBuckets = buckets;

\*ppEntries = entries;

}

\_\_inline void RemoveAt(const int i, const int last, const uint targetBucket)

{

if (last < 0)

{

buckets[targetBucket] = entries[i].next;

}

else

{

entries[last].next = entries[i].next;

}

entries[i].Clear();

SetNextFreeEntryIndex(entries[i], freeCount == 0 ? -1 : freeList);

freeList = i;

freeCount++;

#if PROFILE\_DICTIONARY

if (stats)

stats->Remove(buckets[targetBucket] == -1);

#endif

}

#if DBG\_DUMP

public:

void Dump()

{

printf("Dumping Dictionary\n");

printf("-------------------\n");

for (uint i = 0; i < bucketCount; i++)

{

printf("Bucket value: %d\n", buckets[i]);

for (int j = buckets[i]; j >= 0; j = entries[j].next)

{

printf("%d => %d Next: %d\n", entries[j].Key(), entries[j].Value(), entries[j].next);

}

}

}

#endif

protected:

template<class TDictionary, class Leaf>

class IteratorBase abstract

{

protected:

EntryType \*const entries;

int entryIndex;

#if DBG

protected:

TDictionary &dictionary;

private:

int usedEntryCount;

#endif

protected:

IteratorBase(TDictionary &dictionary, const int entryIndex)

: entries(dictionary.entries),

entryIndex(entryIndex)

#if DBG

,

dictionary(dictionary),

usedEntryCount(dictionary.Count())

#endif

{

}

protected:

void OnEntryRemoved()

{

DebugOnly(--usedEntryCount);

}

private:

bool IsValid\_Virtual() const

{

return static\_cast<const Leaf \*>(this)->IsValid();

}

protected:

bool IsValid() const

{

Assert(dictionary.entries == entries);

Assert(dictionary.Count() == usedEntryCount);

return true;

}

public:

EntryType &Current() const

{

Assert(IsValid\_Virtual());

Assert(!IsFreeEntry(entries[entryIndex]));

return entries[entryIndex];

}

TKey CurrentKey() const

{

return Current().Key();

}

const TValue &CurrentValue() const

{

return Current().Value();

}

TValue &CurrentValueReference() const

{

return Current().Value();

}

void SetCurrentValue(const TValue &value) const

{

#if DBG

// For BaseHashSet, save the key before changing the value to verify that the key does not change

const TKey previousKey = CurrentKey();

const hash\_t previousHashCode = GetHashCode(previousKey);

#endif

Current().SetValue(value);

Assert(Current().KeyEquals<Comparer<TKey>>(previousKey, previousHashCode));

}

};

public:

template<class TDictionary>

class EntryIterator sealed : public IteratorBase<TDictionary, EntryIterator<TDictionary>>

{

private:

typedef IteratorBase<TDictionary, EntryIterator<TDictionary>> Base;

private:

const int entryCount;

public:

EntryIterator(TDictionary &dictionary) : Base(dictionary, 0), entryCount(dictionary.count)

{

if(IsValid() && IsFreeEntry(entries[entryIndex]))

{

MoveNext();

}

}

public:

bool IsValid() const

{

Assert(dictionary.count == entryCount);

Assert(entryIndex >= 0);

Assert(entryIndex <= entryCount);

return Base::IsValid() && entryIndex < entryCount;

}

public:

void MoveNext()

{

Assert(IsValid());

do

{

++entryIndex;

} while(IsValid() && IsFreeEntry(entries[entryIndex]));

}

};

public:

template<class TDictionary>

class BucketEntryIterator sealed : public IteratorBase<TDictionary, BucketEntryIterator<TDictionary>>

{

private:

typedef IteratorBase<TDictionary, BucketEntryIterator<TDictionary>> Base;

private:

TDictionary &dictionary;

int \*const buckets;

const uint bucketCount;

uint bucketIndex;

int previousEntryIndexInBucket;

int indexOfEntryAfterRemovedEntry;

public:

BucketEntryIterator(TDictionary &dictionary)

: Base(dictionary, -1),

dictionary(dictionary),

buckets(dictionary.buckets),

bucketCount(dictionary.bucketCount),

bucketIndex(0u - 1)

#if DBG

,

previousEntryIndexInBucket(-2),

indexOfEntryAfterRemovedEntry(-2)

#endif

{

if(dictionary.Count() != 0)

{

MoveNextBucket();

}

}

public:

bool IsValid() const

{

Assert(dictionary.buckets == buckets);

Assert(dictionary.bucketCount == bucketCount);

Assert(entryIndex >= -1);

Assert(entryIndex < dictionary.count);

Assert(bucketIndex == 0u - 1 || bucketIndex <= bucketCount);

Assert(previousEntryIndexInBucket >= -2);

Assert(previousEntryIndexInBucket < dictionary.count);

Assert(indexOfEntryAfterRemovedEntry >= -2);

Assert(indexOfEntryAfterRemovedEntry < dictionary.count);

return Base::IsValid() && entryIndex >= 0;

}

public:

void MoveNext()

{

if(IsValid())

{

previousEntryIndexInBucket = entryIndex;

entryIndex = Current().next;

}

else

{

Assert(indexOfEntryAfterRemovedEntry >= -1);

entryIndex = indexOfEntryAfterRemovedEntry;

}

if(!IsValid())

{

MoveNextBucket();

}

}

private:

void MoveNextBucket()

{

Assert(!IsValid());

while(++bucketIndex < bucketCount)

{

entryIndex = buckets[bucketIndex];

if(IsValid())

{

previousEntryIndexInBucket = -1;

break;

}

}

}

public:

void RemoveCurrent()

{

Assert(previousEntryIndexInBucket >= -1);

indexOfEntryAfterRemovedEntry = Current().next;

dictionary.RemoveAt(entryIndex, previousEntryIndexInBucket, bucketIndex);

OnEntryRemoved();

entryIndex = -1;

}

};

template<class TDictionary, class Leaf> friend class IteratorBase;

template<class TDictionary> friend class EntryIterator;

template<class TDictionary> friend class BucketEntryIterator;

PREVENT\_ASSIGN(BaseDictionary);

};

template <class TKey, class TValue> class SimpleHashedEntry;

template <

class TElement,

class TAllocator,

class SizePolicy = PowerOf2SizePolicy,

class TKey = TElement,

template <typename ValueOrKey> class Comparer = DefaultComparer,

template <typename, typename> class Entry = SimpleHashedEntry,

typename Lock = NoResizeLock

>

class BaseHashSet : protected BaseDictionary<TKey, TElement, TAllocator, SizePolicy, Comparer, Entry, Lock>

{

typedef BaseDictionary<TKey, TElement, TAllocator, SizePolicy, Comparer, Entry, Lock> Base;

typedef Entry<TKey, TElement> EntryType;

friend struct JsDiag::RemoteDictionary<BaseHashSet<TElement, TAllocator, SizePolicy, TKey, Comparer, Entry, Lock>>;

public:

BaseHashSet(AllocatorType \* allocator, int capacity = 0) : BaseDictionary(allocator, capacity) {}

using Base::GetAllocator;

int Count() const

{

return \_\_super::Count();

}

int Add(TElement const& element)

{

return \_\_super::Add(ValueToKey<TKey, TElement>::ToKey(element), element);

}

// Add only if there isn't an existing element

int AddNew(TElement const& element)

{

return \_\_super::AddNew(ValueToKey<TKey, TElement>::ToKey(element), element);

}

int Item(TElement const& element)

{

return \_\_super::Item(ValueToKey<TKey, TElement>::ToKey(element), element);

}

void Clear()

{

\_\_super::Clear();

}

using Base::Reset;

TElement const& Lookup(TKey const& key)

{

// Use a static to pass the null default value, since the

// default value may get returned out of the current scope by ref.

static const TElement nullElement = nullptr;

return \_\_super::Lookup(key, nullElement);

}

template <typename KeyType>

TElement const& LookupWithKey(KeyType const& key)

{

static const TElement nullElement = nullptr;

return \_\_super::LookupWithKey(key, nullElement);

}

bool Contains(TElement const& element) const

{

return ContainsKey(ValueToKey<TKey, TElement>::ToKey(element));

}

using Base::ContainsKey;

using Base::TryGetValue;

using Base::TryGetReference;

bool RemoveKey(const TKey &key)

{

return Base::Remove(key);

}

bool Remove(TElement const& element)

{

return \_\_super::Remove(ValueToKey<TKey, TElement>::ToKey(element));

}

EntryIterator<const BaseHashSet> GetIterator() const

{

return EntryIterator<const BaseHashSet>(\*this);

}

EntryIterator<BaseHashSet> GetIterator()

{

return EntryIterator<BaseHashSet>(\*this);

}

BucketEntryIterator<BaseHashSet> GetIteratorWithRemovalSupport()

{

return BucketEntryIterator<BaseHashSet>(\*this);

}

template<class Fn>

void Map(Fn fn)

{

MapUntil([fn](TElement const& value) -> bool

{

fn(value);

return false;

});

}

template<class Fn>

void MapAndRemoveIf(Fn fn)

{

\_\_super::MapAndRemoveIf([fn](EntryType const& entry) -> bool

{

return fn(entry.Value());

});

}

template<class Fn>

bool MapUntil(Fn fn)

{

return \_\_super::MapEntryUntil([fn](EntryType const& entry) -> bool

{

return fn(entry.Value());

});

}

bool EnsureCapacity()

{

return \_\_super::EnsureCapacity();

}

int GetNextIndex()

{

return \_\_super::GetNextIndex();

}

int GetLastIndex()

{

return \_\_super::GetLastIndex();

}

using Base::GetValueAt;

bool TryGetValueAt(int index, TElement \* value) const

{

return \_\_super::TryGetValueAt(index, value);

}

BaseHashSet \*Clone()

{

return AllocatorNew(AllocatorType, alloc, BaseHashSet, \*this);

}

void Copy(const BaseHashSet \*const other)

{

DoCopy(other);

}

void LockResize()

{

\_\_super::LockResize();

}

void UnlockResize()

{

\_\_super::UnlockResize();

}

public:

using Base::EntryIterator;

using Base::BucketEntryIterator;

friend class Base;

// The following syntax works in BaseDictionary (where the classes are defined), but not in derived

// classes such as BaseHashSet

// template<class TDictionary, class Leaf> friend class IteratorBase;

// template<class TDictionary> friend class EntryIterator;

// template<class TDictionary> friend class BucketEntryIterator;

friend class Base::IteratorBase<const BaseHashSet, EntryIterator<const BaseHashSet>>;

friend class Base::IteratorBase<const BaseHashSet, BucketEntryIterator<const BaseHashSet>>;

friend class Base::IteratorBase<BaseHashSet, EntryIterator<BaseHashSet>>;

friend class Base::IteratorBase<BaseHashSet, BucketEntryIterator<BaseHashSet>>;

friend class EntryIterator<const BaseHashSet>;

friend class EntryIterator<BaseHashSet>;

friend class BucketEntryIterator<const BaseHashSet>;

friend class BucketEntryIterator<BaseHashSet>;

PREVENT\_ASSIGN(BaseHashSet);

};

template <

class TKey,

class TValue,

class TAllocator,

class SizePolicy = PowerOf2SizePolicy,

template <typename ValueOrKey> class Comparer = DefaultComparer,

template <typename K, typename V> class Entry = SimpleDictionaryEntry,

class LockPolicy = Js::DefaultListLockPolicy, // Controls lock policy for read/map/write/add/remove items

class SyncObject = CriticalSection

>

class SynchronizedDictionary: protected BaseDictionary<TKey, TValue, TAllocator, SizePolicy, Comparer, Entry>

{

private:

SyncObject\* syncObj;

public:

typedef TKey KeyType;

typedef TValue ValueType;

typedef BaseDictionary<TKey, TValue, TAllocator, SizePolicy, Comparer, Entry>::EntryType EntryType;

typedef SynchronizedDictionary<TKey, TValue, TAllocator, SizePolicy, Comparer, Entry, LockPolicy, SyncObject> DictionaryType;

private:

friend class Js::RemoteDictionary<DictionaryType>;

public:

SynchronizedDictionary(AllocatorType \* allocator, int capacity, SyncObject\* syncObject):

BaseDictionary(allocator, capacity),

syncObj(syncObject)

{}

#ifdef DBG

void Dump()

{

LockPolicy::ReadLock autoLock(syncObj);

\_\_super::Dump();

}

#endif

TAllocator \*GetAllocator() const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::GetAllocator();

}

inline int Count() const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::Count();

}

inline int Capacity() const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::Capacity();

}

TValue Item(const TKey& key)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::Item(key);

}

bool IsInAdd()

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::IsInAdd();

}

int Add(const TKey& key, const TValue& value)

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::Add(key, value);

}

int AddNew(const TKey& key, const TValue& value)

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::AddNew(key, value);

}

int Item(const TKey& key, const TValue& value)

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::Item(key, value);

}

bool Contains(KeyValuePair<TKey, TValue> keyValuePair)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::Contains(keyValuePair);

}

bool Remove(KeyValuePair<TKey, TValue> keyValuePair)

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::Remove(keyValuePair);

}

void Clear()

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::Clear();

}

void Reset()

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::Reset();

}

bool ContainsKey(const TKey& key)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::ContainsKey(key);

}

template <typename TLookup>

inline const TValue& LookupWithKey(const TLookup& key, const TValue& defaultValue)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::LookupWithKey(key, defaultValue);

}

inline const TValue& Lookup(const TKey& key, const TValue& defaultValue)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::Lookup(key, defaultValue);

}

template <typename TLookup>

bool TryGetValue(const TLookup& key, TValue\* value)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::TryGetValue(key, value);

}

bool TryGetValueAndRemove(const TKey& key, TValue\* value)

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::TryGetValueAndRemove(key, value);

}

template <typename TLookup>

\_\_inline bool TryGetReference(const TLookup& key, TValue\*\* value)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::TryGetReference(key, value);

}

template <typename TLookup>

\_\_inline bool TryGetReference(const TLookup& key, TValue\*\* value, int\* index)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::TryGetReference(key, value, index);

}

const TValue& GetValueAt(const int& index) const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::GetValueAt(index);

}

TValue\* GetReferenceAt(const int& index)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::GetReferenceAt(index);

}

TKey const& GetKeyAt(const int& index)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::GetKeyAt(index);

}

bool Remove(const TKey& key)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::Remove(key);

}

template<class Fn>

void MapReference(Fn fn)

{

// TODO: Verify that Map doesn't actually modify the list

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::MapReference(fn);

}

template<class Fn>

bool MapUntilReference(Fn fn) const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::MapUntilReference(fn);

}

template<class Fn>

void MapAddress(Fn fn) const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::MapAddress(fn);

}

template<class Fn>

bool MapUntilAddress(Fn fn) const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::MapUntilAddress(fn);

}

template<class Fn>

void Map(Fn fn) const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::Map(fn);

}

template<class Fn>

bool MapUntil(Fn fn) const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::MapUntil(fn);

}

template<class Fn>

void MapAndRemoveIf(Fn fn)

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::MapAndRemoveIf(fn);

}

PREVENT\_COPY(SynchronizedDictionary);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

#include "DataStructures\BigInt.h"

#include "common\NumberUtilitiesBase.h"

#include "common\NumberUtilities.h"

namespace Js

{

BigInt & BigInt::operator= (BigInt &bi)

{

AssertMsg(false, "can't assign BigInts");

return \*this;

}

#if DBG

void BigInt::AssertValid(bool fCheckVal)

{

Assert(m\_cluMax >= kcluMaxInit);

Assert(m\_prglu != 0);

Assert(m\_clu >= 0 && m\_clu <= m\_cluMax);

Assert(!fCheckVal || 0 == m\_clu || 0 != m\_prglu[m\_clu - 1]);

Assert((m\_prglu == m\_rgluInit) == (m\_cluMax == kcluMaxInit));

}

#endif

BigInt::BigInt(void)

{

m\_cluMax = kcluMaxInit;

m\_clu = 0;

m\_prglu = m\_rgluInit;

AssertBi(this);

}

BigInt::~BigInt(void)

{

if (m\_prglu != m\_rgluInit)

free(m\_prglu);

}

long BigInt::Clu(void)

{

return m\_clu;

}

ulong BigInt::Lu(long ilu)

{

AssertBi(this);

Assert(ilu < m\_clu);

return m\_prglu[ilu];

}

bool BigInt::FResize(long clu)

{

AssertBiNoVal(this);

ulong \*prglu;

if (clu <= m\_cluMax)

return true;

clu += clu;

if (m\_prglu == m\_rgluInit)

{

if ((INT\_MAX / sizeof(ulong) < clu) || (NULL == (prglu = (ulong \*)malloc(clu \* sizeof(ulong)))))

return false;

if (0 < m\_clu)

js\_memcpy\_s(prglu, clu \* sizeof(ulong), m\_prglu, m\_clu \* sizeof(ulong));

}

else if (NULL == (prglu = (ulong \*)realloc(m\_prglu, clu \* sizeof(ulong))))

return false;

m\_prglu = prglu;

m\_cluMax = clu;

AssertBiNoVal(this);

return true;

}

bool BigInt::FInitFromRglu(ulong \*prglu, long clu)

{

AssertBi(this);

Assert(clu >= 0);

Assert(prglu != 0);

if (clu > m\_cluMax && !FResize(clu))

return false;

m\_clu = clu;

if (clu > 0)

js\_memcpy\_s(m\_prglu, m\_clu \* sizeof(ulong), prglu, clu \* sizeof(ulong));

AssertBi(this);

return true;

}

bool BigInt::FInitFromBigint(BigInt \*pbiSrc)

{

AssertBi(this);

AssertBi(pbiSrc);

Assert(this != pbiSrc);

return FInitFromRglu(pbiSrc->m\_prglu, pbiSrc->m\_clu);

}

template <typename EncodedChar>

bool BigInt::FInitFromDigits(const EncodedChar \*prgch, long cch, long \*pcchDig)

{

AssertBi(this);

Assert(cch >= 0);

Assert(prgch != 0);

Assert(pcchDig != 0);

ulong luAdd;

ulong luMul;

long clu = (cch + 8) / 9;

const EncodedChar \*pchLim = prgch + cch;

if (clu > m\_cluMax && !FResize(clu))

return false;

m\_clu = 0;

luAdd = 0;

luMul = 1;

for (\*pcchDig = cch; prgch < pchLim; prgch++)

{

if (\*prgch == '.')

{

(\*pcchDig)--;

continue;

}

Assert(NumberUtilities::IsDigit(\*prgch));

if (luMul == 1000000000)

{

AssertVerify(FMulAdd(luMul, luAdd));

luMul = 1;

luAdd = 0;

}

luMul \*= 10;

luAdd = luAdd \* 10 + \*prgch - '0';

}

Assert(1 < luMul);

AssertVerify(FMulAdd(luMul, luAdd));

AssertBi(this);

return true;

}

bool BigInt::FMulAdd(ulong luMul, ulong luAdd)

{

AssertBi(this);

Assert(luMul != 0);

ulong luT;

ulong \*plu = m\_prglu;

ulong \*pluLim = plu + m\_clu;

for (; plu < pluLim; plu++)

{

\*plu = NumberUtilities::MulLu(\*plu, luMul, &luT);

if (luAdd)

luT += NumberUtilities::AddLu(plu, luAdd);

luAdd = luT;

}

if (0 == luAdd)

goto LDone;

if (m\_clu >= m\_cluMax && !FResize(m\_clu + 1))

return false;

m\_prglu[m\_clu++] = luAdd;

LDone:

AssertBi(this);

return true;

}

bool BigInt::FMulPow5(long c5)

{

AssertBi(this);

Assert(c5 >= 0);

const ulong k5to13 = 1220703125;

long clu = (c5 + 12) / 13;

ulong luT;

if (0 == m\_clu || 0 == c5)

return true;

if (m\_clu + clu > m\_cluMax && !FResize(m\_clu + clu))

return false;

for (; c5 >= 13; c5 -= 13)

AssertVerify(FMulAdd(k5to13, 0));

if (c5 > 0)

{

for (luT = 5; --c5 > 0; )

luT \*= 5;

AssertVerify(FMulAdd(luT, 0));

}

AssertBi(this);

return true;

}

bool BigInt::FShiftLeft(long cbit)

{

AssertBi(this);

Assert(cbit >= 0);

long ilu;

long clu;

ulong luExtra;

if (0 == cbit || 0 == m\_clu)

return true;

clu = cbit >> 5;

cbit &= 0x001F;

if (cbit > 0)

{

ilu = m\_clu - 1;

luExtra = m\_prglu[ilu] >> (32 - cbit);

for (; ; ilu--)

{

m\_prglu[ilu] <<= cbit;

if (0 == ilu)

break;

m\_prglu[ilu] |= m\_prglu[ilu - 1] >> (32 - cbit);

}

}

else

luExtra = 0;

if (clu > 0 || 0 != luExtra)

{

// Make sure there's enough room.

ilu = m\_clu + (0 != luExtra) + clu;

if (ilu > m\_cluMax && !FResize(ilu))

return false;

if (clu > 0)

{

// Shift the ulongs.

memmove(m\_prglu + clu, m\_prglu, m\_clu \* sizeof(ulong));

memset(m\_prglu, 0, clu \* sizeof(ulong));

m\_clu += clu;

}

// Throw on the extra one.

if (0 != luExtra)

m\_prglu[m\_clu++] = luExtra;

}

AssertBi(this);

return true;

}

void BigInt::ShiftLusRight(long clu)

{

AssertBi(this);

Assert(clu >= 0);

if (clu >= m\_clu)

{

m\_clu = 0;

AssertBi(this);

return;

}

if (clu > 0)

{

memmove(m\_prglu, m\_prglu + clu, (m\_clu - clu) \* sizeof(ulong));

m\_clu -= clu;

}

AssertBi(this);

}

void BigInt::ShiftRight(long cbit)

{

AssertBi(this);

Assert(cbit >= 0);

long ilu;

long clu = cbit >> 5;

cbit &= 0x001F;

if (clu > 0)

ShiftLusRight(clu);

if (cbit == 0 || m\_clu == 0)

{

AssertBi(this);

return;

}

for (ilu = 0; ; )

{

m\_prglu[ilu] >>= cbit;

if (++ilu >= m\_clu)

{

// Last one.

if (0 == m\_prglu[ilu - 1])

m\_clu--;

break;

}

m\_prglu[ilu - 1] |= m\_prglu[ilu] << (32 - cbit);

}

AssertBi(this);

}

int BigInt::Compare(BigInt \*pbi)

{

AssertBi(this);

AssertBi(pbi);

long ilu;

if (m\_clu > pbi->m\_clu)

return 1;

if (m\_clu < pbi->m\_clu)

return -1;

if (0 == m\_clu)

return 0;

#pragma prefast(suppress:\_\_WARNING\_LOOP\_ONLY\_EXECUTED\_ONCE,"noise")

for (ilu = m\_clu - 1; m\_prglu[ilu] == pbi->m\_prglu[ilu]; ilu--)

{

if (0 == ilu)

return 0;

}

Assert(ilu >= 0 && ilu < m\_clu);

Assert(m\_prglu[ilu] != pbi->m\_prglu[ilu]);

return (m\_prglu[ilu] > pbi->m\_prglu[ilu]) ? 1 : -1;

}

bool BigInt::FAdd(BigInt \*pbi)

{

AssertBi(this);

AssertBi(pbi);

Assert(this != pbi);

long cluMax, cluMin;

long ilu;

int wCarry;

if ((cluMax = m\_clu) < (cluMin = pbi->m\_clu))

{

cluMax = pbi->m\_clu;

cluMin = m\_clu;

if (cluMax > m\_cluMax && !FResize(cluMax + 1))

return false;

}

wCarry = 0;

for (ilu = 0; ilu < cluMin; ilu++)

{

if (0 != wCarry)

wCarry = NumberUtilities::AddLu(&m\_prglu[ilu], wCarry);

wCarry += NumberUtilities::AddLu(&m\_prglu[ilu], pbi->m\_prglu[ilu]);

}

if (m\_clu < pbi->m\_clu)

{

for (; ilu < cluMax; ilu++)

{

m\_prglu[ilu] = pbi->m\_prglu[ilu];

if (0 != wCarry)

wCarry = NumberUtilities::AddLu(&m\_prglu[ilu], wCarry);

}

m\_clu = cluMax;

}

else

{

for (; 0 != wCarry && ilu < cluMax; ilu++)

wCarry = NumberUtilities::AddLu(&m\_prglu[ilu], wCarry);

}

if (0 != wCarry)

{

if (m\_clu >= m\_cluMax && !FResize(m\_clu + 1))

return false;

m\_prglu[m\_clu++] = wCarry;

}

AssertBi(this);

return true;

}

void BigInt::Subtract(BigInt \*pbi)

{

AssertBi(this);

AssertBi(pbi);

Assert(this != pbi);

long ilu;

int wCarry;

ulong luT;

if (m\_clu < pbi->m\_clu)

goto LNegative;

wCarry = 1;

for (ilu = 0; (ilu < pbi->m\_clu) && (ilu < pbi->m\_cluMax); ilu++)

{

Assert(wCarry == 0 || wCarry == 1);

luT = pbi->m\_prglu[ilu];

// NOTE: We should really do:

// wCarry = AddLu(&m\_prglu[ilu], wCarry);

// wCarry += AddLu(&m\_prglu[ilu], ~luT);

// The only case where this is different than

// wCarry = AddLu(&m\_prglu[ilu], ~luT + wCarry);

// is when luT == 0 and 1 == wCarry, in which case we don't

// need to add anything and wCarry should still be 1, so we can

// just skip the operations.

if (0 != luT || 0 == wCarry)

wCarry = NumberUtilities::AddLu(&m\_prglu[ilu], ~luT + wCarry);

}

while ((0 == wCarry) && (ilu < m\_clu) && (ilu < m\_cluMax))

wCarry = NumberUtilities::AddLu(&m\_prglu[ilu], 0xFFFFFFFF);

if (0 == wCarry)

{

LNegative:

// pbi was bigger than this.

AssertMsg(false, "Who's subtracting to negative?");

m\_clu = 0;

}

else if (ilu == m\_clu)

{

// Trim off zeros.

while (--ilu >= 0 && 0 == m\_prglu[ilu])

;

m\_clu = ilu + 1;

}

AssertBi(this);

}

int BigInt::DivRem(BigInt \*pbi)

{

AssertBi(this);

AssertBi(pbi);

Assert(this != pbi);

long ilu, clu;

int wCarry;

int wQuo;

int wT;

ulong luT, luHi, luLo;

clu = pbi->m\_clu;

Assert(m\_clu <= clu);

if ((m\_clu < clu) || (clu <= 0))

return 0;

// Get a lower bound on the quotient.

wQuo = (int)(m\_prglu[clu - 1] / (pbi->m\_prglu[clu - 1] + 1));

Assert(wQuo >= 0 && wQuo <= 9);

// Handle 0 and 1 as special cases.

switch (wQuo)

{

case 0:

break;

case 1:

Subtract(pbi);

break;

default:

luHi = 0;

wCarry = 1;

for (ilu = 0; ilu < clu; ilu++)

{

Assert(wCarry == 0 || wCarry == 1);

// Compute the product.

luLo = NumberUtilities::MulLu(wQuo, pbi->m\_prglu[ilu], &luT);

luHi = luT + NumberUtilities::AddLu(&luLo, luHi);

// Subtract the product. See note in BigInt::Subtract.

if (0 != luLo || 0 == wCarry)

wCarry = NumberUtilities::AddLu(&m\_prglu[ilu], ~luLo + wCarry);

}

Assert(1 == wCarry);

Assert(ilu == clu);

// Trim off zeros.

while (--ilu >= 0 && 0 == m\_prglu[ilu])

;

m\_clu = ilu + 1;

}

if (wQuo < 9 && (wT = Compare(pbi)) >= 0)

{

// Quotient was off too small (by one).

wQuo++;

if (wT == 0)

m\_clu = 0;

else

Subtract(pbi);

}

Assert(Compare(pbi) < 0);

return wQuo;

}

double BigInt::GetDbl(void)

{

double dbl;

ulong luHi, luLo;

ulong lu1, lu2, lu3;

long ilu;

int cbit;

switch (m\_clu)

{

case 0:

return 0;

case 1:

return m\_prglu[0];

case 2:

dbl = m\_prglu[1];

NumberUtilities::LuHiDbl(dbl) += 0x02000000;

return dbl + m\_prglu[0];

}

Assert(3 <= m\_clu);

if (m\_clu > 32)

{

// Result is infinite.

NumberUtilities::LuHiDbl(dbl) = 0x7FF00000;

NumberUtilities::LuLoDbl(dbl) = 0;

return dbl;

}

lu1 = m\_prglu[m\_clu - 1];

lu2 = m\_prglu[m\_clu - 2];

lu3 = m\_prglu[m\_clu - 3];

Assert(0 != lu1);

cbit = 31 - NumberUtilities::CbitZeroLeft(lu1);

if (cbit == 0)

{

luHi = lu2;

luLo = lu3;

}

else

{

luHi = (lu1 << (32 - cbit)) | (lu2 >> cbit);

// Or 1 if there are any remaining nonzero bits in lu3, so we take

// them into account when rounding.

luLo = (lu2 << (32 - cbit)) | (lu3 >> cbit) | (0 != (lu3 << (32 - cbit)));

}

// Set the mantissa bits.

NumberUtilities::LuHiDbl(dbl) = luHi >> 12;

NumberUtilities::LuLoDbl(dbl) = (luHi << 20) | (luLo >> 12);

// Set the exponent field.

NumberUtilities::LuHiDbl(dbl) |= (0x03FF + cbit + (m\_clu - 1) \* 0x0020) << 20;

// Do IEEE rounding.

if (luLo & 0x0800)

{

if ((luLo & 0x07FF) || (NumberUtilities::LuLoDbl(dbl) & 1))

{

if (0 == ++NumberUtilities::LuLoDbl(dbl))

++NumberUtilities::LuHiDbl(dbl);

}

else

{

// If there are any non-zero bits in m\_prglu from 0 to m\_clu - 4, round up.

for (ilu = m\_clu - 4; ilu >= 0; ilu--)

{

if (0 != m\_prglu[ilu])

{

if (0 == ++NumberUtilities::LuLoDbl(dbl))

++NumberUtilities::LuHiDbl(dbl);

break;

}

}

}

}

return dbl;

}

namespace // anonymous

{

template <typename EncodedChar>

void Reference(void)

{

BigInt i;

i.FInitFromDigits<EncodedChar>(NULL, 0, NULL);

}

void Instantiations(void)

{

Reference<wchar\_t>();

Reference<utf8char\_t>();

}

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Big non-negative integer class.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class BigInt

{

private:

// Make this big enough that we rarely have to call malloc.

enum { kcluMaxInit = 30 };

long m\_cluMax;

long m\_clu;

ulong \*m\_prglu;

ulong m\_rgluInit[kcluMaxInit];

inline BigInt & operator= (BigInt &bi);

bool FResize(long clu);

#if DBG

#define AssertBi(pbi) Assert(pbi); (pbi)->AssertValid(true);

#define AssertBiNoVal(pbi) Assert(pbi); (pbi)->AssertValid(false);

inline void AssertValid(bool fCheckVal);

#else //!DBG

#define AssertBi(pbi)

#define AssertBiNoVal(pbi)

#endif //!DBG

public:

BigInt(void);

~BigInt(void);

bool FInitFromRglu(ulong \*prglu, long clu);

bool FInitFromBigint(BigInt \*pbiSrc);

template <typename EncodedChar>

bool FInitFromDigits(const EncodedChar \*prgch, long cch, long \*pcchDec);

bool FMulAdd(ulong luMul, ulong luAdd);

bool FMulPow5(long c5);

bool FShiftLeft(long cbit);

void ShiftLusRight(long clu);

void ShiftRight(long cbit);

int Compare(BigInt \*pbi);

bool FAdd(BigInt \*pbi);

void Subtract(BigInt \*pbi);

int DivRem(BigInt \*pbi);

long Clu(void);

ulong Lu(long ilu);

double GetDbl(void);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

typedef uint BVIndex;

const BVIndex BVInvalidIndex = (uint)-1;

const int MachBits = 8;

#include "UnitBitVector.h"

#include "FixedBitVectorEnumerator.h"

#include "FixedBitVector.h"

#include "SparseBitVector.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

#include "Option.h"

#include "ImmutableList.h"

#include "BufferBuilder.h"

namespace Js

{

#if DBG

void

BufferBuilder::TraceOutput(byte \* buffer, uint32 size) const

{

if (PHASE\_TRACE1(Js::ByteCodeSerializationPhase))

{

Output::Print(L"%08X: %-40s:", this->offset, this->clue);

for (uint i = 0; i < size; i ++)

{

Output::Print(L" %02x", buffer[this->offset + i]);

}

Output::Print(L"\n");

}

}

#endif

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// Buffer builder is used to layout out binary content which contains offsets

// from one part of the content to another.

// It works in two-pass fashion:

// - Pass one fixes the real offset of each element (BufferBuilder) of the

// content.

// - Pass two writes the actual content including any relative offset values.

//----------------------------------------------------------------------------

#pragma once

#if \_M\_IX86

#define serialization\_alignment

#elif \_M\_X64 || defined(\_M\_ARM32\_OR\_ARM64)

#define serialization\_alignment \_\_unaligned

#else

#error Must define alignment capabilities for processor

#endif

struct \_SIMDValue;

typedef \_SIMDValue SIMDValue;

namespace Js

{

// The base buffer builder class

class BufferBuilder

{

protected:

LPCWSTR clue;

BufferBuilder(LPCWSTR clue)

: clue(clue), offset(0xffffffff) { }

public:

uint32 offset;

virtual uint32 FixOffset(uint32 offset) = 0;

virtual void Write(\_\_in\_bcount(bufferSize) byte \* buffer, \_\_in uint32 bufferSize) const = 0;

#if DBG

protected:

void TraceOutput(byte \* buffer, uint32 size) const;

#endif

};

#if VARIABLE\_INT\_ENCODING

#define VARIABLE\_INT\_TAGBIT\_COUNT (1)

#define VARIABLE\_INT\_BYTE\_SHIFT (8 - VARIABLE\_INT\_TAGBIT\_COUNT)

#define VARIABLE\_INT\_BYTE\_MAX (1 << VARIABLE\_INT\_BYTE\_SHIFT)

#define VARIABLE\_INT\_BYTE\_MASK ~((byte) VARIABLE\_INT\_BYTE\_MAX)

#define SENTINEL\_BYTE\_COUNT 1

#define ONE\_BYTE\_MAX ((byte) 0xfd)

#define TWO\_BYTE\_MAX ((uint16) 0xffff)

#define TWO\_BYTE\_SENTINEL ONE\_BYTE\_MAX + 1

#define FOUR\_BYTE\_SENTINEL ONE\_BYTE\_MAX + 2

#define MIN\_SENTINEL TWO\_BYTE\_SENTINEL

#endif

#if INSTRUMENT\_BUFFER\_INTS

static uint Counts[] = { 0, 0, 0, 0 };

#endif

// Templatized buffer builder for writing fixed-size content

template<typename T, bool useVariableIntEncoding>

struct BufferBuilderOf : BufferBuilder

{

typedef serialization\_alignment T value\_type;

value\_type value;

BufferBuilderOf(LPCWSTR clue, const T & value)

: BufferBuilder(clue), value(value)

{ }

// Assume that the value is 0- for negative values of value, we'll just use the default encoding

bool UseOneByte() const

{

return value >= 0 && value <= ONE\_BYTE\_MAX;

}

bool UseTwoBytes() const

{

return value > ONE\_BYTE\_MAX && value <= TWO\_BYTE\_MAX;

}

uint32 FixOffset(uint32 offset) override

{

this->offset = offset;

if (useVariableIntEncoding)

{

if (UseOneByte())

{

return this->offset + sizeof(serialization\_alignment byte);

}

else if (UseTwoBytes())

{

return this->offset + sizeof(serialization\_alignment uint16) + SENTINEL\_BYTE\_COUNT;

}

return this->offset + sizeof(serialization\_alignment T) + SENTINEL\_BYTE\_COUNT;

}

else

{

return this->offset + sizeof(serialization\_alignment T);

}

}

void Write(\_\_in\_bcount(bufferSize) byte \* buffer, \_\_in uint32 bufferSize) const

{

DebugOnly(uint32 size = sizeof(T));

#if INSTRUMENT\_BUFFER\_INTS

if (value < ((1 << 8)))

{

Counts[0]++;

}

else if (value < ((1 << 16)))

{

Counts[1]++;

}

else if (value < ((1 << 24)))

{

Counts[2]++;

}

else

{

Counts[3]++;

}

#endif

if (useVariableIntEncoding)

{

if (UseOneByte())

{

if (bufferSize - this->offset<sizeof(serialization\_alignment byte))

{

Throw::FatalInternalError();

}

DebugOnly(size = sizeof(byte));

\*(serialization\_alignment byte\*)(buffer + this->offset) = (byte) value;

}

else if (UseTwoBytes())

{

if (bufferSize - this->offset<sizeof(serialization\_alignment uint16))

{

Throw::FatalInternalError();

}

DebugOnly(size = sizeof(uint16));

\*(serialization\_alignment byte\*)(buffer + this->offset) = TWO\_BYTE\_SENTINEL;

\*(serialization\_alignment uint16\*)(buffer + this->offset + SENTINEL\_BYTE\_COUNT) = (uint16) this->value;

}

else

{

if (bufferSize - this->offset<sizeof(serialization\_alignment T))

{

Throw::FatalInternalError();

}

\*(serialization\_alignment byte\*)(buffer + this->offset) = FOUR\_BYTE\_SENTINEL;

\*(serialization\_alignment T\*)(buffer + this->offset + SENTINEL\_BYTE\_COUNT) = this->value;

#if INSTRUMENT\_BUFFER\_INTS

printf("[BCGENSTATS] %d, %d\n", value, sizeof(T));

#endif

}

}

else

{

if (bufferSize - this->offset<sizeof(serialization\_alignment T))

{

Throw::FatalInternalError();

}

\*(serialization\_alignment T\*)(buffer + this->offset) = value;

}

DebugOnly(TraceOutput(buffer, size));

}

};

template<typename T>

struct BufferBuilderOf<T, false>: BufferBuilder

{

typedef serialization\_alignment T value\_type;

value\_type value;

BufferBuilderOf(LPCWSTR clue, const T & value)

: BufferBuilder(clue), value(value)

{ }

uint32 FixOffset(uint32 offset) override

{

this->offset = offset;

return this->offset + sizeof(serialization\_alignment T);

}

void Write(\_\_in\_bcount(bufferSize) byte \* buffer, \_\_in uint32 bufferSize) const

{

if (bufferSize - this->offset<sizeof(serialization\_alignment T))

{

Throw::FatalInternalError();

}

\*(serialization\_alignment T\*)(buffer + this->offset) = value;

DebugOnly(TraceOutput(buffer, sizeof(T)));

}

};

template <typename T>

struct ConstantSizedBufferBuilderOf : BufferBuilderOf<T, false>

{

ConstantSizedBufferBuilderOf(LPCWSTR clue, const T & value)

: BufferBuilderOf(clue, value)

{ }

};

#if VARIABLE\_INT\_ENCODING

typedef BufferBuilderOf<int16, true> BufferBuilderInt16;

typedef BufferBuilderOf<int, true> BufferBuilderInt32;

typedef ConstantSizedBufferBuilderOf<byte> BufferBuilderByte;

typedef ConstantSizedBufferBuilderOf<float> BufferBuilderFloat;

typedef ConstantSizedBufferBuilderOf<double> BufferBuilderDouble;

typedef ConstantSizedBufferBuilderOf<SIMDValue> BufferBuilderSIMD;

#else

typedef ConstantSizedBufferBuilderOf<int16> BufferBuilderInt16;

typedef ConstantSizedBufferBuilderOf<int> BufferBuilderInt32;

typedef ConstantSizedBufferBuilderOf<byte> BufferBuilderByte;

typedef ConstantSizedBufferBuilderOf<float> BufferBuilderFloat;

typedef ConstantSizedBufferBuilderOf<double> BufferBuilderDouble;

typedef ConstantSizedBufferBuilderOf<SIMDValue> BufferBuilderSIMD;

#endif

// A buffer builder which contains a list of buffer builders

struct BufferBuilderList : BufferBuilder

{

regex::ImmutableList<BufferBuilder\*> \* list;

BufferBuilderList(LPCWSTR clue)

: BufferBuilder(clue), list(nullptr)

{ }

uint32 FixOffset(uint32 offset) override

{

this->offset = offset;

return list->Accumulate(offset,[](uint32 size, BufferBuilder \* builder)->uint32 {

return builder->FixOffset(size);

});

}

void Write(\_\_in\_bcount(bufferSize) byte \* buffer, \_\_in uint32 bufferSize) const

{

return list->Iterate([&](BufferBuilder \* builder) {

builder->Write(buffer, bufferSize);

});

}

};

// A buffer builder which points to another buffer builder.

// At write time, it will write the offset from the start of the raw buffer to

// the pointed-to location.

struct BufferBuilderRelativeOffset : BufferBuilder

{

BufferBuilder \* pointsTo;

uint32 additionalOffset;

BufferBuilderRelativeOffset(LPCWSTR clue, BufferBuilder \* pointsTo, uint32 additionalOffset)

: BufferBuilder(clue), pointsTo(pointsTo), additionalOffset(additionalOffset)

{ }

BufferBuilderRelativeOffset(LPCWSTR clue, BufferBuilder \* pointsTo)

: BufferBuilder(clue), pointsTo(pointsTo), additionalOffset(0)

{ }

uint32 FixOffset(uint32 offset) override

{

this->offset = offset;

return this->offset + sizeof(int);

}

void Write(\_\_in\_bcount(bufferSize) byte \* buffer, \_\_in uint32 bufferSize) const

{

if (bufferSize - this->offset<sizeof(int))

{

Throw::FatalInternalError();

}

int offsetOfPointedTo = pointsTo->offset;

\*(int\*)(buffer + this->offset) = offsetOfPointedTo + additionalOffset;

DebugOnly(TraceOutput(buffer, sizeof(int)));

}

};

// A buffer builder which holds a raw byte buffer

struct BufferBuilderRaw : BufferBuilder

{

uint32 size;

const byte \* raw;

BufferBuilderRaw(LPCWSTR clue, \_\_in uint32 size, \_\_in\_bcount(size) const byte \* raw)

: BufferBuilder(clue), size(size), raw(raw)

{ }

uint32 FixOffset(uint32 offset) override

{

this->offset = offset;

return this->offset + size;

}

void Write(\_\_in\_bcount(bufferSize) byte \* buffer, \_\_in uint32 bufferSize) const

{

if (bufferSize - this->offset<size)

{

Throw::FatalInternalError();

}

js\_memcpy\_s(buffer + this->offset, bufferSize-this->offset, raw, size);

DebugOnly(TraceOutput(buffer, size));

}

};

// A buffer builder which aligns its contents to the specified alignment

struct BufferBuilderAligned : BufferBuilder

{

BufferBuilder \* content;

uint32 alignment;

uint32 padding;

BufferBuilderAligned(LPCWSTR clue, BufferBuilder \* content, uint32 alignment)

: BufferBuilder(clue), content(content), alignment(alignment), padding(0)

{ }

uint32 FixOffset(uint32 offset) override

{

this->offset = offset;

// Calculate padding

offset = ::Math::Align(this->offset, this->alignment);

this->padding = offset - this->offset;

Assert(this->padding < this->alignment);

return content->FixOffset(offset);

}

void Write(\_\_in\_bcount(bufferSize) byte \* buffer, \_\_in uint32 bufferSize) const

{

if (bufferSize - this->offset < this->padding)

{

Throw::FatalInternalError();

}

for (uint32 i = 0; i < this->padding; i++)

{

buffer[this->offset + i] = 0;

}

this->content->Write(buffer, bufferSize);

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template <typename T, uint size>

class CircularBuffer

{

public:

CircularBuffer():

writeIndex(0),

filled(false)

{

}

void Clear()

{

this->writeIndex = 0;

this->filled = false;

}

void Add(const T& value)

{

if (!Contains(value))

{

entries[writeIndex] = value;

uint nextIndex = (writeIndex + 1) % size;

if (nextIndex < writeIndex && !filled)

{

filled = true;

}

writeIndex = nextIndex;

}

}

bool Contains(const T& value)

{

for (uint i = 0; i < GetMaxIndex(); i++)

{

if (DefaultComparer<T>::Equals(entries[i], value))

{

return true;

}

}

return false;

}

uint GetMaxIndex()

{

return (filled ? size : writeIndex);

}

const T& Item(uint index)

{

Assert(index < GetMaxIndex());

return entries[index];

}

#ifdef VERBOSE\_EVAL\_MAP

void Dump()

{

Output::Print(L"Length: %d, writeIndex: %d, filled: %d\n", size, writeIndex, filled);

for (uint i = 0; i < GetMaxIndex(); i++)

{

Output::Print(L"Item %d: %s\n", i, entries[i].str.GetBuffer());

}

Output::Flush();

}

#endif

bool IsEmpty()

{

return (writeIndex == 0 && !filled);

}

int GetCount()

{

if (!filled) return writeIndex;

return size;

}

private:

uint writeIndex;

bool filled;

T entries[size];

};

template <class TKey, int MRUSize, class TAllocator = Recycler>

class MRURetentionPolicy

{

public:

typedef CircularBuffer<TKey, MRUSize> TMRUStoreType;

MRURetentionPolicy(TAllocator\* allocator)

{

store = AllocatorNew(TAllocator, allocator, TMRUStoreType);

}

void NotifyAdd(const TKey& key)

{

this->store->Add(key);

}

bool CanEvict(const TKey& key)

{

return !store->Contains(key);

}

void DumpKeepAlives()

{

store->Dump();

}

private:

TMRUStoreType\* store;

};

template <

class TKey,

class TValue,

class TAllocator,

class SizePolicy,

class CacheRetentionPolicy,

template <typename ValueOrKey> class Comparer = DefaultComparer,

template <typename K, typename V> class Entry = SimpleDictionaryEntry

>

class Cache

{

private:

typedef BaseDictionary<TKey, TValue, TAllocator, SizePolicy, Comparer, Entry> TCacheStoreType;

typedef typename TCacheStoreType::AllocatorType AllocatorType;

class CacheStore : public TCacheStoreType

{

public:

CacheStore(AllocatorType\* allocator, int capacity) : BaseDictionary(allocator, capacity), inAdd(false) {};

bool IsInAdd()

{

return this->inAdd;

}

int Add(const TKey& key, const TValue& value)

{

AutoRestoreValue<bool> var(&this->inAdd, true);

return \_\_super::Add(key, value);

}

void SetIsInAdd(bool value) {inAdd = value; }

private:

bool inAdd;

};

public:

typedef TKey KeyType;

typedef TValue ValueType;

typedef void (\*OnItemEvictedCallback)(const TKey& key, TValue value);

Cache(AllocatorType \* allocator, int capacity = 0):

cachePolicyType(allocator)

{

this->cacheStore = AllocatorNew(AllocatorType, allocator, CacheStore, allocator, capacity);

}

int Add(const TKey& key, const TValue& value)

{

int index = this->cacheStore->Add(key, value);

this->cachePolicyType.NotifyAdd(key);

return index;

}

void SetIsInAdd(bool value) {this->cacheStore->SetIsInAdd(value); }

void NotifyAdd(const TKey& key)

{

this->cachePolicyType.NotifyAdd(key);

}

bool TryGetValue(const TKey& key, TValue\* value)

{

return cacheStore->TryGetValue(key, value);

}

bool TryGetReference(const TKey& key, TValue\*\* value, int\* index)

{

return cacheStore->TryGetReference(key, value, index);

}

bool TryGetValueAndRemove(const TKey& key, TValue\* value)

{

return cacheStore->TryGetValueAndRemove(key, value);

}

TKey const& GetKeyAt(const int& index)

{

return cacheStore->GetKeyAt(index);

}

template <class Fn>

void Clean(Fn callback)

{

if (!this->cacheStore->IsInAdd())

{

// Queue up items to be removed

// TODO: Don't use Contains since that's linear- store pointers to the eval map key instead, and set a bit indicating that its in the dictionary?

cacheStore->MapAndRemoveIf([this, callback](const CacheStore::EntryType &entry) {

if (this->cachePolicyType.CanEvict(entry.Key()) || CONFIG\_FLAG(ForceCleanCacheOnCollect))

{

callback(entry.Key(), entry.Value());

if (!CONFIG\_FLAG(ForceCleanCacheOnCollect))

{

return true;

}

}

return false;

});

if (CONFIG\_FLAG(ForceCleanCacheOnCollect))

{

this->cacheStore->Clear();

Assert(this->cacheStore->Count() == 0);

}

}

}

template <class Fn>

void CleanAll(Fn callback)

{

Assert(!this->cacheStore->IsInAdd());

cacheStore->MapAndRemoveIf([this, callback](const CacheStore::EntryType &entry) -> bool {

callback(entry.Key(), entry.Value());

return true;

});

}

void DumpKeepAlives()

{

cachePolicyType.DumpKeepAlives();

}

private:

CacheStore\* cacheStore;

CacheRetentionPolicy cachePolicyType;

};

}

<?xml version="1.0" encoding="utf-8"?>

<Project DefaultTargets="Build" ToolsVersion="12.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Import Condition="'$(ChakraBuildPathImported)'!='true'" Project="$(SolutionDir)Chakra.Build.Paths.props"/>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.ProjectConfiguration.props" />

<PropertyGroup Label="Globals">

<TargetName>Chakra.Common.DataStructures</TargetName>

<ProjectGuid>{5643D42A-C38D-4D82-9662-58470B3AC9F7}</ProjectGuid>

<RootNamespace>JS</RootNamespace>

<Keyword>Win32Proj</Keyword>

</PropertyGroup>

<PropertyGroup Label="Configuration">

<ConfigurationType>StaticLibrary</ConfigurationType>

</PropertyGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />

<Import Project="$(BuildConfigPropsPath)Chakra.Build.props" />

<PropertyGroup>

<\_ProjectFileVersion>10.0.30319.1</\_ProjectFileVersion>

</PropertyGroup>

<ItemDefinitionGroup>

<ClCompile>

<AdditionalIncludeDirectories>

$(MSBuildThisFileDirectory)..;

%(AdditionalIncludeDirectories)

</AdditionalIncludeDirectories>

<PrecompiledHeader>Use</PrecompiledHeader>

<PrecompiledHeaderFile>CommonDataStructuresPch.h</PrecompiledHeaderFile>

</ClCompile>

</ItemDefinitionGroup>

<ItemGroup>

<ClCompile Include="$(MSBuildThisFileDirectory)BigInt.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)BufferBuilder.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)DictionaryStats.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)growingArray.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)ImmutableList.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)InternalString.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Interval.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)FixedBitVector.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)FixedBitVectorEnumerator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)InternalStringNoCaseComparer.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SizePolicy.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SparseBitVector.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CommonDataStructuresPch.cpp">

<PrecompiledHeader>Create</PrecompiledHeader>

</ClCompile>

</ItemGroup>

<ItemGroup>

<ClInclude Include="BigInt.h" />

<ClInclude Include="BaseDictionary.h" />

<ClInclude Include="BufferBuilder.h" />

<ClInclude Include="Cache.h" />

<ClInclude Include="CharacterBuffer.h" />

<ClInclude Include="CommonDataStructuresPch.h" />

<ClInclude Include="ContinuousPageStack.h" />

<ClInclude Include="DoublyLinkedList.h" />

<ClInclude Include="DoublyLinkedListElement.h" />

<ClInclude Include="growingArray.h" />

<ClInclude Include="ImmutableList.h" />

<ClInclude Include="interval.h" />

<ClInclude Include="LineOffsetCache.h" />

<ClInclude Include="LeafValueDictionary.h" />

<ClInclude Include="MruDictionary.h" />

<ClInclude Include="PageStack.h" />

<ClInclude Include="Pair.h" />

<ClInclude Include="Queue.h" />

<ClInclude Include="RegexKey.h" />

<ClInclude Include="SizePolicy.h" />

<ClInclude Include="InternalString.h" />

<ClInclude Include="BitVector.h" />

<ClInclude Include="comparer.h" />

<ClInclude Include="dictionary.h" />

<ClInclude Include="DictionaryEntry.h" />

<ClInclude Include="Dlist.h" />

<ClInclude Include="FixedBitVector.h" />

<ClInclude Include="FixedBitVectorEnumerator.h" />

<ClInclude Include="FixedStack.h" />

<ClInclude Include="HashTable.h" />

<ClInclude Include="KeyValuePair.h" />

<ClInclude Include="LargeStack.h" />

<ClInclude Include="list.h" />

<ClInclude Include="quicksort.h" />

<ClInclude Include="SimpleHashTable.h" />

<ClInclude Include="SList.h" />

<ClInclude Include="SparseArray.h" />

<ClInclude Include="SparseBitVector.h" />

<ClInclude Include="stack.h" />

<ClInclude Include="StringBuilder.h" />

<ClInclude Include="Tree.h" />

<ClInclude Include="UnitBitVector.h" />

<ClInclude Include="WeakReferenceDictionary.h" />

<ClInclude Include="InternalStringNoCaseComparer.h" />

<ClInclude Include="Option.h" />

</ItemGroup>

<ItemGroup>

<None Include="DoublyLinkedList.inl" />

<None Include="DoublyLinkedListElement.inl" />

</ItemGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.targets" Condition="exists('$(BuildConfigPropsPath)Chakra.Build.targets')"/>

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />

</Project>

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

static const charcount\_t MAX\_FAST\_HASH\_LENGTH = 256;

// A buffer of characters, may have embedded null.

template <typename T>

class CharacterBuffer

{

public:

CharacterBuffer() : string(nullptr), len((charcount\_t)-1) {}

CharacterBuffer(T const \* string, charcount\_t len) : string(string), len(len) {}

bool operator==(CharacterBuffer const& other) const

{

Assert(string != nullptr);

if (this->len != other.len)

{

return false;

}

return this->string == other.string || StaticEquals(string, other.string, this->len);

}

operator hash\_t() const

{

Assert(string != nullptr);

return StaticGetHashCode(string, len);

}

int FastHash() const

{

Assert(string != nullptr);

return InternalGetHashCode<true>(string, len);

}

CharacterBuffer& operator=(T const \* s)

{

Assert(s == nullptr);

string = nullptr;

len = (charcount\_t)-1;

return \*this;

}

static bool StaticEquals(\_\_in\_z T const \* s1, \_\_in\_z T const\* s2, \_\_in charcount\_t length);

static int StaticGetHashCode(\_\_in\_z T const \* s, \_\_in charcount\_t length)

{

return InternalGetHashCode<false>(s, length);

}

// This must be identical to Trident's getHash function in fastDOMCompiler.pl

template <bool fastHash>

static int InternalGetHashCode(\_\_in\_z T const \* s, \_\_in charcount\_t length)

{

// TODO: This hash performs poorly on small strings, consider finding a better hash function

// now that some type handlers hash by string instead of PropertyId.

int hash = 0;

charcount\_t hashLength = length;

if (fastHash)

{

hashLength = min(length, MAX\_FAST\_HASH\_LENGTH);

}

for (charcount\_t i = 0; i < hashLength; i++)

{

hash = \_rotl(hash, 7);

hash ^= s[i];

}

return hash;

}

T const \* GetBuffer() const { return string; }

charcount\_t GetLength() const { return len; }

private:

T const \* string;

charcount\_t len;

};

template<>

inline bool

CharacterBuffer<WCHAR>::StaticEquals(\_\_in\_z WCHAR const \* s1, \_\_in\_z WCHAR const \* s2, \_\_in charcount\_t length)

{

return wmemcmp(s1, s2, length) == 0;

}

template<>

inline bool

CharacterBuffer<unsigned char>::StaticEquals(\_\_in\_z unsigned char const \* s1, \_\_in\_z unsigned char const \*s2, \_\_in charcount\_t length)

{

return memcmp(s1, s2, length) == 0;

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonMinMemory.h"

// === C Runtime Header Files ===

#include <wchar.h>

#if defined(\_UCRT)

#include <cmath>

#else

#include <math.h>

#endif

// === Codex Header Files ===

#include "codex\Utf8Codex.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

typedef uint hash\_t;

#define TAGHASH(hash) ((static\_cast<hash\_t>(hash) << 1) | 1)

#define UNTAGHASH(hash) (static\_cast<hash\_t>(hash) >> 1)

// This comparer can create good hash codes and comparisons for all value,

// pointer and string types using template specialization.

template <typename T>

struct DefaultComparer

{

\_\_inline static bool Equals(const T &x, const T &y)

{

return x == y;

}

\_\_inline static hash\_t GetHashCode(const T &i)

{

return (hash\_t)i;

}

};

template <>

struct DefaultComparer<double>

{

\_\_inline static bool Equals(double x, double y)

{

return x == y;

}

\_\_inline static hash\_t GetHashCode(double d)

{

\_\_int64 i64 = \*(\_\_int64\*)&d;

return (uint)((i64>>32) ^ (uint)i64);

}

};

template <typename T>

struct DefaultComparer<T \*>

{

\_\_inline static bool Equals(T \* x, T \* y)

{

return x == y;

}

\_\_inline static hash\_t GetHashCode(T \* i)

{

// Shifting helps us eliminate any sameness due to our alignment strategy.

// TODO: This works for Arena memory only. Recycler memory is 16 byte aligned.

// Find a good universal hash for pointers.

uint hash = (uint)(((size\_t)i) >> ArenaAllocator::ObjectAlignmentBitShift);

return hash;

}

};

template <>

struct DefaultComparer<size\_t>

{

\_\_inline static bool Equals(size\_t x, size\_t y)

{

return x == y;

}

\_\_inline static uint GetHashCode(size\_t i)

{

#if \_WIN64

// For 64 bits we want all 64 bits of the pointer to be represented in the hash code.

uint32 hi = ((UINT\_PTR) i >> 32);

uint32 lo = (uint32) (i & 0xFFFFFFFF);

uint hash = hi ^ lo;

#else

uint hash = i;

#endif

return hash;

}

static int Compare(size\_t i1, size\_t i2)

{

if (i1 < i2)

{

return -1;

}

else if (i1 > i2)

{

return 1;

}

else

{

return 0;

}

}

};

// This specialization does a better job of creating hash codes

// for recycler pointers.

template <typename T>

struct RecyclerPointerComparer

{

\_\_inline static bool Equals(T x, T y)

{

return x == y;

}

\_\_inline static hash\_t GetHashCode(T i)

{

// Shifting helps us eliminate any sameness due to our alignment strategy.

// TODO: This works for Recycler memory only. Arena memory is 8 byte aligned.

// Find a good universal hash for pointers.

uint hash = (uint)(((size\_t)i) >> HeapConstants::ObjectAllocationShift);

return hash;

}

};

template <>

struct DefaultComparer<GUID>

{

\_\_inline static bool Equals(GUID const& x, GUID const& y)

{

return x == y;

}

\_\_inline static hash\_t GetHashCode(GUID const& guid)

{

char\* p = (char\*)&guid;

int hash = 0;

for (int i = 0; i < sizeof(GUID); i++)

{

hash = \_rotl(hash, 7);

hash ^= (uint32)(p[i]);

}

return hash;

}

};

template<typename T>

struct StringComparer

{

\_\_inline static bool Equals(T str1, T str2)

{

return ::wcscmp(str1, str2) == 0;

}

\_\_inline static hash\_t GetHashCode(T str)

{

int hash = 0;

while (\*str)

{

hash = \_rotl(hash, 7);

hash ^= \*str;

str++;

}

return hash;

}

static int Compare(T str1, T str2)

{

return ::wcscmp(str1, str2);

}

};

template<>

struct DefaultComparer<WCHAR\*> : public StringComparer<WCHAR\*> {};

template<>

struct DefaultComparer<const WCHAR\*> : public StringComparer<const WCHAR\*> {};

template <typename T, typename TComparer>

struct SpecializedComparer

{

template <typename T> class TComparerType;

template <> class TComparerType<T> : public TComparer {};

};

namespace regex

{

template <class T> struct Comparer

{

public:

virtual bool Equals(T item1, T item2) = 0;

virtual int GetHashCode(T item) = 0;

virtual int Compare(T item1, T item2) = 0;

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// -----------------------------------------------------------------------------------------------------------------------------

// ContinuousPageStack declaration

// -----------------------------------------------------------------------------------------------------------------------------

template<const size\_t InitialPageCount = 1>

class ContinuousPageStack : protected Allocator

{

protected:

class Iterator

{

protected:

size\_t nextTop;

protected:

\_\_inline Iterator(const ContinuousPageStack &stack);

public:

\_\_inline size\_t Position() const;

\_\_inline operator bool() const;

};

friend class Iterator;

private:

PageAllocator \*const pageAllocator;

PageAllocation \*pageAllocation;

size\_t bufferSize;

protected:

size\_t nextTop;

protected:

\_\_inline ContinuousPageStack(PageAllocator \*const pageAllocator, void (\*const outOfMemoryFunc)());

~ContinuousPageStack();

\_\_inline char \*Buffer() const;

public:

\_\_inline bool IsEmpty() const;

\_\_inline void Clear();

\_\_inline size\_t Position() const;

\_\_inline char\* Push(const size\_t size);

\_\_inline char\* Top(const size\_t size) const;

\_\_inline char\* Pop(const size\_t size);

\_\_inline void UnPop(const size\_t size);

\_\_inline void PopTo(const size\_t position);

private:

void Resize(size\_t requestedSize);

};

// -----------------------------------------------------------------------------------------------------------------------------

// ContinuousPageStackOfFixedElements declaration

// -----------------------------------------------------------------------------------------------------------------------------

template<class T, const size\_t InitialPageCount = 1>

class ContinuousPageStackOfFixedElements : public ContinuousPageStack<InitialPageCount>

{

public:

class Iterator : public ContinuousPageStack<InitialPageCount>::Iterator

{

private:

const ContinuousPageStackOfFixedElements &stack;

public:

\_\_inline Iterator(const ContinuousPageStackOfFixedElements &stack);

\_\_inline T &operator \*() const;

\_\_inline T \*operator ->() const;

\_\_inline Iterator &operator ++(); // pre-increment

\_\_inline Iterator operator ++(int); // post-increment

};

friend class Iterator;

public:

\_\_inline ContinuousPageStackOfFixedElements(PageAllocator \*const pageAllocator, void (\*const outOfMemoryFunc)());

public:

\_\_inline char\* Push();

\_\_inline T\* Top() const;

\_\_inline T\* Pop();

\_\_inline void UnPop();

\_\_inline void PopTo(const size\_t position);

};

// -----------------------------------------------------------------------------------------------------------------------------

// ContinuousPageStackOfVariableElements declaration

// -----------------------------------------------------------------------------------------------------------------------------

template<class T, const size\_t InitialPageCount = 1>

class ContinuousPageStackOfVariableElements : public ContinuousPageStack<InitialPageCount>

{

public:

class Iterator : public ContinuousPageStack<InitialPageCount>::Iterator

{

private:

size\_t topElementSize;

const ContinuousPageStackOfVariableElements &stack;

public:

\_\_inline Iterator(const ContinuousPageStackOfVariableElements &stack);

\_\_inline T &operator \*() const;

\_\_inline T \*operator ->() const;

\_\_inline Iterator &operator ++(); // pre-increment

\_\_inline Iterator operator ++(int); // post-increment

};

friend class Iterator;

private:

class VariableElement

{

private:

const size\_t previousElementSize;

char data[0];

public:

\_\_inline VariableElement(const size\_t previousElementSize);

public:

template<class ActualT> \_\_inline static size\_t Size();

\_\_inline size\_t PreviousElementSize() const;

\_\_inline char \*Data();

};

private:

size\_t topElementSize;

public:

\_\_inline ContinuousPageStackOfVariableElements(PageAllocator \*const pageAllocator, void (\*const outOfMemoryFunc)());

public:

template<class ActualT> \_\_inline char\* Push();

\_\_inline T\* Top() const;

\_\_inline T\* Pop();

template<class ActualT> \_\_inline void UnPop();

\_\_inline void PopTo(const size\_t position);

};

// -----------------------------------------------------------------------------------------------------------------------------

// ContinuousPageStack definition

// -----------------------------------------------------------------------------------------------------------------------------

// --------

// Iterator

// --------

template<const size\_t InitialPageCount>

\_\_inline ContinuousPageStack<InitialPageCount>::Iterator::Iterator(const ContinuousPageStack &stack)

: nextTop(stack.nextTop)

{

}

template<const size\_t InitialPageCount>

\_\_inline size\_t ContinuousPageStack<InitialPageCount>::Iterator::Position() const

{

return nextTop;

}

template<const size\_t InitialPageCount>

\_\_inline ContinuousPageStack<InitialPageCount>::Iterator::operator bool() const

{

return nextTop != 0;

}

// -------------------

// ContinuousPageStack

// -------------------

template<const size\_t InitialPageCount>

\_\_inline ContinuousPageStack<InitialPageCount>::ContinuousPageStack(

PageAllocator \*const pageAllocator,

void (\*const outOfMemoryFunc)())

: Allocator(outOfMemoryFunc), pageAllocator(pageAllocator), bufferSize(0), nextTop(0)

{

Assert(pageAllocator);

}

template<const size\_t InitialPageCount>

ContinuousPageStack<InitialPageCount>::~ContinuousPageStack()

{

if(bufferSize && !pageAllocator->IsClosed())

pageAllocator->ReleaseAllocation(pageAllocation);

}

template<const size\_t InitialPageCount>

\_\_inline char \*ContinuousPageStack<InitialPageCount>::Buffer() const

{

Assert(bufferSize);

return pageAllocation->GetAddress();

}

template<const size\_t InitialPageCount>

\_\_inline bool ContinuousPageStack<InitialPageCount>::IsEmpty() const

{

return nextTop == 0;

}

template<const size\_t InitialPageCount>

\_\_inline void ContinuousPageStack<InitialPageCount>::Clear()

{

nextTop = 0;

}

template<const size\_t InitialPageCount>

\_\_inline size\_t ContinuousPageStack<InitialPageCount>::Position() const

{

return nextTop;

}

template<const size\_t InitialPageCount>

\_\_inline char\* ContinuousPageStack<InitialPageCount>::Push(const size\_t size)

{

Assert(size);

if(bufferSize - nextTop < size)

Resize(size);

char\* const res = Buffer() + nextTop;

nextTop += size;

return res;

}

template<const size\_t InitialPageCount>

\_\_inline char\* ContinuousPageStack<InitialPageCount>::Top(const size\_t size) const

{

if (nextTop == 0)

return 0;

else

{

Assert(size != 0);

Assert(size <= nextTop);

return Buffer() + (nextTop - size);

}

}

template<const size\_t InitialPageCount>

\_\_inline char\* ContinuousPageStack<InitialPageCount>::Pop(const size\_t size)

{

if (nextTop == 0)

return 0;

else

{

Assert(size != 0);

Assert(nextTop >= size);

nextTop -= size;

return Buffer() + nextTop;

}

}

template<const size\_t InitialPageCount>

\_\_inline void ContinuousPageStack<InitialPageCount>::UnPop(const size\_t size)

{

Assert(size != 0);

Assert(nextTop + size <= bufferSize);

nextTop += size;

}

template<const size\_t InitialPageCount>

void ContinuousPageStack<InitialPageCount>::Resize(size\_t requestedSize)

{

Assert(requestedSize);

Assert(requestedSize <= InitialPageCount \* AutoSystemInfo::PageSize);

if(!bufferSize)

{

pageAllocation = pageAllocator->AllocAllocation(InitialPageCount);

if (!pageAllocation)

{

outOfMemoryFunc();

AnalysisAssert(false);

}

bufferSize = pageAllocation->GetSize();

return;

}

PageAllocation \*const newPageAllocation = pageAllocator->AllocAllocation(pageAllocation->GetPageCount() \* 2);

if (!newPageAllocation)

{

outOfMemoryFunc();

AnalysisAssert(false);

}

js\_memcpy\_s(newPageAllocation->GetAddress(), newPageAllocation->GetSize(), Buffer(), nextTop);

pageAllocator->ReleaseAllocation(pageAllocation);

pageAllocation = newPageAllocation;

bufferSize = newPageAllocation->GetSize();

}

template<const size\_t InitialPageCount>

\_\_inline void ContinuousPageStack<InitialPageCount>::PopTo(const size\_t position)

{

Assert(position <= nextTop);

nextTop = position;

}

// -----------------------------------------------------------------------------------------------------------------------------

// ContinuousPageStackOfFixedElements definition

// -----------------------------------------------------------------------------------------------------------------------------

// --------

// Iterator

// --------

template<class T, const size\_t InitialPageCount>

\_\_inline ContinuousPageStackOfFixedElements<T, InitialPageCount>::Iterator::Iterator(

const ContinuousPageStackOfFixedElements &stack)

: ContinuousPageStack<InitialPageCount>::Iterator(stack), stack(stack)

{

}

template<class T, const size\_t InitialPageCount>

\_\_inline T &ContinuousPageStackOfFixedElements<T, InitialPageCount>::Iterator::operator \*() const

{

Assert(\*this);

Assert(nextTop <= stack.nextTop);

return \*reinterpret\_cast<T \*>(&stack.Buffer()[nextTop - sizeof(T)]);

}

template<class T, const size\_t InitialPageCount>

\_\_inline T \*ContinuousPageStackOfFixedElements<T, InitialPageCount>::Iterator::operator ->() const

{

return &\*\*this;

}

template<class T, const size\_t InitialPageCount>

\_\_inline typename ContinuousPageStackOfFixedElements<T, InitialPageCount>::Iterator &ContinuousPageStackOfFixedElements<T, InitialPageCount>::Iterator::operator ++() // pre-increment

{

Assert(\*this);

nextTop -= sizeof(T);

return \*this;

}

template<class T, const size\_t InitialPageCount>

\_\_inline typename ContinuousPageStackOfFixedElements<T, InitialPageCount>::Iterator ContinuousPageStackOfFixedElements<T, InitialPageCount>::Iterator::operator ++(int) // post-increment

{

Iterator it(\*this);

++\*this;

return it;

}

// ----------------------------------

// ContinuousPageStackOfFixedElements

// ----------------------------------

template<class T, const size\_t InitialPageCount>

\_\_inline ContinuousPageStackOfFixedElements<T, InitialPageCount>::ContinuousPageStackOfFixedElements(

PageAllocator \*const pageAllocator,

void (\*const outOfMemoryFunc)())

: ContinuousPageStack(pageAllocator, outOfMemoryFunc)

{

}

template<class T, const size\_t InitialPageCount>

\_\_inline char\* ContinuousPageStackOfFixedElements<T, InitialPageCount>::Push()

{

return ContinuousPageStack::Push(sizeof(T));

}

template<class T, const size\_t InitialPageCount>

\_\_inline T\* ContinuousPageStackOfFixedElements<T, InitialPageCount>::Top() const

{

return reinterpret\_cast<T\*>(ContinuousPageStack::Top(sizeof(T)));

}

template<class T, const size\_t InitialPageCount>

\_\_inline T\* ContinuousPageStackOfFixedElements<T, InitialPageCount>::Pop()

{

return reinterpret\_cast<T\*>(ContinuousPageStack::Pop(sizeof(T)));

}

template<class T, const size\_t InitialPageCount>

\_\_inline void ContinuousPageStackOfFixedElements<T, InitialPageCount>::UnPop()

{

return ContinuousPageStack::UnPop(sizeof(T));

}

template<class T, const size\_t InitialPageCount>

\_\_inline void ContinuousPageStackOfFixedElements<T, InitialPageCount>::PopTo(const size\_t position)

{

ContinuousPageStack::PopTo(position);

}

// -----------------------------------------------------------------------------------------------------------------------------

// ContinuousPageStackOfVariableElements definition

// -----------------------------------------------------------------------------------------------------------------------------

// --------

// Iterator

// --------

template<class T, const size\_t InitialPageCount>

\_\_inline ContinuousPageStackOfVariableElements<T, InitialPageCount>::Iterator::Iterator(

const ContinuousPageStackOfVariableElements &stack)

: ContinuousPageStack<InitialPageCount>::Iterator(stack), topElementSize(stack.topElementSize), stack(stack)

{

}

template<class T, const size\_t InitialPageCount>

\_\_inline T &ContinuousPageStackOfVariableElements<T, InitialPageCount>::Iterator::operator \*() const

{

Assert(\*this);

Assert(nextTop <= stack.nextTop);

return \*reinterpret\_cast<T\*>(reinterpret\_cast<VariableElement \*>(&stack.Buffer()[nextTop - topElementSize])->Data());

}

template<class T, const size\_t InitialPageCount>

\_\_inline T \*ContinuousPageStackOfVariableElements<T, InitialPageCount>::Iterator::operator ->() const

{

return &\*\*this;

}

template<class T, const size\_t InitialPageCount>

\_\_inline typename ContinuousPageStackOfVariableElements<T, InitialPageCount>::Iterator &ContinuousPageStackOfVariableElements<T, InitialPageCount>::Iterator::operator ++() // pre-increment

{

Assert(\*this);

Assert(nextTop <= stack.nextTop);

topElementSize = reinterpret\_cast<VariableElement \*>(&stack.Buffer()[nextTop -= topElementSize])->PreviousElementSize();

return \*this;

}

template<class T, const size\_t InitialPageCount>

\_\_inline typename ContinuousPageStackOfVariableElements<T, InitialPageCount>::Iterator ContinuousPageStackOfVariableElements<T, InitialPageCount>::Iterator::operator ++(int) // post-increment

{

Iterator it(\*this);

++\*this;

return it;

}

// ---------------

// VariableElement

// ---------------

template<class T, const size\_t InitialPageCount>

\_\_inline ContinuousPageStackOfVariableElements<T, InitialPageCount>::VariableElement::VariableElement(

const size\_t previousElementSize)

: previousElementSize(previousElementSize)

{

}

template<class T, const size\_t InitialPageCount>

template<class ActualT>

\_\_inline size\_t ContinuousPageStackOfVariableElements<T, InitialPageCount>::VariableElement::Size()

{

return sizeof(VariableElement) + sizeof(ActualT);

}

template<class T, const size\_t InitialPageCount>

\_\_inline size\_t ContinuousPageStackOfVariableElements<T, InitialPageCount>::VariableElement::PreviousElementSize() const

{

return previousElementSize;

}

template<class T, const size\_t InitialPageCount>

\_\_inline char\* ContinuousPageStackOfVariableElements<T, InitialPageCount>::VariableElement::Data()

{

return data;

}

// -------------------------------------

// ContinuousPageStackOfVariableElements

// -------------------------------------

template<class T, const size\_t InitialPageCount>

\_\_inline ContinuousPageStackOfVariableElements<T, InitialPageCount>::ContinuousPageStackOfVariableElements(

PageAllocator \*const pageAllocator,

void (\*const outOfMemoryFunc)())

: ContinuousPageStack(pageAllocator, outOfMemoryFunc)

{

}

template<class T, const size\_t InitialPageCount>

template<class ActualT>

\_\_inline char\* ContinuousPageStackOfVariableElements<T, InitialPageCount>::Push()

{

TemplateParameter::SameOrDerivedFrom<ActualT, T>(); // ActualT must be the same type as, or a type derived from, T

VariableElement \*const element =

new(ContinuousPageStack::Push(VariableElement::Size<ActualT>())) VariableElement(topElementSize);

topElementSize = VariableElement::Size<ActualT>();

return element->Data();

}

template<class T, const size\_t InitialPageCount>

\_\_inline T\* ContinuousPageStackOfVariableElements<T, InitialPageCount>::Top() const

{

VariableElement\* const element = reinterpret\_cast<VariableElement\*>(ContinuousPageStack::Top(topElementSize));

return element == 0 ? 0 : reinterpret\_cast<T\*>(element->Data());

}

template<class T, const size\_t InitialPageCount>

\_\_inline T\* ContinuousPageStackOfVariableElements<T, InitialPageCount>::Pop()

{

VariableElement \*const element = reinterpret\_cast<VariableElement\*>(ContinuousPageStack::Pop(topElementSize));

if (element == 0)

return 0;

else

{

topElementSize = element->PreviousElementSize();

return reinterpret\_cast<T\*>(element->Data());

}

}

template<class T, const size\_t InitialPageCount>

template<class ActualT>

\_\_inline void ContinuousPageStackOfVariableElements<T, InitialPageCount>::UnPop()

{

TemplateParameter::SameOrDerivedFrom<ActualT, T>(); // ActualT must be the same type as, or a type derived from, T

ContinuousPageStack::UnPop(VariableElement::Size<ActualT>());

Assert(reinterpret\_cast<VariableElement\*>(ContinuousPageStack::Top(VariableElement::Size<ActualT>()))->PreviousElementSize() == topElementSize);

topElementSize = VariableElement::Size<ActualT>();

}

template<class T, const size\_t InitialPageCount>

\_\_inline void ContinuousPageStackOfVariableElements<T, InitialPageCount>::PopTo(const size\_t position)

{

Assert(position <= nextTop);

if(position != nextTop)

{

Assert(position + sizeof(VariableElement) <= nextTop);

topElementSize = reinterpret\_cast<VariableElement \*>(&Buffer()[position])->PreviousElementSize();

}

ContinuousPageStack::PopTo(position);

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template <class TKey, class TValue> class WeakRefDictionaryEntry

{

public:

static const int INVALID\_HASH\_VALUE = 0;

hash\_t hash; // Lower 31 bits of hash code << 1 | 1, 0 if unused

int next; // Index of next entry, -1 if last

const RecyclerWeakReference<TKey>\* key; // Key of entry- this entry holds a weak reference to the key

TValue value; // Value of entry

};

// TODO: convert to BaseDictionary- easier now to have custom dictionary since this does compacting

// and weak reference resolution

template <class TKey, class TValue, class KeyComparer = DefaultComparer<const TKey\*>, bool cleanOnInsert = true> class WeaklyReferencedKeyDictionary

{

public:

typedef WeakRefDictionaryEntry<TKey, TValue> EntryType;

typedef TKey KeyType;

typedef TValue ValueType;

typedef void (\*EntryRemovalCallbackMethodType)(const EntryType& e, void\* cookie);

struct EntryRemovalCallback

{

EntryRemovalCallbackMethodType fnCallback;

void\* cookie;

};

private:

int size;

int\* buckets;

EntryType \* entries;

int count;

int version;

int freeList;

int freeCount;

Recycler\* recycler;

EntryRemovalCallback entryRemovalCallback;

uint lastWeakReferenceCleanupId;

bool disableCleanup;

public:

// Allow WeaklyReferencedKeyDictionary field to be inlined in classes with DEFINE\_VTABLE\_CTOR\_MEMBER\_INIT

WeaklyReferencedKeyDictionary(VirtualTableInfoCtorEnum) { }

WeaklyReferencedKeyDictionary(Recycler\* recycler, int capacity = 0, EntryRemovalCallback\* pEntryRemovalCallback = nullptr):

buckets(nullptr),

size(0),

entries(nullptr),

count(0),

version(0),

freeList(0),

freeCount(0),

recycler(recycler),

lastWeakReferenceCleanupId(recycler->GetWeakReferenceCleanupId()),

disableCleanup(false)

{

if (pEntryRemovalCallback != nullptr)

{

this->entryRemovalCallback.fnCallback = pEntryRemovalCallback->fnCallback;

this->entryRemovalCallback.cookie = pEntryRemovalCallback->cookie;

}

else

{

this->entryRemovalCallback.fnCallback = nullptr;

}

if (capacity > 0) { Initialize(capacity); }

}

~WeaklyReferencedKeyDictionary()

{

}

inline int Count()

{

return count - freeCount;

}

TValue Item(TKey\* key)

{

int i = FindEntry(key);

if (i >= 0) return entries[i].value;

Js::Throw::FatalInternalError();

}

void Item(TKey\* key, const TValue value)

{

Insert(key, value, false);

}

const TValue& GetValueAt(const int& index) const

{

if (index >= 0 && index < count)

{

return entries[index].value;

}

Js::Throw::FatalInternalError();

}

bool TryGetValue(const TKey\* key, TValue\* value)

{

int i = FindEntry<TKey>(key);

if (i >= 0)

{

\*value = entries[i].value;

return true;

}

return false;

}

bool TryGetValueAndRemove(const TKey\* key, TValue\* value)

{

if (buckets == nullptr) return false;

hash\_t hash = GetHashCode(key);

uint targetBucket = hash % size;

int last = -1;

int i = 0;

if ((i = FindEntry<TKey>(key, hash, targetBucket, last)) != -1)

{

\*value = entries[i].value;

RemoveEntry(i, last, targetBucket);

return true;

}

return false;

}

template <typename TLookup>

inline TValue Lookup(const TLookup\* key, TValue defaultValue, \_\_out TKey const\*\* pKeyOut)

{

int i = FindEntry(key);

if (i >= 0)

{

(\*pKeyOut) = entries[i].key->Get();

return entries[i].value;

}

(\*pKeyOut) = nullptr;

return defaultValue;

}

inline TValue Lookup(const TKey\* key, TValue defaultValue)

{

int i = FindEntry(key);

if (i >= 0)

{

return entries[i].value;

}

return defaultValue;

}

const RecyclerWeakReference<TKey>\* Add(TKey\* key, TValue value)

{

return Insert(key, value, true);

}

const RecyclerWeakReference<TKey>\* UncheckedAdd(TKey\* key, TValue value)

{

return Insert(key, value, true, false);

}

const RecyclerWeakReference<TKey>\* UncheckedAdd(const RecyclerWeakReference<TKey>\* weakRef, TValue value)

{

return UncheckedInsert(weakRef, value);

}

template<class Fn>

void Map(Fn fn)

{

for(int i = 0; i < size; i++)

{

if(buckets[i] != -1)

{

for(int previousIndex = -1, currentIndex = buckets[i]; currentIndex != -1;)

{

EntryType &currentEntry = entries[currentIndex];

TKey \* key = currentEntry.key->Get();

if(key != nullptr)

{

fn(key, currentEntry.value, currentEntry.key);

// Keep the entry

previousIndex = currentIndex;

currentIndex = currentEntry.next;

}

else

{

// Remove the entry

const int nextIndex = currentEntry.next;

RemoveEntry(currentIndex, previousIndex, i);

currentIndex = nextIndex;

}

}

}

}

}

void SetDisableCleanup(bool disableCleanup)

{

this->disableCleanup = disableCleanup;

}

bool GetDisableCleanup()

{

return this->disableCleanup;

}

bool Clean()

{

if (!disableCleanup && recycler->GetWeakReferenceCleanupId() != this->lastWeakReferenceCleanupId)

{

Map([](TKey \* key, TValue value, const RecyclerWeakReference<TKey>\* weakRef) {});

this->lastWeakReferenceCleanupId = recycler->GetWeakReferenceCleanupId();

}

return freeCount > 0;

}

void Clear()

{

if (count > 0)

{

for (int i = 0; i < size; i++) buckets[i] = -1;

memset(entries, 0, sizeof(EntryType) \* size);

freeList = -1;

count = 0;

freeCount = 0;

}

}

void EnsureCapacity()

{

if (freeCount == 0 && count == size)

{

if (cleanOnInsert && Clean())

{

Assert(freeCount > 0);

}

else

{

Resize();

}

}

}

private:

const RecyclerWeakReference<TKey>\* UncheckedInsert(const RecyclerWeakReference<TKey>\* weakRef, TValue value)

{

if (buckets == nullptr) Initialize(0);

int hash = GetHashCode(weakRef->FastGet());

uint bucket = (uint)hash % size;

Assert(FindEntry(weakRef->FastGet()) == -1);

return Insert(weakRef, value, hash, bucket);

}

const RecyclerWeakReference<TKey>\* Insert(TKey\* key, TValue value, bool add, bool checkForExisting = true)

{

if (buckets == nullptr) Initialize(0);

hash\_t hash = GetHashCode(key);

uint bucket = hash % size;

if (checkForExisting)

{

int previous = -1;

int i = FindEntry(key, hash, bucket, previous);

if (i != -1)

{

if (add)

{

Js::Throw::FatalInternalError();

}

entries[i].value = value;

version++;

return entries[i].key;

}

}

// We know we need to insert- so first try creating the weak reference, before adding it to

// the dictionary. If we OOM here, we still leave the dictionary as we found it.

const RecyclerWeakReference<TKey>\* weakRef = recycler->CreateWeakReferenceHandle<TKey>(key);

return Insert(weakRef, value, hash, bucket);

}

const RecyclerWeakReference<TKey>\* Insert(const RecyclerWeakReference<TKey>\* weakRef, TValue value, int hash, uint bucket)

{

int index;

if (freeCount > 0)

{

index = freeList;

freeList = entries[index].next;

freeCount--;

}

else

{

if (count == size)

{

if (cleanOnInsert && Clean())

{

index = freeList;

freeList = entries[index].next;

freeCount--;

}

else

{

Resize();

bucket = (uint)hash % size;

index = count;

count++;

}

}

else

{

index = count;

count++;

}

}

entries[index].next = buckets[bucket];

entries[index].key = weakRef;

entries[index].hash = hash;

entries[index].value = value;

buckets[bucket] = index;

version++;

return entries[index].key;

}

void Resize()

{

int newSize = PrimePolicy::GetSize(count \* 2);

if (newSize <= count)

{

// throw OOM if we can't increase the dictionary size

Js::Throw::OutOfMemory();

}

int\* newBuckets = RecyclerNewArrayLeaf(recycler, int, newSize);

for (int i = 0; i < newSize; i++) newBuckets[i] = -1;

EntryType\* newEntries = RecyclerNewArray(recycler, EntryType, newSize);

js\_memcpy\_s(newEntries, sizeof(EntryType) \* newSize, entries, sizeof(EntryType) \* count);

AnalysisAssert(count < newSize);

for (int i = 0; i < count; i++)

{

uint bucket = (uint)newEntries[i].hash % newSize;

newEntries[i].next = newBuckets[bucket];

newBuckets[bucket] = i;

}

buckets = newBuckets;

size = newSize;

entries = newEntries;

}

template <typename TLookup>

\_\_inline hash\_t GetHashCode(const TLookup\* key)

{

return TAGHASH(KeyComparer::GetHashCode(key));

}

template <typename TLookup>

inline int FindEntry(const TLookup\* key)

{

if (buckets != nullptr)

{

hash\_t hash = GetHashCode(key);

uint bucket = (uint)hash % size;

int previous = -1;

return FindEntry(key, hash, bucket, previous);

}

return -1;

}

template <typename TLookup>

inline int FindEntry(const TLookup\* key, hash\_t const hash, uint& bucket, int& previous)

{

if (buckets != nullptr)

{

BOOL inSweep = this->recycler->IsSweeping();

previous = -1;

for (int i = buckets[bucket]; i >= 0; )

{

if (entries[i].hash == hash)

{

TKey\* strongRef = nullptr;

if (!inSweep)

{

// Quickly check for null if we're not in sweep- if it's null, it's definitely been collected

// so remove

strongRef = entries[i].key->FastGet();

}

else

{

// If we're in sweep, use the slower Get which checks if the object is getting collected

// This could return null too but we won't clean it up now, we'll clean it up later

strongRef = entries[i].key->Get();

}

if (strongRef == nullptr)

{

i = RemoveEntry(i, previous, bucket);

continue;

}

else

{

// if we get here, strongRef is not null

if (KeyComparer::Equals(strongRef, key))

return i;

}

}

previous = i;

i = entries[i].next;

}

}

return -1;

}

void Initialize(int capacity)

{

int size = PrimePolicy::GetSize(capacity);

int\* buckets = RecyclerNewArrayLeaf(recycler, int, size);

EntryType \* entries = RecyclerNewArray(recycler, EntryType, size);

// No need for auto pointers here since these are both recycler

// allocated objects

if (buckets != nullptr && entries != nullptr)

{

this->size = size;

this->buckets = buckets;

for (int i = 0; i < size; i++) buckets[i] = -1;

this->entries = entries;

this->freeList = -1;

}

}

int RemoveEntry(int i, int previous, uint bucket)

{

int next = entries[i].next;

if (previous < 0) // Previous < 0 => first node

{

buckets[bucket] = entries[i].next;

}

else

{

entries[previous].next = entries[i].next;

}

if (this->entryRemovalCallback.fnCallback != nullptr)

{

this->entryRemovalCallback.fnCallback(entries[i], this->entryRemovalCallback.cookie);

}

entries[i].next = freeList;

entries[i].key = nullptr;

entries[i].hash = EntryType::INVALID\_HASH\_VALUE;

// Hold onto the pid here so that we can reuse it

// entries[i].value = nullptr;

freeList = i;

freeCount++;

version++;

return next;

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template <class TValue>

class BaseValueEntry

{

protected:

TValue value; // data of entry

void Set(TValue const& value)

{

this->value = value;

}

public:

int next; // Index of next entry, -1 if last

static bool SupportsCleanup()

{

return false;

}

static bool NeedsCleanup(BaseValueEntry<TValue>&)

{

return false;

}

TValue const& Value() const { return value; }

TValue& Value() { return value; }

void SetValue(TValue const& value) { this->value = value; }

};

template <class TValue>

class ValueEntry: public BaseValueEntry<TValue>

{

public:

void Clear()

{

}

};

// Class specialization for pointer values to support clearing

template <class TValue>

class ValueEntry<TValue\*>: public BaseValueEntry<TValue\*>

{

public:

void Clear()

{

this->value = nullptr;

}

};

template <>

class ValueEntry<bool>: public BaseValueEntry<bool>

{

public:

void Clear()

{

this->value = false;

}

};

template <>

class ValueEntry<int>: public BaseValueEntry<int>

{

public:

void Clear()

{

this->value = 0;

}

};

template <>

class ValueEntry<uint>: public BaseValueEntry<uint>

{

public:

void Clear()

{

this->value = 0;

}

};

template<class TKey, class TValue>

struct ValueToKey

{

static TKey ToKey(const TValue &value) { return static\_cast<TKey>(value); }

};

// Used by BaseHashSet, the default is that the key is the same as the value

template <class TKey, class TValue>

class ImplicitKeyValueEntry : public ValueEntry<TValue>

{

public:

TKey Key() const { return ValueToKey<TKey, TValue>::ToKey(value); }

void Set(TKey const& key, TValue const& value)

{

\_\_super::Set(value);

}

};

template <class TKey, class TValue>

class BaseKeyValueEntry : public ValueEntry<TValue>

{

protected:

TKey key; // key of entry

void Set(TKey const& key, TValue const& value)

{

\_\_super::Set(value);

this->key = key;

}

public:

TKey const& Key() const { return key; }

};

template <class TKey, class TValue>

class KeyValueEntry : public BaseKeyValueEntry<TKey, TValue>

{

};

template <class TKey, class TValue>

class KeyValueEntry<TKey\*, TValue> : public BaseKeyValueEntry<TKey\*, TValue>

{

public:

void Clear()

{

\_\_super::Clear();

this->key = nullptr;

}

};

template <class TValue>

class KeyValueEntry<int, TValue> : public BaseKeyValueEntry<int, TValue>

{

public:

void Clear()

{

\_\_super::Clear();

this->key = 0;

}

};

template <class TKey, class TValue, template <class K, class V> class THashEntry>

class DefaultHashedEntry : public THashEntry<TKey, TValue>

{

public:

template<typename Comparer, typename TLookup>

\_\_inline bool KeyEquals(TLookup const& otherKey, hash\_t otherHashCode)

{

return Comparer::Equals(Key(), otherKey);

}

template<typename Comparer>

\_\_inline hash\_t GetHashCode()

{

return ((Comparer::GetHashCode(Key()) & 0x7fffffff) << 1) | 1;

}

void Set(TKey const& key, TValue const& value, int hashCode)

{

\_\_super::Set(key, value);

}

};

template <class TKey, class TValue, template <class K, class V> class THashEntry>

class CacheHashedEntry : public THashEntry<TKey, TValue>

{

hash\_t hashCode; // Lower 31 bits of hash code << 1 | 1, 0 if unused

public:

static const int INVALID\_HASH\_VALUE = 0;

template<typename Comparer, typename TLookup>

\_\_inline bool KeyEquals(TLookup const& otherKey, hash\_t otherHashCode)

{

Assert(TAGHASH(Comparer::GetHashCode(Key())) == this->hashCode);

return this->hashCode == otherHashCode && Comparer::Equals(Key(), otherKey);

}

template<typename Comparer>

\_\_inline hash\_t GetHashCode()

{

Assert(TAGHASH(Comparer::GetHashCode(Key())) == this->hashCode);

return hashCode;

}

void Set(TKey const& key, TValue const& value, hash\_t hashCode)

{

\_\_super::Set(key, value);

this->hashCode = hashCode;

}

void Clear()

{

\_\_super::Clear();

this->hashCode = INVALID\_HASH\_VALUE;

}

};

template <class TKey, class TValue>

class SimpleHashedEntry : public DefaultHashedEntry<TKey, TValue, ImplicitKeyValueEntry> {};

template <class TKey, class TValue>

class HashedEntry : public CacheHashedEntry<TKey, TValue, ImplicitKeyValueEntry> {};

template <class TKey, class TValue>

class SimpleDictionaryEntry : public DefaultHashedEntry<TKey, TValue, KeyValueEntry> {};

template <class TKey, class TValue>

class DictionaryEntry: public CacheHashedEntry<TKey, TValue, KeyValueEntry> {};

template <class TKey, class TValue>

class WeakRefValueDictionaryEntry: public SimpleDictionaryEntry<TKey, TValue>

{

public:

void Clear()

{

// Assuming nullable keys for now

// This might change in future

this->key = NULL;

this->value = NULL;

}

static bool SupportsCleanup()

{

return true;

}

static bool NeedsCleanup(WeakRefValueDictionaryEntry<TKey, TValue> const& entry)

{

TValue weakReference = entry.Value();

return (weakReference == nullptr || weakReference->Get() == nullptr);

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

#if PROFILE\_DICTIONARY

#include "DictionaryStats.h"

DictionaryType\* DictionaryStats::dictionaryTypes = NULL;

CRITICAL\_SECTION DictionaryStats::dictionaryTypesCriticalSection;

DictionaryStats\* DictionaryStats::Create(const char\* name, uint bucketCount)

{

if (!Js::Configuration::Global.flags.IsEnabled(Js::ProfileDictionaryFlag) ||

Js::Configuration::Global.flags.ProfileDictionary < 0)

return NULL;

return ::new DictionaryStats(name, bucketCount);

}

DictionaryStats\* DictionaryStats::Clone()

{

DictionaryStats\* cloned = NoCheckHeapNew(DictionaryStats, pName, initialSize);

cloned->finalSize = finalSize;

cloned->countOfEmptyBuckets = countOfEmptyBuckets;

cloned->countOfResize = countOfResize;

cloned->itemCount = itemCount;

cloned->maxDepth = maxDepth;

cloned->lookupCount = lookupCount;

cloned->collisionCount = collisionCount;

cloned->lookupDepthTotal = lookupDepthTotal;

cloned->maxLookupDepth = maxLookupDepth;

cloned->pName = pName;

return cloned;

}

DictionaryStats::DictionaryStats(const char\* name, uint bucketCount)

:

initialSize(bucketCount),

finalSize(bucketCount),

countOfEmptyBuckets(bucketCount),

countOfResize(0),

itemCount(0),

maxDepth(0),

lookupCount(0),

collisionCount(0),

lookupDepthTotal(0),

maxLookupDepth(0),

pNext(NULL),

pName(NULL)

{

if(dictionaryTypes == NULL)

{

InitializeCriticalSection(&DictionaryStats::dictionaryTypesCriticalSection);

}

EnterCriticalSection(&DictionaryStats::dictionaryTypesCriticalSection);

DictionaryType\* type = NULL;

// See if we already created instance(s) of this type

DictionaryType\* current = dictionaryTypes;

while(current)

{

if (strncmp(name, current->name, \_countof(current->name)-1) == 0)

{

type = current;

break;

}

current = current->pNext;

}

if (!type)

{

// We haven't seen this type before so add a new entry for it

type = NoCheckHeapNew(DictionaryType);

type->pNext = dictionaryTypes;

dictionaryTypes = type;

type->instancesCount = 0;

strncpy\_s(type->name, name, \_countof(type->name)-1);

type->name[sizeof(type->name)-1]='\0';

}

LeaveCriticalSection(&dictionaryTypesCriticalSection);

// keep a pointer to the name in case we are asked to clone ourselves

pName = type->name;

// Add ourself in the list

pNext = type->instances;

type->instances = this;

++(type->instancesCount);

}

void DictionaryStats::Resize(uint newSize, uint emptyBucketCount)

{

finalSize = newSize;

countOfEmptyBuckets = emptyBucketCount;

++countOfResize;

}

void DictionaryStats::Insert(uint depth)

{

++itemCount;

if (maxDepth < depth)

maxDepth = depth;

if (depth == 1 && countOfEmptyBuckets > 0)

--countOfEmptyBuckets;

}

void DictionaryStats::Remove(bool isBucketEmpty)

{

if (itemCount > 0)

--itemCount;

if (isBucketEmpty)

++countOfEmptyBuckets;

}

void DictionaryStats::Lookup(uint depth)

{

// Note, lookup and collision math only works out if depth is 0-based.

// I.e., depth of 1 means there was 1 collision and the lookup found key at second item in the bucket

lookupCount += 1;

lookupDepthTotal += depth;

if (depth > 0)

collisionCount += 1;

if (maxLookupDepth < depth)

maxLookupDepth = depth;

}

void DictionaryStats::OutputStats()

{

if (!dictionaryTypes)

return;

EnterCriticalSection(&DictionaryStats::dictionaryTypesCriticalSection);

DictionaryType\* current = dictionaryTypes;

Output::Print(L"PROFILE DICTIONARY\n");

Output::Print(L"%8s %13s %13s %13s %13s %13s %13s %13s %14s %14s %13s %13s %13s %s\n", L"Metric",L"StartSize", L"EndSize", L"Resizes", L"Items", L"MaxDepth", L"EmptyBuckets", L"Lookups", L"Collisions", L"AvgLookupDepth", L"AvgCollDepth", L"MaxLookupDepth", L"Instances", L"Type");

while(current)

{

DictionaryType \*type = current;

DictionaryStats \*instance = type->instances;

double size = 0, max\_size = 0;

double endSize = 0, max\_endSize = 0;

double resizes = 0, max\_resizes = 0;

double items = 0, max\_items = 0;

double depth = 0, max\_depth = 0;

double empty = 0, max\_empty = 0;

double lookups = 0, max\_lookups = 0;

double collisions = 0, max\_collisions = 0;

double avglookupdepth = 0, max\_avglookupdepth = 0;

double avgcollisiondepth = 0, max\_avgcollisiondepth = 0;

double maxlookupdepth = 0, max\_maxlookupdepth = 0;

bool dumpInstances = false;

//if(strstr(type->name, "SimpleDictionaryPropertyDescriptor") != nullptr)

//{

// dumpInstances = true;

//}

while(instance)

{

ComputeStats(instance->initialSize, size, max\_size);

ComputeStats(instance->finalSize, endSize, max\_endSize);

ComputeStats(instance->countOfResize, resizes, max\_resizes);

ComputeStats(instance->itemCount, items, max\_items);

ComputeStats(instance->maxDepth, depth, max\_depth);

ComputeStats(instance->countOfEmptyBuckets, empty, max\_empty);

ComputeStats(instance->lookupCount, lookups, max\_lookups);

ComputeStats(instance->collisionCount, collisions, max\_collisions);

if (instance->lookupCount > 0)

{

ComputeStats((double)instance->lookupDepthTotal / (double)instance->lookupCount, avglookupdepth, max\_avglookupdepth);

}

if (instance->collisionCount > 0)

{

ComputeStats((double)instance->lookupDepthTotal / (double)instance->collisionCount, avgcollisiondepth, max\_avgcollisiondepth);

}

ComputeStats(instance->maxLookupDepth, maxlookupdepth, max\_maxlookupdepth);

if(dumpInstances)

{

double avgld = 0.0;

double avgcd = 0.0;

if (instance->lookupCount > 0)

{

avgld = (double)instance->lookupDepthTotal / (double)instance->lookupCount;

avgcd = (double)instance->lookupDepthTotal / (double)instance->collisionCount;

}

Output::Print(L"%8s %13d %13d %13d %13d %13d %13d %13d %14d %14.2f %13.2f %13d \n",

L"INS:",

instance->initialSize, instance->finalSize, instance->countOfResize,

instance->itemCount, instance->maxDepth, instance->countOfEmptyBuckets,

instance->lookupCount, instance->collisionCount, avgld, avgcd,

instance->maxLookupDepth);

}

instance = instance->pNext;

}

if (max\_depth >= Js::Configuration::Global.flags.ProfileDictionary)

{

Output::Print(L"%8s %13.0f %13.0f %13.2f %13.0f %13.2f %13.0f %13.0f %14.0f %14.2f %13.2f %13.2f %13d %S\n", L"AVG:",

size/type->instancesCount, endSize/type->instancesCount, resizes/type->instancesCount, items/type->instancesCount,

depth/type->instancesCount, empty/type->instancesCount, lookups/type->instancesCount, collisions/type->instancesCount,

avglookupdepth/type->instancesCount, avgcollisiondepth/type->instancesCount, maxlookupdepth/type->instancesCount, type->instancesCount, type->name);

Output::Print(L"%8s %13.0f %13.0f %13.2f %13.0f %13.2f %13.0f %13.0f %14.0f %14.2f %13.2f %13.2f %13d %S\n\n", L"MAX:",

max\_size, max\_endSize, max\_resizes, max\_items, max\_depth, max\_empty, max\_lookups, max\_collisions, max\_avglookupdepth,

max\_avgcollisiondepth, max\_maxlookupdepth, type->instancesCount, type->name);

}

current = current->pNext;

}

Output::Print(L"====================================================================================\n");

ClearStats();

LeaveCriticalSection(&DictionaryStats::dictionaryTypesCriticalSection);

DeleteCriticalSection(&DictionaryStats::dictionaryTypesCriticalSection);

}

void DictionaryStats::ComputeStats(uint input, double &total, double &max)

{

total += input;

if (input > max)

max = input;

}

void DictionaryStats::ComputeStats(double input, double &total, double &max)

{

total += input;

if (input > max)

max = input;

}

void DictionaryStats::ClearStats()

{

// Clear the collection since we already reported on what we already collected

DictionaryType\* current = dictionaryTypes;

while(current)

{

DictionaryType \*type = current;

DictionaryStats \*pNext = type->instances;

while(pNext)

{

DictionaryStats \*pCurrent = pNext;

pNext = pNext->pNext;

delete pCurrent;

}

current = current->pNext;

delete type;

}

dictionaryTypes = NULL;

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

//----------------------------------------------------------------------------

//

// File: DList.h

//

// Template for Doubly Linked List

//----------------------------------------------------------------------------

template <typename TData, typename TCount = DefaultCount> class DListBase;

template <typename TData> class DListNode;

template <typename TData>

class DListNodeBase

{

public:

DListNodeBase<TData> \* Next() const { return next.base; }

DListNodeBase<TData> \*& Next() { return next.base; }

DListNodeBase<TData> \* Prev() const { return prev.base; }

DListNodeBase<TData> \*& Prev() { return prev.base; }

private:

// The next node can be a real node with data, or it point back to the start of the list

// Use a union to show it in the debugger (instead of casting everywhere)

union

{

DListNodeBase<TData> \* base;

DListNode<TData> \* node;

DListBase<TData> \* list;

} next;

union

{

DListNodeBase<TData> \* base;

DListNode<TData> \* node;

DListBase<TData> \* list;

} prev;

};

template <typename TData>

class DListNode : public DListNodeBase<TData>

{

public:

DListNode() : data() {}

// Constructing with parameter

template <typename TParam1>

DListNode(TParam1 param1) : data(param1) {}

// Constructing with parameter

template <typename TParam1, typename TParam2>

DListNode(TParam1 param1, TParam2 param2) : data(param1, param2) {}

// Constructing with parameter

template <typename TParam1, typename TParam2, typename TParam3>

DListNode(TParam1 param1, TParam2 param2, TParam3 param3) : data(param1, param2, param3) {}

// Constructing with parameter

template <typename TParam1, typename TParam2, typename TParam3, typename TParam4>

DListNode(TParam1 param1, TParam2 param2, TParam3 param3, TParam4 param4) : data(param1, param2, param3, param4) {}

// Constructing using copy constructor

DListNode(TData const& data) : data(data) {};

TData data;

};

template<typename TData, typename TCount>

class DListBase : protected DListNodeBase<TData>, public TCount

{

private:

typedef DListNodeBase<TData> NodeBase;

typedef DListNode<TData> Node;

bool IsHead(NodeBase const \* node) const

{

return (node == this);

}

public:

class Iterator

{

public:

Iterator() : list(nullptr), current(nullptr) {}

Iterator(DListBase const \* list) : list(list), current(list) {};

bool IsValid() const

{

return (current != nullptr && !list->IsHead(current));

}

void Reset()

{

current = list;

}

// TODO: only need inline for DListBase<Segment, FakeCount>::Iterator::Next

\_\_forceinline

bool Next()

{

Assert(current != nullptr);

if (list->IsHead(current->Next()))

{

current = nullptr;

return false;

}

current = current->Next();

return true;

}

TData const& Data() const

{

Assert(this->IsValid());

return ((Node \*)current)->data;

}

TData& Data()

{

Assert(this->IsValid());

return ((Node \*)current)->data;

}

protected:

DListBase const \* list;

NodeBase const \* current;

};

class EditingIterator : public Iterator

{

public:

EditingIterator() : Iterator() {};

EditingIterator(DListBase \* list) : Iterator(list) {};

template <typename TAllocator>

void RemoveCurrent(TAllocator \* allocator)

{

Assert(current != nullptr);

Assert(!list->IsHead(current));

NodeBase \* last = current->Prev();

NodeBase \* node = const\_cast<NodeBase \*>(current);

DListBase::RemoveNode(node);

AllocatorDelete(TAllocator, allocator, (Node \*)node);

current = last;

const\_cast<DListBase \*>(list)->DecrementCount();

}

template <typename TAllocator>

TData \* InsertNodeBefore(TAllocator \* allocator)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node);

if (newNode)

{

NodeBase \* node = const\_cast<NodeBase \*>(current);

DListBase::InsertNodeBefore(node, newNode);

const\_cast<DListBase \*>(list)->IncrementCount();

return newNode->data;

}

}

template <typename TAllocator>

bool InsertBefore(TAllocator \* allocator, TData const& data)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, data);

if (newNode)

{

NodeBase \* node = const\_cast<NodeBase \*>(current);

DListBase::InsertNodeBefore(node, newNode);

const\_cast<DListBase \*>(list)->IncrementCount();

return true;

}

return false;

}

void MoveCurrentTo(DListBase \* toList)

{

NodeBase \* last = current->Prev();

NodeBase \* node = const\_cast<NodeBase \*>(current);

DListBase::RemoveNode(node);

DListBase::InsertNodeBefore(toList->Next(), node);

current = last;

const\_cast<DListBase \*>(list)->DecrementCount();

toList->IncrementCount();

}

};

explicit DListBase()

{

Reset();

}

~DListBase()

{

AssertMsg(this->Empty(), "DListBase need to be cleared explicitly with an allocator");

}

void Reset()

{

this->Next() = this;

this->Prev() = this;

this->SetCount(0);

}

template <typename TAllocator>

void Clear(TAllocator \* allocator)

{

NodeBase \* current = this->Next();

while (!this->IsHead(current))

{

NodeBase \* next = current->Next();

AllocatorDelete(TAllocator, allocator, (Node \*)current);

current = next;

}

this->Next() = this;

this->Prev() = this;

this->SetCount(0);

}

bool Empty() const { return this->IsHead(this->Next()); }

bool HasOne() const { return !Empty() && this->IsHead(this->Next()->Next()); }

TData const& Head() const { Assert(!Empty()); return ((Node \*)this->Next())->data; }

TData& Head() { Assert(!Empty()); return ((Node \*)this->Next())->data; }

TData const& Tail() const { Assert(!Empty()); return ((Node \*)this->Prev())->data; }

TData & Tail() { Assert(!Empty()); return ((Node \*)this->Prev())->data; }

template <typename TAllocator>

bool Append(TAllocator \* allocator, TData const& data)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, data);

if (newNode)

{

DListBase::InsertNodeAfter(this->Prev(), newNode);

this->IncrementCount();

return true;

}

return false;

}

template <typename TAllocator>

bool Prepend(TAllocator \* allocator, TData const& data)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, data);

if (newNode)

{

DListBase::InsertNodeBefore(this->Next(), newNode);

this->IncrementCount();

return true;

}

return false;

}

template <typename TAllocator>

TData \* PrependNode(TAllocator \* allocator)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node);

if (newNode)

{

DListBase::InsertNodeBefore(this->Next(), newNode);

this->IncrementCount();

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator, typename TParam1>

TData \* PrependNode(TAllocator \* allocator, TParam1 param1)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, param1);

if (newNode)

{

DListBase::InsertNodeBefore(this->Next(), newNode);

this->IncrementCount();

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator, typename TParam1, typename TParam2>

TData \* PrependNode(TAllocator \* allocator, TParam1 param1, TParam2 param2)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, param1, param2);

if (newNode)

{

DListBase::InsertNodeBefore(this->Next(), newNode);

this->IncrementCount();

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator, typename TParam1, typename TParam2, typename TParam3>

TData \* PrependNode(TAllocator \* allocator, TParam1 param1, TParam2 param2, TParam3 param3)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, param1, param2, param3);

if (newNode)

{

DListBase::InsertNodeBefore(this->Next(), newNode);

this->IncrementCount();

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator, typename TParam1, typename TParam2, typename TParam3, typename TParam4>

TData \* PrependNode(TAllocator \* allocator, TParam1 param1, TParam2 param2, TParam3 param3, TParam4 param4)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, param1, param2, param3, param4);

if (newNode)

{

DListBase::InsertNodeBefore(this->Next(), newNode);

this->IncrementCount();

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator>

void RemoveHead(TAllocator \* allocator)

{

Assert(!this->Empty());

NodeBase \* node = this->Next();

DListBase::RemoveNode(node);

AllocatorDelete(TAllocator, allocator, (Node \*)node);

this->DecrementCount();

}

template <typename TAllocator>

bool Remove(TAllocator \* allocator, TData const& data)

{

EditingIterator iter(this);

while (iter.Next())

{

if (iter.Data() == data)

{

iter.RemoveCurrent(allocator);

return true;

}

}

return false;

}

template <typename TAllocator>

void RemoveElement(TAllocator \* allocator, TData \* element)

{

Node \* node = CONTAINING\_RECORD(element, Node, data);

#if DBG\_DUMP

Assert(HasNode(node));

#endif

DListBase::RemoveNode(node);

AllocatorDelete(TAllocator, allocator, node);

this->DecrementCount();

}

bool Has(TData data) const

{

Iterator iter(this);

while (iter.Next())

{

if (iter.Data() == data)

{

return true;

}

}

return false;

}

void MoveTo(DListBase \* list)

{

list->Prev()->Next() = this->Next();

this->Next()->Prev() = list->Prev();

list->Prev() = this->Prev();

this->Prev()->Next() = list;

this->Prev() = this;

this->Next() = this;

list->AddCount(\*this);

this->SetCount(0);

}

void MoveHeadTo(DListBase \* list)

{

Assert(!this->Empty());

NodeBase \* node = this->Next();

DListBase::RemoveNode(node);

DListBase::InsertNodeBefore(list->Next(), node);

this->DecrementCount();

list->IncrementCount();

}

void MoveElementTo(TData \* element, DListBase \* list)

{

Node \* node = CONTAINING\_RECORD(element, Node, data);

#if DBG\_DUMP

Assert(HasNode(node));

#endif

DListBase::RemoveNode(node);

DListBase::InsertNodeBefore(list->Next(), node);

this->DecrementCount();

list->IncrementCount();

}

#if DBG\_DUMP

bool HasElement(TData const \* element) const

{

Node \* node = CONTAINING\_RECORD(element, Node, data);

return HasNode(node);

}

#endif

private:

#if DBG\_DUMP

bool HasNode(NodeBase \* node) const

{

NodeBase \* current = this->Next();

while (!this->IsHead(current))

{

if (node == current)

{

return true;

}

current = current->Next();

}

return false;

}

#endif

// disable copy constructor

DListBase(DListBase const& list);

static void InsertNodeAfter(NodeBase \* node, NodeBase \* newNode)

{

newNode->Prev() = node;

newNode->Next() = node->Next();

node->Next()->Prev() = newNode;

node->Next() = newNode;

}

static void InsertNodeBefore(NodeBase \* node, NodeBase \* newNode)

{

newNode->Prev() = node->Prev();

newNode->Next() = node;

node->Prev()->Next() = newNode;

node->Prev() = newNode;

}

static void RemoveNode(NodeBase \* node)

{

node->Prev()->Next() = node->Next();

node->Next()->Prev() = node->Prev();

}

};

#define FOREACH\_DLISTBASE\_ENTRY(T, data, list) \

{ \

DListBase<T>::Iterator \_\_iter(list); \

while (\_\_iter.Next()) \

{ \

T& data = \_\_iter.Data();

#define NEXT\_DLISTBASE\_ENTRY \

} \

}

#define FOREACH\_DLISTBASE\_ENTRY\_EDITING(T, data, list, iter) \

DListBase<T>::EditingIterator iter(list); \

while (iter.Next()) \

{ \

T& data = iter.Data();

#define NEXT\_DLISTBASE\_ENTRY\_EDITING \

}

template <typename TData, typename TAllocator, typename TCount = DefaultCount>

class DList : public DListBase<TData, TCount>

{

public:

class EditingIterator : public DListBase::EditingIterator

{

public:

EditingIterator() : DListBase::EditingIterator() {}

EditingIterator(DList \* list) : DListBase::EditingIterator(list) {}

void RemoveCurrent()

{

\_\_super::RemoveCurrent(Allocator());

}

TData& InsertNodeBefore()

{

return \_\_super::InsertNodeBefore(Allocator());

}

void InsertBefore(TData const& data)

{

\_\_super::InsertBefore(Allocator(), data);

}

private:

TAllocator \* Allocator() const

{

return ((DList const \*)list)->allocator;

}

};

explicit DList(TAllocator \* allocator) : allocator(allocator) {}

~DList()

{

Clear();

}

void Clear()

{

\_\_super::Clear(allocator);

}

bool Append(TData const& data)

{

return \_\_super::Append(allocator, data);

}

bool Prepend(TData const& data)

{

return \_\_super::Prepend(allocator, data);

}

TData \* PrependNode()

{

return \_\_super::PrependNode(allocator);

}

template <typename TParam1>

TData \* PrependNode(TParam1 param1)

{

return \_\_super::PrependNode(allocator, param1);

}

template <typename TParam1, typename TParam2>

TData \* PrependNode(TParam1 param1, TParam2 param2)

{

return \_\_super::PrependNode(allocator, param1, param2);

}

template <typename TParam1, typename TParam2, typename TParam3>

TData \* PrependNode(TParam1 param1, TParam2 param2, TParam3 param3)

{

return \_\_super::PrependNode(allocator, param1, param2, param3);

}

template <typename TParam1, typename TParam2, typename TParam3, typename TParam4>

TData \* PrependNode(TParam1 param1, TParam2 param2, TParam3 param3, TParam4 param4)

{

return \_\_super::PrependNode(allocator, param1, param2, param3, param4);

}

void RemoveHead()

{

\_\_super::RemoveHead(allocator);

}

bool Remove(TData const& data)

{

return \_\_super::Remove(allocator, data);

}

void RemoveElement(TData \* data)

{

return \_\_super::RemoveElement(allocator, data);

}

private:

TAllocator \* allocator;

};

template <typename TData, typename TAllocator = ArenaAllocator>

class DListCounted : public DList<TData, TAllocator, RealCount>

{

public:

explicit DListCounted(TAllocator \* allocator) : DList(allocator) {}

};

#define FOREACH\_DLIST\_ENTRY(T, alloc, data, list) \

{ \

DList<T, alloc>::Iterator \_\_iter(list); \

while (\_\_iter.Next()) \

{ \

T& data = \_\_iter.Data();

#define NEXT\_DLIST\_ENTRY \

} \

}

#define FOREACH\_DLIST\_ENTRY\_EDITING(T, alloc, data, list, iter) \

DList<T, alloc>::EditingIterator iter(list); \

while (iter.Next()) \

{ \

T& data = iter.Data();

#define NEXT\_DLIST\_ENTRY\_EDITING \

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template<class T>

class DoublyLinkedList

{

private:

T \*head, \*tail;

public:

DoublyLinkedList();

public:

T \*Head() const;

T \*Tail() const;

public:

bool Contains(T \*const element) const;

bool ContainsSubsequence(T \*const first, T \*const last) const;

public:

bool IsEmpty();

void Clear();

void LinkToBeginning(T \*const element);

void LinkToEnd(T \*const element);

void LinkBefore(T \*const element, T \*const nextElement);

void LinkAfter(T \*const element, T \*const previousElement);

T \*UnlinkFromBeginning();

T \*UnlinkFromEnd();

void UnlinkPartial(T \*const element);

void Unlink(T \*const element);

void MoveToBeginning(T \*const element);

void UnlinkSubsequenceFromEnd(T \*const first);

void UnlinkSubsequence(T \*const first, T \*const last);

void MoveSubsequenceToBeginning(T \*const first, T \*const last);

// JScriptDiag doesn't seem to like the PREVENT\_COPY macro

private:

DoublyLinkedList(const DoublyLinkedList &other);

DoublyLinkedList &operator =(const DoublyLinkedList &other);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template<class T>

DoublyLinkedList<T>::DoublyLinkedList() : head(nullptr), tail(nullptr)

{

}

template<class T>

T \*DoublyLinkedList<T>::Head() const

{

return head;

}

template<class T>

T \*DoublyLinkedList<T>::Tail() const

{

return tail;

}

template<class T>

bool DoublyLinkedList<T>::Contains(T \*const element) const

{

return T::Contains(element, head);

}

template<class T>

bool DoublyLinkedList<T>::ContainsSubsequence(T \*const first, T \*const last) const

{

return T::ContainsSubsequence(first, last, head);

}

template<class T>

bool DoublyLinkedList<T>::IsEmpty()

{

return head == nullptr;

}

template<class T>

void DoublyLinkedList<T>::Clear()

{

tail = head = nullptr;

}

template<class T>

void DoublyLinkedList<T>::LinkToBeginning(T \*const element)

{

T::LinkToBeginning(element, &head, &tail);

}

template<class T>

void DoublyLinkedList<T>::LinkToEnd(T \*const element)

{

T::LinkToEnd(element, &head, &tail);

}

template<class T>

void DoublyLinkedList<T>::LinkBefore(T \*const element, T \*const nextElement)

{

T::LinkBefore(element, nextElement, &head, &tail);

}

template<class T>

void DoublyLinkedList<T>::LinkAfter(T \*const element, T \*const previousElement)

{

T::LinkAfter(element, previousElement, &head, &tail);

}

template<class T>

T \*DoublyLinkedList<T>::UnlinkFromBeginning()

{

T \*const element = head;

if(element)

T::UnlinkFromBeginning(element, &head, &tail);

return element;

}

template<class T>

T \*DoublyLinkedList<T>::UnlinkFromEnd()

{

T \*const element = tail;

if(element)

T::UnlinkFromEnd(element, &head, &tail);

return element;

}

template<class T>

void DoublyLinkedList<T>::UnlinkPartial(T \*const element)

{

T::UnlinkPartial(element, &head, &tail);

}

template<class T>

void DoublyLinkedList<T>::Unlink(T \*const element)

{

T::Unlink(element, &head, &tail);

}

template<class T>

void DoublyLinkedList<T>::MoveToBeginning(T \*const element)

{

T::MoveToBeginning(element, &head, &tail);

}

template<class T>

void DoublyLinkedList<T>::UnlinkSubsequenceFromEnd(T \*const first)

{

T::UnlinkSubsequenceFromEnd(first, &head, &tail);

}

template<class T>

void DoublyLinkedList<T>::UnlinkSubsequence(T \*const first, T \*const last)

{

T::UnlinkSubsequence(first, last, &head, &tail);

}

template<class T>

void DoublyLinkedList<T>::MoveSubsequenceToBeginning(T \*const first, T \*const last)

{

T::MoveSubsequenceToBeginning(first, last, &head, &tail);

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template<class T>

class DoublyLinkedListElement

{

private:

T \*previous, \*next;

public:

DoublyLinkedListElement();

public:

T \*Previous() const;

T \*Next() const;

template<class D> static bool Contains(D \*const element, D \*const head);

template<class D> static bool ContainsSubsequence(D \*const first, D \*const last, D \*const head);

template<class D> static void LinkToBeginning(D \*const element, D \* \*const head, D \* \*const tail);

template<class D> static void LinkToEnd(D \*const element, D \* \*const head, D \* \*const tail);

template<class D> static void LinkBefore(D \*const element, D \*const nextElement, D \* \*const head, D \* \*const tail);

template<class D> static void LinkAfter(D \*const element, D \*const previousElement, D \* \*const head, D \* \*const tail);

template<class D> static void UnlinkFromBeginning(D \*const element, D \* \*const head, D \* \*const tail);

template<class D> static void UnlinkFromEnd(D \*const element, D \* \*const head, D \* \*const tail);

template<class D> static void UnlinkPartial(D \*const element, D \* \*const head, D \* \*const tail);

template<class D> static void Unlink(D \*const element, D \* \*const head, D \* \*const tail);

template<class D> static void MoveToBeginning(D \*const element, D \* \*const head, D \* \*const tail);

template<class D> static void UnlinkSubsequenceFromEnd(D \*const first, D \* \*const head, D \* \*const tail);

template<class D> static void UnlinkSubsequence(D \*const first, D \*const last, D \* \*const head, D \* \*const tail);

template<class D> static void MoveSubsequenceToBeginning(D \*const first, D \*const last, D \* \*const head, D \* \*const tail);

// JScriptDiag doesn't seem to like the PREVENT\_COPY macro

private:

DoublyLinkedListElement(const DoublyLinkedListElement &other);

DoublyLinkedListElement &operator =(const DoublyLinkedListElement &other);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template<class T>

DoublyLinkedListElement<T>::DoublyLinkedListElement() : previous(0), next(0)

{

TemplateParameter::SameOrDerivedFrom<T, DoublyLinkedListElement<T>>();

}

template<class T>

T \*DoublyLinkedListElement<T>::Previous() const

{

return previous;

}

template<class T>

T \*DoublyLinkedListElement<T>::Next() const

{

return next;

}

template<class T>

template<class D>

bool DoublyLinkedListElement<T>::Contains(D \*const element, D \*const head)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(element);

Assert(!head || !head->previous);

if(!element->previous && !element->next)

return element == head;

for(T \*e = head; e; e = e->next)

{

if(e == element)

return true;

}

return false;

}

template<class T>

template<class D>

bool DoublyLinkedListElement<T>::ContainsSubsequence(D \*const first, D \*const last, D \*const head)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(first);

Assert(last);

Assert(!head || !head->previous);

if(first == last && !first->previous && !first->next)

return first == head;

bool foundFirst = false;

for(T \*e = head; e; e = e->next)

{

if(e == first)

foundFirst = true;

if(e == last)

return foundFirst;

}

return false;

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::LinkToBeginning(D \*const element, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(element);

Assert(head);

Assert(tail);

Assert(!\*head || !(\*head)->previous);

Assert(!\*tail || !(\*tail)->next);

Assert(!element->previous);

Assert(!element->next);

Assert(!Contains(element, \*head));

element->previous = 0;

element->next = \*head;

\*head = element;

if(element->next)

element->next->previous = element;

else

{

Assert(!\*tail);

\*tail = element;

}

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::LinkToEnd(D \*const element, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(element);

Assert(head);

Assert(tail);

Assert(!\*head || !(\*head)->previous);

Assert(!\*tail || !(\*tail)->next);

Assert(!element->previous);

Assert(!element->next);

Assert(!Contains(element, \*head));

element->previous = \*tail;

element->next = 0;

\*tail = element;

if(element->previous)

element->previous->next = element;

else

{

Assert(!\*head);

\*head = element;

}

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::LinkBefore(D \*const element, D \*const nextElement, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(element);

Assert(nextElement);

Assert(element != nextElement);

Assert(head);

Assert(tail);

Assert(\*head);

Assert(\*tail);

Assert(!(\*head)->previous);

Assert(!(\*tail)->next);

Assert(!element->previous);

Assert(!element->next);

Assert(Contains(nextElement, \*head));

element->next = nextElement;

T \*const previousElement = nextElement->previous;

element->previous = previousElement;

nextElement->previous = element;

if(previousElement)

previousElement->next = element;

else

{

Assert(\*head == nextElement);

\*head = element;

}

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::LinkAfter(D \*const element, D \*const previousElement, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(element);

Assert(previousElement);

Assert(element != previousElement);

Assert(head);

Assert(tail);

Assert(\*head);

Assert(\*tail);

Assert(!(\*head)->previous);

Assert(!(\*tail)->next);

Assert(!element->previous);

Assert(!element->next);

Assert(Contains(previousElement, \*head));

element->previous = previousElement;

T \*const nextElement = previousElement->next;

element->next = nextElement;

previousElement->next = element;

if(nextElement)

nextElement->previous = element;

else

{

Assert(\*tail == previousElement);

\*tail = element;

}

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::UnlinkFromBeginning(D \*const element, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(element);

Assert(head);

Assert(tail);

Assert(\*head);

Assert(\*tail);

Assert(!(\*head)->previous);

Assert(!(\*tail)->next);

Assert(!element->previous);

Assert(\*head == element);

\*head = static\_cast<D \*>(element->next);

if(element->next)

{

element->next->previous = 0;

element->next = 0;

}

else

{

Assert(\*tail == element);

\*tail = 0;

}

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::UnlinkFromEnd(D \*const element, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(element);

Assert(head);

Assert(tail);

Assert(\*head);

Assert(\*tail);

Assert(!(\*head)->previous);

Assert(!(\*tail)->next);

Assert(!element->next);

Assert(\*tail == element);

\*tail = static\_cast<D \*>(element->previous);

if(element->previous)

{

element->previous->next = 0;

element->previous = 0;

}

else

{

Assert(\*head == element);

\*head = 0;

}

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::UnlinkPartial(D \*const element, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(element);

Assert(head);

Assert(tail);

Assert(\*head);

Assert(\*tail);

Assert(!(\*head)->previous);

Assert(!(\*tail)->next);

Assert(Contains(element, \*head));

if(element->previous)

element->previous->next = element->next;

else

{

Assert(\*head == element);

\*head = static\_cast<D \*>(element->next);

}

if(element->next)

element->next->previous = element->previous;

else

{

Assert(\*tail == element);

\*tail = static\_cast<D \*>(element->previous);

}

// Partial unlink does not zero the previous and next links of the unlinked element so that the linked list can be

// iterated on a separate thread while unlinking, without missing elements that are in the linked list before and after

// this unlink

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::Unlink(D \*const element, D \* \*const head, D \* \*const tail)

{

UnlinkPartial(element, head, tail);

element->previous = 0;

element->next = 0;

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::MoveToBeginning(D \*const element, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(element);

Assert(head);

Assert(tail);

Assert(\*head);

Assert(\*tail);

Assert(!(\*head)->previous);

Assert(!(\*tail)->next);

Assert(Contains(element, \*head));

if(!element->previous)

{

Assert(\*head == element);

return;

}

element->previous->next = element->next;

if(element->next)

element->next->previous = element->previous;

else

{

Assert(\*tail == element);

\*tail = static\_cast<D \*>(element->previous);

}

element->previous = 0;

element->next = \*head;

\*head = element;

element->next->previous = element;

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::UnlinkSubsequenceFromEnd(D \*const first, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(first);

Assert(head);

Assert(tail);

Assert(\*head);

Assert(\*tail);

Assert(!(\*head)->previous);

Assert(!(\*tail)->next);

Assert(Contains(first, \*head));

if(first->previous)

first->previous->next = 0;

else

{

Assert(\*head == first);

\*head = 0;

}

\*tail = static\_cast<D \*>(first->previous);

first->previous = 0;

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::UnlinkSubsequence(D \*const first, D \*const last, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(first);

Assert(last);

Assert(head);

Assert(tail);

Assert(\*head);

Assert(\*tail);

Assert(!(\*head)->previous);

Assert(!(\*tail)->next);

Assert(ContainsSubsequence(first, last, \*head));

if(first->previous)

first->previous->next = last->next;

else

{

Assert(\*head == first);

\*head = static\_cast<D \*>(last->next);

}

if(last->next)

last->next->previous = first->previous;

else

{

Assert(\*tail == last);

\*tail = static\_cast<D \*>(first->previous);

}

first->previous = 0;

last->next = 0;

}

template<class T>

template<class D>

void DoublyLinkedListElement<T>::MoveSubsequenceToBeginning(D \*const first, D \*const last, D \* \*const head, D \* \*const tail)

{

TemplateParameter::SameOrDerivedFrom<D, T>();

Assert(first);

Assert(last);

Assert(head);

Assert(tail);

Assert(\*head);

Assert(\*tail);

Assert(!(\*head)->previous);

Assert(!(\*tail)->next);

Assert(ContainsSubsequence(first, last, \*head));

if(!first->previous)

{

Assert(\*head == first);

return;

}

first->previous->next = last->next;

if(last->next)

last->next->previous = first->previous;

else

{

Assert(\*tail == last);

\*tail = static\_cast<D \*>(first->previous);

}

first->previous = 0;

last->next = \*head;

\*head = static\_cast<D \*>(first);

last->next->previous = last;

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

template <bool fastHash>

struct EvalMapStringInternal

{

JsUtil::CharacterBuffer<wchar\_t> str;

hash\_t hash;

ModuleID moduleID;

BOOL strict;

BOOL isLibraryCode;

EvalMapStringInternal() : str(), moduleID(0), strict(FALSE), isLibraryCode(FALSE), hash(0) {};

EvalMapStringInternal(\_\_in\_ecount(charLength) wchar\_t const\* content, int charLength, ModuleID moduleID, BOOL strict, BOOL isLibraryCode)

: str(content, charLength), moduleID(moduleID), strict(strict), isLibraryCode(isLibraryCode)

{

// NOTE: this hash is not equivalent to the character buffer hash

// Don't use a CharacteBuffer to do a map lookup on the EvalMapString.

if (fastHash)

{

hash = TAGHASH(str.FastHash());

}

else

{

hash = TAGHASH((hash\_t)str);

}

};

EvalMapStringInternal& operator=(void \* str)

{

Assert(str == null);

memset(this, 0, sizeof(EvalMapString));

return (\*this);

}

inline ModuleID GetModuleID() const

{

return moduleID;

}

inline BOOL IsStrict() const

{

return strict;

}

// Equality and hash function

bool operator==(EvalMapStringInternal const& other) const

{

return this->str == other.str &&

this->GetModuleID() == other.GetModuleID() &&

this->IsStrict() == other.IsStrict() &&

this->isLibraryCode == other.isLibraryCode;

}

operator hash\_t() const

{

return UNTAGHASH(hash);

}

};

typedef EvalMapStringInternal<true> FastEvalMapString;

typedef EvalMapStringInternal<false> EvalMapString;

void ConvertKey(const FastEvalMapString& src, EvalMapString& dest);

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

BVFixed::BVFixed(BVFixed \* initBv) :

len(initBv->Length())

{

this->Copy(initBv);

}

BVFixed::BVFixed(BVIndex length, bool initialSet) :

len(length)

{

Assert(length != 0);

if (initialSet)

{

this->SetAll();

}

else

{

this->ClearAll();

}

}

size\_t

BVFixed::GetAllocSize(BVIndex length)

{

Assert(length != 0);

return sizeof(BVFixed) + sizeof(BVUnit) \* BVFixed::WordCount(length);

}

void

BVFixed::Init(BVIndex length)

{

Assert(length != 0);

len = length;

}

BVIndex

BVFixed::WordCount(BVIndex length)

{

Assert(length != 0);

return ((length - 1) >> BVUnit::ShiftValue) + 1;

}

BVIndex

BVFixed::WordCount() const

{

return WordCount(Length());

}

const BVUnit \*

BVFixed::BitsFromIndex(BVIndex i) const

{

AssertRange(i);

return &this->data[BVUnit::Position(i)];

}

BVUnit \*

BVFixed::BitsFromIndex(BVIndex i)

{

AssertRange(i);

return &this->data[BVUnit::Position(i)];

}

const BVUnit \*

BVFixed::BeginUnit() const

{

return &this->data[0];

}

const BVUnit \*

BVFixed::EndUnit() const

{

return &this->data[WordCount()];

}

BVUnit \*

BVFixed::BeginUnit()

{

return &this->data[0];

}

BVUnit \*

BVFixed::EndUnit()

{

return &this->data[WordCount()];

}

BVIndex

BVFixed::GetNextBit(BVIndex i) const

{

AssertRange(i);

const BVUnit \* chunk = BitsFromIndex(i);

BVIndex base = BVUnit::Floor(i);

BVIndex offset = chunk->GetNextBit(BVUnit::Offset(i));

if(-1 != offset)

{

return base + offset;

}

while(++chunk != this->EndUnit())

{

base += BVUnit::BitsPerWord;

offset = chunk->GetNextBit();

if(-1 != offset)

{

return base + offset;

}

}

return BVInvalidIndex;

}

void

BVFixed::AssertRange(BVIndex i) const

{

AssertMsg(i < this->Length(), "index out of bound");

}

void

BVFixed::AssertBV(const BVFixed \*bv) const

{

AssertMsg(NULL != bv, "Cannot operate on NULL bitvector");

}

BVIndex

BVFixed::Length() const

{

return this->len;

}

void

BVFixed::SetAll()

{

memset(&this->data[0], -1, WordCount() \* sizeof(BVUnit));

ClearEnd();

}

void

BVFixed::ClearAll()

{

ZeroMemory(&this->data[0], WordCount() \* sizeof(BVUnit));

}

BOOLEAN

BVFixed::TestAndSet(BVIndex i)

{

AssertRange(i);

BVUnit \* bvUnit = this->BitsFromIndex(i);

BVIndex offset = BVUnit::Offset(i);

BOOLEAN bit = bvUnit->Test(offset);

bvUnit->Set(offset);

return bit;

}

BOOLEAN

BVFixed::TestAndClear(BVIndex i)

{

AssertRange(i);

BVUnit \* bvUnit = this->BitsFromIndex(i);

BVIndex offset = BVUnit::Offset(i);

BOOLEAN bit = bvUnit->Test(offset);

bvUnit->Clear(offset);

return bit;

}

BOOLEAN

BVFixed::operator[](BVIndex i) const

{

AssertRange(i);

return this->Test(i);

}

void

BVFixed::Or(const BVFixed\*bv)

{

AssertBV(bv);

this->for\_each(bv, &BVUnit::Or);

}

//

// Xors the two bit vectors and returns the count of bits which are different.

//

uint

BVFixed::DiffCount(const BVFixed\*bv) const

{

const BVUnit \*i, \*j;

uint count = 0;

for(i = this->BeginUnit(), j = bv->BeginUnit();

i != this->EndUnit() && j != bv->EndUnit();

i++, j++)

{

count += i->DiffCount(\*j);

}

// Assumes that the default value of is 0

while(i != this->EndUnit())

{

count += i->Count();

i++;

}

while(j != bv->EndUnit())

{

count += j->Count();

j++;

}

return count;

}

void

BVFixed::OrComplimented(const BVFixed\*bv)

{

AssertBV(bv);

this->for\_each(bv, &BVUnit::OrComplimented);

ClearEnd();

}

void

BVFixed::And(const BVFixed\*bv)

{

AssertBV(bv);

this->for\_each(bv, &BVUnit::And);

}

void

BVFixed::Minus(const BVFixed\*bv)

{

AssertBV(bv);

this->for\_each(bv, &BVUnit::Minus);

}

void

BVFixed::Copy(const BVFixed\*bv)

{

AssertBV(bv);

Assert(len >= bv->len);

#if 1

js\_memcpy\_s(&this->data[0], WordCount() \* sizeof(BVUnit), &bv->data[0], bv->WordCount() \* sizeof(BVUnit));

#else

this->for\_each(bv, &BVUnit::Copy);

#endif

}

void

BVFixed::CopyBits(const BVFixed \* bv, BVIndex i)

{

AssertBV(bv);

BVIndex offset = BVUnit::Offset(i);

BVIndex position = BVUnit::Position(i);

BVIndex len = bv->WordCount() - position;

BVIndex copylen = min(WordCount(), len);

if (offset == 0)

{

js\_memcpy\_s(&this->data[0], copylen \* sizeof(BVUnit), &bv->data[BVUnit::Position(i)], copylen \* sizeof(BVUnit));

}

else

{

BVIndex pos = position;

for (BVIndex j = 0; j < copylen; j++)

{

Assert(pos < bv->WordCount());

this->data[j] = bv->data[pos];

this->data[j].ShiftRight(offset);

pos++;

if (pos >= bv->WordCount())

{

break;

}

BVUnit temp = bv->data[pos];

temp.ShiftLeft(BVUnit::BitsPerWord - offset);

this->data[j].Or(temp);

}

}

#if DBG

for (BVIndex curr = i; curr < i + this->Length(); curr++)

{

Assert(this->Test(curr - i) == bv->Test(curr));

}

#endif

}

void

BVFixed::ComplimentAll()

{

for(BVIndex i=0; i < this->WordCount(); i++)

{

this->data[i].ComplimentAll();

}

ClearEnd();

}

void

BVFixed::ClearEnd()

{

uint offset = BVUnit::Offset(this->Length());

if (offset != 0)

{

Assert((((uint64)1 << BVUnit::Offset(this->Length())) - 1) == BVUnit::GetTopBitsClear(this->Length()));

this->data[this->WordCount() - 1].And(BVUnit::GetTopBitsClear(this->Length()));

}

}

BVIndex

BVFixed::Count() const

{

BVIndex sum = 0;

for (BVIndex i=0; i < this->WordCount(); i++)

{

sum += this->data[i].Count();

}

Assert(sum <= this->Length());

return sum;

}

JsUtil::FBVEnumerator

BVFixed::BeginSetBits()

{

return JsUtil::FBVEnumerator(this->BeginUnit(), this->EndUnit());

}

bool BVFixed::IsAllClear() const

{

bool isClear = true;

for (BVIndex i=0; i < this->WordCount(); i++)

{

isClear = this->data[i].IsEmpty() && isClear;

if(!isClear)

{

break;

}

}

return isClear;

}

#if DBG\_DUMP

void

BVFixed::Dump() const

{

bool hasBits = false;

Output::Print(L"[ ");

for(BVIndex i=0; i < this->WordCount(); i++)

{

hasBits = this->data[i].Dump(i \* BVUnit::BitsPerWord, hasBits);

}

Output::Print(L"]\n");

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#define FOREACH\_BITSET\_IN\_FIXEDBV(index, bv) \

{ \

BVIndex index; \

for(JsUtil::FBVEnumerator \_bvenum = bv->BeginSetBits(); \

!\_bvenum.End(); \

\_bvenum++) \

{ \

index = \_bvenum.GetCurrent(); \

#define NEXT\_BITSET\_IN\_FIXEDBV }}

class BVFixed

{

// Data

protected:

BVIndex len;

BVUnit data[];

private:

BVFixed(BVFixed \* initBv);

BVFixed(BVIndex length, bool initialSet = false);

void ClearEnd();

// Creation Factory

public:

template <typename TAllocator>

static BVFixed \* New(TAllocator\* alloc, BVFixed \* initBv);

template <typename TAllocator>

static BVFixed \* New(BVIndex length, TAllocator\* alloc, bool initialSet = false);

template <typename TAllocator>

static BVFixed \* NewNoThrow(BVIndex length, TAllocator\* alloc, bool initialSet = false);

template <typename TAllocator>

void Delete(TAllocator \* alloc);

// For preallocated memory

static size\_t GetAllocSize(BVIndex length);

void Init(BVIndex length);

// Implementation

protected:

void AssertRange(BVIndex i) const;

void AssertBV(const BVFixed \* bv) const;

static BVIndex WordCount(BVIndex length);

const BVUnit \* BitsFromIndex(BVIndex i) const;

BVUnit \* BitsFromIndex(BVIndex i);

const BVUnit \* BeginUnit() const;

BVUnit \* BeginUnit();

const BVUnit \* EndUnit() const;

BVUnit \* EndUnit();

template<class Fn>

\_\_inline void for\_each(const BVFixed \*bv2, const Fn callback)

{

AssertMsg(this->len == bv2->len, "Fatal: The 2 bitvectors should have had the same length.");

BVUnit \* i;

const BVUnit \* j;

for(i = this->BeginUnit(), j = bv2->BeginUnit();

i != this->EndUnit() ;

i++, j++)

{

(i->\*callback)(\*j);

}

}

// Methods

public:

void Set(BVIndex i)

{

AssertRange(i);

this->BitsFromIndex(i)->Set(BVUnit::Offset(i));

}

void Clear(BVIndex i)

{

AssertRange(i);

this->BitsFromIndex(i)->Clear(BVUnit::Offset(i));

}

void Compliment(BVIndex i)

{

AssertRange(i);

this->BitsFromIndex(i)->Complement(BVUnit::Offset(i));

}

BOOLEAN Test(BVIndex i) const

{

AssertRange(i);

return this->BitsFromIndex(i)->Test(BVUnit::Offset(i));

}

BOOLEAN operator[](BVIndex i) const;

BVIndex GetNextBit(BVIndex i) const;

BOOLEAN TestAndSet(BVIndex i);

BOOLEAN TestAndClear(BVIndex i);

void OrComplimented(const BVFixed \* bv);

void Or(const BVFixed \*bv);

uint DiffCount(const BVFixed\* bv) const;

void And(const BVFixed \*bv);

void Minus(const BVFixed \*bv);

void Copy(const BVFixed \*bv);

void CopyBits(const BVFixed \* bv, BVIndex i);

void ComplimentAll();

void SetAll();

void ClearAll();

BVIndex Count() const;

BVIndex Length() const;

JsUtil::FBVEnumerator BeginSetBits();

BVIndex WordCount() const;

bool IsAllClear() const;

template<typename Container>

Container GetRange(BVIndex start, BVIndex len) const;

template<typename Container>

void SetRange(Container\* value, BVIndex start, BVIndex len);

BVUnit\* GetData() const

{

return (BVUnit\*)data;

}

#if DBG\_DUMP

void Dump() const;

#endif

};

template <typename TAllocator>

BVFixed \* BVFixed::New(TAllocator \* alloc, BVFixed \* initBv)

{

BVIndex length = initBv->Length();

BVFixed \*result = AllocatorNewPlus(TAllocator, alloc, sizeof(BVUnit) \* BVFixed::WordCount(length), BVFixed, initBv);

return result;

}

template <typename TAllocator>

BVFixed \* BVFixed::New(BVIndex length, TAllocator \* alloc, bool initialSet)

{

BVFixed \*result = AllocatorNewPlus(TAllocator, alloc, sizeof(BVUnit) \* BVFixed::WordCount(length), BVFixed, length, initialSet);

return result;

}

template <typename TAllocator>

BVFixed \* BVFixed::NewNoThrow(BVIndex length, TAllocator \* alloc, bool initialSet)

{

BVFixed \*result = AllocatorNewNoThrowPlus(TAllocator, alloc, sizeof(BVUnit) \* BVFixed::WordCount(length), BVFixed, length, initialSet);

return result;

}

template <typename TAllocator>

void BVFixed::Delete(TAllocator \* alloc)

{

AllocatorDeletePlus(TAllocator, alloc, sizeof(BVUnit) \* this->WordCount(), this);

}

template<typename Container>

Container BVFixed::GetRange(BVIndex start, BVIndex len) const

{

AssertRange(start);

if (len == 0)

{

return Container(0);

}

Assert(len <= sizeof(Container) \* MachBits);

AssertMsg(len <= 64, "Currently doesn't support range bigger than 64 bits");

BVIndex end = start + len - 1;

AssertRange(end);

BVIndex iStart = BVUnit::Position(start);

BVIndex iEnd = BVUnit::Position(end);

BVIndex oStart = BVUnit::Offset(start);

BVIndex oEnd = BVUnit::Offset(end);

// Simply using uint64 because it is much easier than to juggle with BVUnit::BVUnitTContainer's size

// Special case, if oEnd == 63, 1 << 64 == 1. Therefore the result is incorrect

uint64 mask = oEnd < 63 ? (((uint64)1 << (oEnd + 1)) - 1) : 0xFFFFFFFFFFFFFFFF;

uint64 range;

// Trivial case

if (iStart == iEnd)

{

// remove the bits after oEnd with mask, then remove the bits before start with shift

range = (mask & this->data[iStart].GetWord()) >> oStart;

}

// Still simple enough

else if (iStart + 1 == iEnd)

{

auto startWord = this->data[iStart].GetWord();

auto endWord = this->data[iEnd].GetWord();

// remove the bits before start with shift

range = startWord >> oStart;

// remove the bits after oEnd with mask then position it after start bits

range |= (mask & endWord) << (BVUnit::BitsPerWord - oStart);

}

// Spans over multiple value, need to loop

else

{

// Get the first bits and move them to the beginning

range = this->data[iStart].GetWord() >> oStart;

// track how many bits have been read so far

int nBitsUsed = BVUnit::BitsPerWord - oStart;

for (uint i = iStart + 1; i < iEnd; ++i)

{

// put all bits from the data in the mid-range. Use the tracked read bits to position them

range |= ((uint64)(this->data[i].GetWord())) << nBitsUsed;

nBitsUsed += BVUnit::BitsPerWord;

}

// Read the last bits and remove those after oEnd with mask

range |= (mask & this->data[iEnd].GetWord()) << nBitsUsed;

}

return Container(range);

}

template<typename Container>

void BVFixed::SetRange(Container\* value, BVIndex start, BVIndex len)

{

AssertRange(start);

if (len == 0)

{

return;

}

Assert(len <= sizeof(Container) \* MachBits);

BVIndex end = start + len - 1;

AssertRange(end);

BVIndex iStart = BVUnit::Position(start);

BVIndex iEnd = BVUnit::Position(end);

BVIndex oStart = BVUnit::Offset(start);

BVIndex oEnd = BVUnit::Offset(end);

BVUnit::BVUnitTContainer\* bits = (BVUnit::BVUnitTContainer\*)value;

const int oStartComplement = BVUnit::BitsPerWord - oStart;

static\_assert((BVUnit::BVUnitTContainer)BVUnit::AllOnesMask > 0, "Container type of BVFixed must be unsigned");

//When making the mask, check the special case when we need all bits

#define MAKE\_MASK(start, end) ( ((end) == BVUnit::BitsPerWord ? BVUnit::AllOnesMask : (((BVUnit::BVUnitTContainer)1 << ((end) - (start))) - 1)) << (start))

// Or the value to set the bits to 1. And the value to set the bits to 0

// The mask is used to make sure we don't modify the bits outside the range

#define SET\_RANGE(i, value, mask) \

this->data[i].Or((value) & mask);\

this->data[i].And((value) | ~mask);

BVUnit::BVUnitTContainer bitsToSet;

// Fast Path

if (iEnd == iStart)

{

const BVUnit::BVUnitTContainer mask = MAKE\_MASK(oStart, oEnd + 1);

// Shift to position the bits

bitsToSet = (\*bits << oStart);

SET\_RANGE(iStart, bitsToSet, mask);

}

// TODO: case iEnd == iStart + 1 to avoid a loop

else if (oStart == 0)

{

// Simpler case where we don't have to shift the bits around

for (uint i = iStart; i < iEnd; ++i)

{

SET\_RANGE(i, \*bits, BVUnit::AllOnesMask);

++bits;

}

// We still need to use a mask to remove the unused bits

const BVUnit::BVUnitTContainer mask = MAKE\_MASK(0, oEnd + 1);

SET\_RANGE(iEnd, \*bits, mask);

}

else

{

// Default case. We need to process everything 1 at a time

{

// First set the first bits

const BVUnit::BVUnitTContainer mask = MAKE\_MASK(oStart, BVUnit::BitsPerWord);

SET\_RANGE(iStart, \*bits << oStart, mask);

}

// Set the bits in the middle

for (uint i = iStart + 1; i < iEnd; ++i)

{

bitsToSet = \*bits >> oStartComplement;

++bits;

bitsToSet |= \*bits << oStart;

SET\_RANGE(i, bitsToSet, BVUnit::AllOnesMask);

}

// Set the last bits

bitsToSet = \*bits >> oStartComplement;

++bits;

bitsToSet |= \*bits << oStart;

{

const BVUnit::BVUnitTContainer mask = MAKE\_MASK(0, oEnd + 1);

SET\_RANGE(iEnd, bitsToSet, mask);

}

}

#undef MAKE\_MASK

#undef SET\_RANGE

}

template <size\_t bitCount>

class BVStatic

{

public:

// Made public to allow for compile-time use

static const size\_t wordCount = ((bitCount - 1) >> BVUnit::ShiftValue) + 1;

// Data

private:

BVUnit data[wordCount];

public:

// Break on member changes. We rely on the layout of this class being static so we can

// use initializer lists to generate collections of BVStatic.

BVStatic()

{

Assert(sizeof(BVStatic<bitCount>) == sizeof(data));

Assert((void\*)this == (void\*)&this->data);

}

// Implementation

private:

void AssertRange(BVIndex i) const { Assert(i < bitCount); }

const BVUnit \* BitsFromIndex(BVIndex i) const { AssertRange(i); return &this->data[BVUnit::Position(i)]; }

BVUnit \* BitsFromIndex(BVIndex i) { AssertRange(i); return &this->data[BVUnit::Position(i)]; }

const BVUnit \* BeginUnit() const { return &this->data[0]; }

BVUnit \* BeginUnit() { return &this->data[0]; }

const BVUnit \* EndUnit() const { return &this->data[wordCount]; }

BVUnit \* EndUnit() { return &this->data[wordCount]; }

template<class Fn>

\_\_inline void for\_each(const BVStatic \*bv2, const Fn callback)

{

BVUnit \* i;

const BVUnit \* j;

for(i = this->BeginUnit(), j = bv2->BeginUnit();

i != this->EndUnit() ;

i++, j++)

{

(i->\*callback)(\*j);

}

}

template<class Fn>

static bool MapUntil(const BVStatic \*bv1, const BVStatic \*bv2, const Fn callback)

{

const BVUnit \* i;

const BVUnit \* j;

for(i = bv1->BeginUnit(), j = bv2->BeginUnit();

i != bv1->EndUnit() ;

i++, j++)

{

if (!callback(\*i, \*j))

{

return false;

}

}

return true;

}

void ClearEnd()

{

uint offset = BVUnit::Offset(bitCount);

if (offset != 0)

{

this->data[wordCount - 1].And((1 << offset) - 1);

}

}

// Methods

public:

void Set(BVIndex i)

{

AssertRange(i);

this->BitsFromIndex(i)->Set(BVUnit::Offset(i));

}

void Clear(BVIndex i)

{

AssertRange(i);

this->BitsFromIndex(i)->Clear(BVUnit::Offset(i));

}

void Compliment(BVIndex i)

{

AssertRange(i);

this->BitsFromIndex(i)->Complement(BVUnit::Offset(i));

}

BOOLEAN Equal(BVStatic<bitCount> const \* o)

{

return MapUntil(this, o, [](BVUnit const& i, BVUnit const &j) { return i.Equal(j); });

}

BOOLEAN Test(BVIndex i) const

{

AssertRange(i);

return this->BitsFromIndex(i)->Test(BVUnit::Offset(i));

}

BOOLEAN TestAndSet(BVIndex i)

{

AssertRange(i);

return \_bittestandset((long \*)this->data, (long) i);

}

BOOLEAN TestIntrinsic(BVIndex i) const

{

AssertRange(i);

return \_bittest((long \*)this->data, (long) i);

}

BOOLEAN TestAndSetInterlocked(BVIndex i)

{

AssertRange(i);

return \_interlockedbittestandset((long \*)this->data, (long) i);

}

BOOLEAN TestAndClear(BVIndex i)

{

AssertRange(i);

BVUnit \* bvUnit = this->BitsFromIndex(i);

BVIndex offset = BVUnit::Offset(i);

BOOLEAN bit = bvUnit->Test(offset);

bvUnit->Clear(offset);

return bit;

}

void OrComplimented(const BVStatic \* bv) { this->for\_each(bv, &BVUnit::OrComplimented); ClearEnd(); }

void Or(const BVStatic \*bv) { this->for\_each(bv, &BVUnit::Or); }

void And(const BVStatic \*bv) { this->for\_each(bv, &BVUnit::And); }

void Minus(const BVStatic \*bv) { this->for\_each(bv, &BVUnit::Minus); }

void Copy(const BVStatic \*bv) { js\_memcpy\_s(&this->data[0], wordCount \* sizeof(BVUnit), &bv->data[0], wordCount \* sizeof(BVUnit)); }

void SetAll() { memset(&this->data[0], -1, wordCount \* sizeof(BVUnit)); ClearEnd(); }

void ClearAll() { memset(&this->data[0], 0, wordCount \* sizeof(BVUnit)); }

void ComplimentAll()

{

for (BVIndex i = 0; i < wordCount; i++)

{

this->data[i].ComplimentAll();

}

ClearEnd();

}

BVIndex Count() const

{

BVIndex sum = 0;

for (BVIndex i = 0; i < wordCount; i++)

{

sum += this->data[i].Count();

}

Assert(sum <= bitCount);

return sum;

}

BVIndex Length() const

{

return bitCount;

}

JsUtil::FBVEnumerator BeginSetBits() { return JsUtil::FBVEnumerator(this->BeginUnit(), this->EndUnit()); }

BVIndex GetNextBit(BVIndex i) const

{

AssertRange(i);

const BVUnit \* chunk = BitsFromIndex(i);

BVIndex base = BVUnit::Floor(i);

BVIndex offset = chunk->GetNextBit(BVUnit::Offset(i));

if (-1 != offset)

{

return base + offset;

}

while (++chunk != this->EndUnit())

{

base += BVUnit::BitsPerWord;

offset = chunk->GetNextBit();

if (-1 != offset)

{

return base + offset;

}

}

return BVInvalidIndex;

}

const BVUnit \* GetRawData() const { return data; }

template <size\_t rangeSize>

BVStatic<rangeSize> \* GetRange(BVIndex startOffset)

{

AssertRange(startOffset);

AssertRange(startOffset + rangeSize - 1);

// Start offset and size must be word-aligned

Assert(BVUnit::Offset(startOffset) == 0);

Assert(BVUnit::Offset(rangeSize) == 0);

return (BVStatic<rangeSize> \*)BitsFromIndex(startOffset);

}

BOOLEAN TestRange(const BVIndex index, uint length) const

{

AssertRange(index);

AssertRange(index + length - 1);

const BVUnit \* bvUnit = BitsFromIndex(index);

uint offset = BVUnit::Offset(index);

if (offset + length <= BVUnit::BitsPerWord)

{

// Bit range is in a single word

return bvUnit->TestRange(offset, length);

}

// Bit range spans words.

// Test the first word, from start offset to end of word

if (!bvUnit->TestRange(offset, (BVUnit::BitsPerWord - offset)))

{

return FALSE;

}

bvUnit++;

length -= (BVUnit::BitsPerWord - offset);

// Test entire words until we are at the last word

while (length >= BVUnit::BitsPerWord)

{

if (!bvUnit->IsFull())

{

return FALSE;

}

bvUnit++;

length -= BVUnit::BitsPerWord;

}

// Test last word (unless we already ended on a word boundary)

if (length > 0)

{

if (!bvUnit->TestRange(0, length))

{

return FALSE;

}

}

return TRUE;

}

void SetRange(const BVIndex index, uint length)

{

AssertRange(index);

AssertRange(index + length - 1);

BVUnit \* bvUnit = BitsFromIndex(index);

uint offset = BVUnit::Offset(index);

if (offset + length <= BVUnit::BitsPerWord)

{

// Bit range is in a single word

return bvUnit->SetRange(offset, length);

}

// Bit range spans words.

// Set the first word, from start offset to end of word

bvUnit->SetRange(offset, (BVUnit::BitsPerWord - offset));

bvUnit++;

length -= (BVUnit::BitsPerWord - offset);

// Set entire words until we are at the last word

while (length >= BVUnit::BitsPerWord)

{

bvUnit->SetAll();

bvUnit++;

length -= BVUnit::BitsPerWord;

}

// Set last word (unless we already ended on a word boundary)

if (length > 0)

{

bvUnit->SetRange(0, length);

}

}

void ClearRange(const BVIndex index, uint length)

{

AssertRange(index);

AssertRange(index + length - 1);

BVUnit \* bvUnit = BitsFromIndex(index);

uint offset = BVUnit::Offset(index);

if (offset + length <= BVUnit::BitsPerWord)

{

// Bit range is in a single word

return bvUnit->ClearRange(offset, length);

}

// Bit range spans words.

// Clear the first word, from start offset to end of word

bvUnit->ClearRange(offset, (BVUnit::BitsPerWord - offset));

bvUnit++;

length -= (BVUnit::BitsPerWord - offset);

// Set entire words until we are at the last word

while (length >= BVUnit::BitsPerWord)

{

bvUnit->ClearAll();

bvUnit++;

length -= BVUnit::BitsPerWord;

}

// Set last word (unless we already ended on a word boundary)

if (length > 0)

{

bvUnit->ClearRange(0, length);

}

}

bool IsAllClear()

{

for (BVIndex i = 0; i < wordCount; i++)

{

if (!this->data[i].IsEmpty())

{

return false;

}

}

return true;

}

#if DBG\_DUMP

void Dump() const

{

bool hasBits = false;

Output::Print(L"[ ");

for (BVIndex i = 0; i < wordCount; i++)

{

hasBits = this->data[i].Dump(i \* BVUnit::BitsPerWord, hasBits);

}

Output::Print(L"]\n");

}

#endif

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

namespace JsUtil

{

FBVEnumerator::FBVEnumerator(BVUnit \* iterStart, BVUnit \* iterEnd):

icur(iterStart), iend(iterEnd),

curOffset(0)

{

if(this->icur != this->iend)

{

this->curUnit = \*iterStart;

this->MoveToNextBit();

}

}

void

FBVEnumerator::MoveToValidWord()

{

while(curUnit.IsEmpty())

{

this->icur++;

if(this->icur == this->iend)

{

return;

}

else

{

this->curUnit = \*this->icur;

this->curOffset += BVUnit::BitsPerWord;

}

}

}

void

FBVEnumerator::MoveToNextBit()

{

if(curUnit.IsEmpty())

{

this->curOffset = BVUnit::Floor(curOffset);

this->MoveToValidWord();

if(this->End())

{

return;

}

}

BVIndex i = curUnit.GetNextBit();

AssertMsg(BVInvalidIndex != i, "Fatal Exception. Error in Bitvector implementation");

curOffset = BVUnit::Floor(curOffset) + i ;

curUnit.Clear(i);

}

void

FBVEnumerator::operator++(int)

{

AssertMsg(this->icur != this->iend, "Iterator past the end of bit stream");

this->MoveToNextBit();

}

BVIndex

FBVEnumerator::GetCurrent() const

{

return this->curOffset;

}

bool

FBVEnumerator::End() const

{

return this->icur == this->iend;

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

class FBVEnumerator

{

// Data

private:

BVUnit \*icur, \*iend;

BVIndex curOffset;

BVUnit curUnit;

// Constructor

public:

FBVEnumerator(BVUnit \* iterStart, BVUnit \* iterEnd);

// Implementation

protected:

void MoveToValidWord();

void MoveToNextBit();

// Methods

public:

void operator++(int);

BVIndex GetCurrent() const;

bool End() const;

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template<class T, int N>

class FixedStack

{

private:

T itemList[N];

int curIndex;

public:

FixedStack(): curIndex(-1)

{

}

void Push(T item)

{

AssertMsg(curIndex < N - 1, "Stack overflow");

if (curIndex >= N - 1)

{

Js::Throw::FatalInternalError();

}

this->itemList[++this->curIndex] = item;

}

T\* Pop()

{

AssertMsg(curIndex >= 0, "Stack Underflow");

if (curIndex < 0)

{

Js::Throw::FatalInternalError();

}

return &this->itemList[this->curIndex--];

}

T\* Peek()

{

AssertMsg(curIndex >= 0, "No element present");

if (curIndex < 0)

{

Js::Throw::FatalInternalError();

}

return & this->itemList[this->curIndex];

}

int Count()

{

return 1 + this->curIndex;

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

#include "Common\UInt32Math.h"

#include "DataStructures\growingArray.h"

namespace JsUtil

{

GrowingUint32HeapArray\* GrowingArray<uint32, HeapAllocator>::Create(uint32 \_length)

{

return HeapNew(GrowingUint32HeapArray, &HeapAllocator::Instance, \_length);

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// Contains a class which will provide a uint32 array which can grow dynamically

// It behaves almost same as regex::List<> except it has less members, is customized for being used in SmallSpanSequence of FunctionBody

#pragma once

#ifdef DIAG\_MEM

extern int listFreeAmount;

#endif

namespace JsUtil

{

template <class TValue, class TAllocator>

class GrowingArray

{

public:

typedef typename AllocatorInfo<TAllocator, TValue>::AllocatorType AllocatorType;

static GrowingArray\* Create(uint32 \_length);

GrowingArray(AllocatorType\* allocator, uint32 \_length)

: buffer(nullptr),

alloc(allocator),

count(0),

length(\_length)

{

EnsureArray();

}

~GrowingArray()

{

if (buffer != nullptr)

{

AllocatorFree(alloc, (TypeAllocatorFunc<AllocatorType, int>::GetFreeFunc()), buffer, UInt32Math::Mul(length, sizeof(TValue)));

}

}

TValue ItemInBuffer(uint32 index)

{

if (index >= count)

{

return 0;

}

return buffer[index];

}

void ItemInBuffer(uint32 index, TValue item)

{

EnsureArray();

Assert(index < count);

buffer[index] = item;

}

void Add(TValue item)

{

EnsureArray();

buffer[count] = item;

count++;

}

uint32 Count() const { return count; }

void SetCount(uint32 \_count) { count = \_count; }

uint32 GetLength() const { return length; }

TValue\* GetBuffer() const { return buffer; }

GrowingArray \* Clone()

{

GrowingArray \* pNewArray = AllocatorNew(AllocatorType, alloc, GrowingArray, alloc, length);

pNewArray->count = count;

if (buffer)

{

pNewArray->buffer = AllocateArray<AllocatorType, TValue, false>(

TRACK\_ALLOC\_INFO(alloc, TValue, AllocatorType, 0, length),

TypeAllocatorFunc<AllocatorType, TValue>::GetAllocFunc(),

length);

const size\_t byteSize = UInt32Math::Mul(length, sizeof(TValue));

js\_memcpy\_s(pNewArray->buffer, byteSize, buffer, byteSize);

}

return pNewArray;

}

private:

TValue\* buffer;

uint32 count;

uint32 length;

AllocatorType\* alloc;

void EnsureArray()

{

if (buffer == nullptr)

{

buffer = AllocateArray<AllocatorType, TValue, false>(

TRACK\_ALLOC\_INFO(alloc, TValue, AllocatorType, 0, length),

TypeAllocatorFunc<AllocatorType, TValue>::GetAllocFunc(),

length);

count = 0;

}

else if (count == length)

{

uint32 newLength = UInt32Math::AddMul<1, 2>(length);

TValue \* newbuffer = AllocateArray<AllocatorType, TValue, false>(

TRACK\_ALLOC\_INFO(alloc, TValue, AllocatorType, 0, newLength),

TypeAllocatorFunc<AllocatorType, TValue>::GetAllocFunc(),

newLength);

const size\_t lengthByteSize = UInt32Math::Mul(length, sizeof(TValue));

const size\_t newLengthByteSize = UInt32Math::Mul(newLength, sizeof(TValue));

js\_memcpy\_s(newbuffer, newLengthByteSize, buffer, lengthByteSize);

#ifdef DIAG\_MEM

listFreeAmount += length;

#endif

if (length != 0)

{

AllocatorFree(alloc, (TypeAllocatorFunc<AllocatorType, int>::GetFreeFunc()), buffer, lengthByteSize);

}

length = newLength;

buffer = newbuffer;

}

}

};

typedef GrowingArray<uint32, HeapAllocator> GrowingUint32HeapArray;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#if PROFILE\_DICTIONARY

#include "DictionaryStats.h"

#endif

template<typename T>

class Bucket

{

public:

T element;

int value;

public:

Bucket() : element(), value(0) {}

static void Copy(Bucket<T> const& bucket, Bucket<T>& newBucket)

{

bucket.element.Copy(&(newBucket.element));

newBucket.value = bucket.value;

}

};

template<typename T, typename TAllocator = ArenaAllocator>

class HashTable

{

public:

TAllocator \* alloc;

uint tableSize;

SListBase<Bucket<T>> \* table;

public:

static HashTable<T, TAllocator> \* New(TAllocator \*allocator, uint tableSize)

{

return AllocatorNewPlus(TAllocator, allocator, (tableSize\*sizeof(SListBase<Bucket<T>>)), HashTable, allocator, tableSize);

}

void Delete()

{

AllocatorDeletePlus(TAllocator, alloc, (tableSize\*sizeof(SListBase<Bucket<T>>)), this);

}

~HashTable()

{

for (uint i = 0; i< tableSize; i++)

{

table[i].Clear(alloc);

}

}

SListBase<Bucket<T>> \* SwapBucket(SListBase<Bucket<T>> \* newTable)

{

SListBase<Bucket<T>> \* retTable = table;

table = newTable;

return retTable;

}

T \* FindOrInsertNew(int value)

{

uint hash = this->Hash(value);

#if PROFILE\_DICTIONARY

uint depth = 1;

#endif

// Keep sorted

FOREACH\_SLISTBASE\_ENTRY\_EDITING(Bucket<T>, bucket, &this->table[hash], iter)

{

if (bucket.value <= value)

{

if (bucket.value == value)

{

return &(bucket.element);

}

break;

}

#if PROFILE\_DICTIONARY

++depth;

#endif

} NEXT\_SLISTBASE\_ENTRY\_EDITING;

Bucket<T> \* newBucket = iter.InsertNodeBefore(this->alloc);

newBucket->value = value;

#if PROFILE\_DICTIONARY

if (stats)

stats->Insert(depth);

#endif

return &newBucket->element;

}

T \* FindOrInsertNewNoThrow(int value)

{

uint hash = this->Hash(value);

#if PROFILE\_DICTIONARY

uint depth = 1;

#endif

// Keep sorted

FOREACH\_SLISTBASE\_ENTRY\_EDITING(Bucket<T>, bucket, &this->table[hash], iter)

{

if (bucket.value <= value)

{

if (bucket.value == value)

{

return &(bucket.element);

}

break;

}

#if PROFILE\_DICTIONARY

++depth;

#endif

} NEXT\_SLISTBASE\_ENTRY\_EDITING;

Bucket<T> \* newBucket = iter.InsertNodeBeforeNoThrow(this->alloc);

if (newBucket == nullptr)

{

return nullptr;

}

newBucket->value = value;

#if PROFILE\_DICTIONARY

if (stats)

stats->Insert(depth);

#endif

return &newBucket->element;

}

T \* FindOrInsert(T element, int value)

{

uint hash = this->Hash(value);

#if PROFILE\_DICTIONARY

uint depth = 1;

#endif

// Keep sorted

FOREACH\_SLISTBASE\_ENTRY\_EDITING(Bucket<T>, bucket, &this->table[hash], iter)

{

if (bucket.value <= value)

{

if (bucket.value == value)

{

return &(bucket.element);

}

break;

}

#if PROFILE\_DICTIONARY

++depth;

#endif

} NEXT\_SLISTBASE\_ENTRY\_EDITING;

Bucket<T> \* newBucket = iter.InsertNodeBefore(this->alloc);

Assert(newBucket != nullptr);

newBucket->value = value;

newBucket->element = element;

#if PROFILE\_DICTIONARY

if (stats)

stats->Insert(depth);

#endif

return nullptr;

}

T \* Get(int value)

{

// Assumes sorted lists

FOREACH\_SLISTBASE\_ENTRY(Bucket<T>, bucket, &this->table[this->Hash(value)])

{

if (bucket.value <= value)

{

if (bucket.value == value)

{

return &(bucket.element);

}

break;

}

} NEXT\_SLISTBASE\_ENTRY;

return nullptr;

}

T GetAndClear(int value)

{

SListBase<Bucket<T>> \* list = &this->table[this->Hash(value)];

#if PROFILE\_DICTIONARY

bool first = true;

#endif

// Assumes sorted lists

FOREACH\_SLISTBASE\_ENTRY\_EDITING(Bucket<T>, bucket, list, iter)

{

if (bucket.value <= value)

{

if (bucket.value == value)

{

T retVal = bucket.element;

iter.RemoveCurrent(this->alloc);

#if PROFILE\_DICTIONARY

if (stats)

stats->Remove(first && !(iter.Next()));

#endif

return retVal;

}

break;

}

#if PROFILE\_DICTIONARY

first = false;

#endif

} NEXT\_SLISTBASE\_ENTRY\_EDITING;

return T();

}

void Clear(int value)

{

SListBase<Bucket<T>> \* list = &this->table[this->Hash(value)];

// Assumes sorted lists

#if PROFILE\_DICTIONARY

bool first = true;

#endif

FOREACH\_SLISTBASE\_ENTRY\_EDITING(Bucket<T>, bucket, list, iter)

{

if (bucket.value <= value)

{

if (bucket.value == value)

{

iter.RemoveCurrent(this->alloc);

#if PROFILE\_DICTIONARY

if (stats)

stats->Remove(first && !(iter.Next()));

#endif

}

return;

}

#if PROFILE\_DICTIONARY

first = false;

#endif

} NEXT\_SLISTBASE\_ENTRY\_EDITING;

}

void And(HashTable<T> \*this2)

{

for (uint i = 0; i < this->tableSize; i++)

{

SListBase<Bucket<T>>::Iterator iter2(&this2->table[i]);

iter2.Next();

FOREACH\_SLISTBASE\_ENTRY\_EDITING(Bucket<T>, bucket, &this->table[i], iter)

{

while (iter2.IsValid() && bucket.value < iter2.Data().value)

{

iter2.Next();

}

if (!iter2.IsValid() || bucket.value != iter2.Data().value || bucket.element != iter2.Data().element)

{

iter.RemoveCurrent(this->alloc);

#if PROFILE\_DICTIONARY

if (stats)

stats->Remove(false);

#endif

continue;

}

else

{

AssertMsg(bucket.value == iter2.Data().value && bucket.element == iter2.Data().element, "Huh??");

}

iter2.Next();

} NEXT\_SLISTBASE\_ENTRY\_EDITING;

}

}

// "And" with fixup actions to take when data don't make it to the result.

template <class FnFrom, class FnTo>

void AndWithFixup(HashTable<T> \*this2, FnFrom fnFixupFrom, FnTo fnFixupTo)

{

for (uint i = 0; i < this->tableSize; i++)

{

SListBase<Bucket<T>>::Iterator iter2(&this2->table[i]);

iter2.Next();

FOREACH\_SLISTBASE\_ENTRY\_EDITING(Bucket<T>, bucket, &this->table[i], iter)

{

while (iter2.IsValid() && bucket.value < iter2.Data().value)

{

// Skipping a this2 value.

fnFixupTo(iter2.Data());

iter2.Next();

}

if (!iter2.IsValid() || bucket.value != iter2.Data().value || bucket.element != iter2.Data().element)

{

// Skipping a this value.

fnFixupFrom(bucket);

iter.RemoveCurrent(this->alloc);

#if PROFILE\_DICTIONARY

if (stats)

stats->Remove(false);

#endif

continue;

}

else

{

AssertMsg(bucket.value == iter2.Data().value && bucket.element == iter2.Data().element, "Huh??");

}

iter2.Next();

} NEXT\_SLISTBASE\_ENTRY\_EDITING;

while (iter2.IsValid())

{

// Skipping a this2 value.

fnFixupTo(iter2.Data());

iter2.Next();

}

}

}

template <class Fn>

void Or(HashTable<T> \* this2, Fn fn)

{

for (uint i = 0; i < this->tableSize; i++)

{

SListBase<Bucket<T>>::Iterator iter2(&this2->table[i]);

iter2.Next();

FOREACH\_SLISTBASE\_ENTRY\_EDITING(Bucket<T>, bucket, &this->table[i], iter)

{

while (iter2.IsValid() && bucket.value < iter2.Data().value)

{

Bucket<T> \* newBucket = iter.InsertNodeBefore(this->alloc);

newBucket->value = iter2.Data().value;

newBucket->element = fn(nullptr, iter2.Data().element);

iter2.Next();

}

if (!iter2.IsValid())

{

break;

}

if (bucket.value == iter2.Data().value)

{

bucket.element = fn(bucket.element, iter2.Data().element);

iter2.Next();

}

} NEXT\_SLISTBASE\_ENTRY\_EDITING;

while (iter2.IsValid())

{

Bucket<T> \* newBucket = iter.InsertNodeBefore(this->alloc);

newBucket->value = iter2.Data().value;

newBucket->element = fn(nullptr, iter2.Data().element);

iter2.Next();

}

}

}

HashTable<T> \*Copy()

{

HashTable<T> \*newTable = HashTable<T>::New(this->alloc, this->tableSize);

for (uint i = 0; i < this->tableSize; i++)

{

this->table[i].CopyTo<Bucket<T>::Copy>(this->alloc, newTable->table[i]);

}

#if PROFILE\_DICTIONARY

if (stats)

newTable->stats = stats->Clone();

#endif

return newTable;

}

void ClearAll()

{

for (uint i = 0; i < this->tableSize; i++)

{

this->table[i].Clear(this->alloc);

}

#if PROFILE\_DICTIONARY

// To not lose previously collected data, we will treat cleared dictionary as a separate instance for stats tracking purpose

stats = DictionaryStats::Create(typeid(this).name(), tableSize);

#endif

}

#if DBG\_DUMP

void Dump(uint newLinePerEntry = 2);

void Dump(void (\*valueDump)(int));

#endif

protected:

HashTable(TAllocator \* allocator, uint tableSize) : alloc(allocator), tableSize(tableSize)

{

Init();

#if PROFILE\_DICTIONARY

stats = DictionaryStats::Create(typeid(this).name(), tableSize);

#endif

}

void Init()

{

table = (SListBase<Bucket<T>> \*)(((char \*)this) + sizeof(HashTable<T>));

for (uint i = 0; i < tableSize; i++)

{

// placement new

::new (&table[i]) SListBase<Bucket<T>>();

}

}

private:

uint Hash(int value) { return (value % this->tableSize); }

#if PROFILE\_DICTIONARY

DictionaryStats \*stats;

#endif

};

template <typename T, uint size, typename TAllocator = ArenaAllocator>

class HashTableS : public HashTable<T, TAllocator>

{

public:

HashTableS(TAllocator \* allocator) : HashTable(allocator, size) {}

void Reset()

{

\_\_super::Init();

}

private:

char tableSpace[size \* sizeof(SListBase<Bucket<T>>)];

};

#define FOREACH\_HASHTABLE\_ENTRY(T, bucket, hashTable) \

for (uint \_iterHash = 0; \_iterHash < (hashTable)->tableSize; \_iterHash++) \

{ \

FOREACH\_SLISTBASE\_ENTRY(Bucket<T>, bucket, &(hashTable)->table[\_iterHash]) \

{

#define NEXT\_HASHTABLE\_ENTRY \

} \

NEXT\_SLISTBASE\_ENTRY; \

}

#if DBG\_DUMP

template <typename T, typename TAllocator>

inline void

HashTable<T, TAllocator>::Dump(uint newLinePerEntry)

{

FOREACH\_HASHTABLE\_ENTRY(T, bucket, this)

{

Output::Print(L"%4d => ", bucket.value);

::Dump<T>(bucket.element);

for (uint i = 0; i < newLinePerEntry; i++)

{

Output::Print(L"\n");

}

}

NEXT\_HASHTABLE\_ENTRY;

}

template <typename T, typename TAllocator>

inline void

HashTable<T, TAllocator>::Dump(void (\*valueDump)(int))

{

FOREACH\_HASHTABLE\_ENTRY(T, bucket, this)

{

valueDump(bucket.value);

Output::Print(L" => ", bucket.value);

::Dump<T>(bucket.element);

Output::Print(L"\n");

}

NEXT\_HASHTABLE\_ENTRY;

}

template <typename T>

inline void Dump(T const& t)

{

t.Dump();

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

#include <strsafe.h>

#include "Option.h"

#include "ImmutableList.h"

template<int chunkSize>

void regex::ImmutableStringBuilder<chunkSize>::AppendInt32(int32 value)

{

WCHAR buffer[11]; // -2,147,483,648 w.o ',' + \0

HRESULT hr = S\_OK;

hr = StringCchPrintfW(buffer, \_countof(buffer), L"%d", value);

AssertMsg(SUCCEEDED(hr), "StringCchPrintfW");

if (FAILED(hr) )

{

Js::Throw::OutOfMemory();

}

// append to the stringBuilder string

this->AppendWithCopy(buffer);

}

template<int chunkSize>

void regex::ImmutableStringBuilder<chunkSize>::AppendUInt64(uint64 value)

{

WCHAR buffer[21]; // 18,446,744,073,709,551,615 w.o ',' + \0

HRESULT hr = S\_OK;

hr = StringCchPrintfW(buffer, \_countof(buffer), L"%llu", value);

AssertMsg(SUCCEEDED(hr), "StringCchPrintfW");

if (FAILED(hr) )

{

Js::Throw::OutOfMemory();

}

// append to the stringBuilder string

this->AppendWithCopy(buffer);

}

template<int chunkSize>

void regex::ImmutableStringBuilder<chunkSize>::AppendWithCopy(\_In\_z\_ LPCWSTR str)

{

AssertMsg(str != nullptr, "str != nullptr");

size\_t strLength = wcslen(str) + 1; // include null-terminated

WCHAR\* buffer = new WCHAR[strLength];

IfNullThrowOutOfMemory(buffer);

wcsncpy\_s(buffer, strLength, str, strLength);

// append in front of the tracked allocated strings

AllocatedStringChunk\* newAllocatedString = new AllocatedStringChunk();

if (newAllocatedString == nullptr)

{

// cleanup

delete[] buffer;

Js::Throw::OutOfMemory();

}

newAllocatedString->dataPtr = buffer;

newAllocatedString->next = this->allocatedStringChunksHead;

this->allocatedStringChunksHead = newAllocatedString;

// append to the stringBuilder string

this->Append(buffer);

}

// template instantiation

template void regex::ImmutableStringBuilder<8>::AppendInt32(int32 value);

template void regex::ImmutableStringBuilder<8>::AppendUInt64(uint64 value);

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#define IfNullThrowOutOfMemory(result) if(result == nullptr) { Js::Throw::OutOfMemory(); }

namespace regex

{

template<class T>

class ImmutableList

{

T value;

ImmutableList<T> \* next;

ImmutableList(T value, ImmutableList<T> \* next)

: value(value), next(next)

{ }

public:

// Delete all the nodes in the list (if any).

void FreeList(ArenaAllocator \* a)

{

if (this == nullptr)

{

return;

}

auto nextToDelete = this->next;

Adelete(a, this);

nextToDelete->FreeList(a);

}

// Info: Return a list with the given value prepended to this one.

// Parameters: value - the value to prepend

ImmutableList<T> \* Prepend(T value, ArenaAllocator \* a)

{

return Anew(a, ImmutableList, value, this);

}

// Info: Return a list with the given value appended to this one <Note this modifies existing list>

// It is requred that the tail passed is empty if this is empty list otherwise its the last node of this list

// Parameters: value - the value to append

// tail - last node of the new list

ImmutableList<T> \* Append(T value, ArenaAllocator \* a, ImmutableList<T> \*\*tail)

{

#if DBG

Assert(tail != nullptr);

Assert((this->IsEmpty() && (\*tail)->IsEmpty()) || (\*tail)->next->IsEmpty());

if (!this->IsEmpty())

{

auto current = this;

while(!current->next->IsEmpty())

{

current = current->next;

}

Assert(current == (\*tail));

}

#endif

if (IsEmpty())

{

\*tail = Anew(a, ImmutableList, value, ImmutableList<T>::Empty());

return \*tail;

}

(\*tail)->next = Anew(a, ImmutableList, value, ImmutableList<T>::Empty());

\*tail = (\*tail)->next;

return this;

}

// Info: Return a list with the given array prepended to this one.

// Parameters: arr - the array to prepend

// count - the elements in the array

ImmutableList<T> \* PrependArray(T \* arr, size\_t count, ArenaAllocator \* a)

{

Assert(count > 0);

// Create the list

// Using Append here instead of prepend so that we dont need to make another pass just to get the tail and attach the current list at the end of array list

auto out = ImmutableList<T>::Empty();

auto tail = out;

out = out->AppendArrayToCurrentList(arr, count, a, &tail);

Assert(!tail->IsEmpty());

Assert(tail->next->IsEmpty());

tail->next = this;

return out;

}

// Info: Return the list which is resulted by appending array to the current list<Note this modifies existing list>

// It is requred that the tail passed is empty if this is empty list otherwise its the last node of this list

// Parameters: arr - the array to append

// count - the elements in the array

// tail - last node of the new list

ImmutableList<T> \* AppendArrayToCurrentList(T \* arr, size\_t count, ArenaAllocator \* a, ImmutableList<T> \*\*tail)

{

auto out = this;

for(size\_t i=0; i<count; i++)

{

out = out->Append(arr[i],a,tail);

}

return out;

}

// Info: Return a list with the given list appended to this one <Note this modifies existing lists>

// returns the list that looks like thisList -> list

// Parameters: list - the list to append

// tail - optional end ptr so we dont need to traverse the list to find the tail node

ImmutableList<T> \* AppendListToCurrentList(ImmutableList<T> \* list, ImmutableList<T> \* tail = ImmutableList<T>::Empty()) // TODO : figure out all the tail scenarios

{

if (list->IsEmpty())

{

return this;

}

auto out = this;

if (out->IsEmpty())

{

return list;

}

if (tail->IsEmpty())

{

// We dont have tail node, find it

tail = this;

auto current = this;

while(!current->IsEmpty())

{

tail = current;

current = current->next;

}

Assert(tail->next->IsEmpty());

}

#if DBG

else

{

auto current = this;

while(!current->next->IsEmpty())

{

current = current->next;

}

Assert(current == tail);

}

#endif

Assert(!tail->IsEmpty() && tail->next->IsEmpty());

tail->next = list;

return out;

}

// Info: Return a new list by calling function f called on each element of this list

// Parameters: f - the mapping function

template<class TOut, class F>

ImmutableList<TOut> \* Select(F f, ArenaAllocator \* a)

{

auto out = ImmutableList<TOut>::Empty();

auto tail = out;

auto current = this;

while(!current->IsEmpty())

{

out = out->Append(f(current->First()), a, &tail);

current = current->next;

}

return out;

}

// Info: Changes the values of current list with the values returned by calling function f on each element <Note this modifies values in the list>

// Parameters: f - the mapping function

template<class F>

ImmutableList<T> \* SelectInPlace(F f)

{

auto current = this;

while(!current->IsEmpty())

{

current->value = f(current->First());

current = current->next;

}

return this;

}

// Info: Return true if any element returns true for the given predicate

// Parameters: f - the predicate

template<class F>

bool Any(F f)

{

return WhereFirst(f).HasValue();

}

// Info: Return a new list by calling function f called on each element of this list. Remove any elements where f returned nullptr.

// Parameters: f - the mapping function

template<class TOut, class F>

ImmutableList<TOut> \* SelectNotNull(F f, ArenaAllocator \* a)

{

auto out = ImmutableList<TOut>::Empty();

auto tail = out;

auto current = this;

while(!current->IsEmpty())

{

auto result = f(current->First());

if (result!=nullptr)

{

out = out->Append(result, a, &tail);

}

current = current->next;

}

return out;

}

// Info: Statically cast one list to another. The elements must be statically related

template<class TOut>

ImmutableList<TOut> \* Cast()

{

#if DBG

// Ensure static\_cast is valid

T t = T();

TOut to = static\_cast<TOut>(t);

to;

#endif

return reinterpret\_cast<ImmutableList<TOut>\*>(this);

}

// Info: Call function f for each element of the list

// Parameters: f - the function to call

template<class F>

void Iterate(F f)

{

auto current = this;

while(!current->IsEmpty())

{

f(current->First());

current = current->next;

}

}

// Info: Call function f for each element of the list with an index for each

// Parameters: f - the function to call

// returns true to continue iterating, false to stop.

template<class F>

void IterateWhile(F f)

{

auto current = this;

while(!current->IsEmpty())

{

bool shouldContinue = f(current->First());

if (!shouldContinue)

{

break;

}

current = current->next;

}

}

// Info: Call function f for each element of the list with an index for each

// Parameters: f - the function to call

template<class F>

void IterateN(F f)

{

auto current = this;

auto index = 0;

while(!current->IsEmpty())

{

f(index, current->First());

current = current->next;

++index;

}

}

// Info: Call function f for first N elements of the list

// Parameters: f - the function to call

template<class F>

void IterateFirstN(size\_t N, F f)

{

Assert(Count() >= N);

auto current = this;

while(N > 0)

{

f(current->First());

current = current->next;

N--;

}

}

// Info: Call function f for first N elements of the list, f2 on the next element and f3 for remaining elements

// Parameters: N - number of elements to call f with

// f - the function to call on first N elements

// f2 - the function to call on N+1th element

// f3 - the function to call on remaining elements

template<class F, class F2, class F3>

void IterateIn3Sets(size\_t N, F f, F2 f2, F3 f3)

{

Assert(Count() > N);

auto current = this;

for (size\_t i = 0; i < N; i++)

{

f(current->First());

current = current->next;

}

Assert(current != nullptr);

f2(current->First());

current = current->next;

while(current)

{

f3(current->First());

current = current->next;

}

}

// Info: Iterate two lists at once. Stop when either list runs out.

// Parameters: f - the function to call on each pair

template <class T2, class F> void IterateWith(ImmutableList<T2> \*e2,F f)

{

auto en1 = this;

auto en2 = e2;

while(!en1->IsEmpty() && !en2->IsEmpty())

{

f(en1->First(),en2->First());

en1 = en1->GetTail();

en2 = en2->GetTail();

}

}

// Info: Iterate over the elements of the enumerable and accumulate a result

// Parameters: f - the function to call for each element

template <class TAccumulator, class F> TAccumulator Accumulate(TAccumulator seed, F f)

{

auto current = this;

auto accumulated = seed;

while(!current->IsEmpty())

{

accumulated = f(accumulated, current->value);

current = current->next;

}

return accumulated;

}

// Info: Sum the elements of the list using values returned from the given function

// Parameters: f - the function to call for each element

template <class TSize, class F> TSize Sum(F f)

{

TSize sum = TSize();

auto current = this;

while(!current->IsEmpty())

{

sum += f(current->value);

current = current->next;

}

return sum;

}

// Info: Return true if f returns true for all elements

// Parameters: f - the function to call for each element

template <class F> bool TrueForAll(F f)

{

auto current = this;

auto stillTrue = true;

while(!current->IsEmpty() && stillTrue)

{

stillTrue = stillTrue && f(current->value);

current = current->next;

}

return stillTrue;

}

// Info: Return this list reversed

ImmutableList<T> \* Reverse(ArenaAllocator \* a)

{

auto out = ImmutableList<T>::Empty();

auto current = this;

while(!current->IsEmpty())

{

out = out->Prepend(current->First(),a);

current = current->next;

}

return out;

}

// Info: Return this list reversed <Reverses the current list>

ImmutableList<T> \* ReverseCurrentList()

{

auto out = ImmutableList<T>::Empty();

auto current = this;

while(!current->IsEmpty())

{

auto next = current->next;

current->next = out;

out = current;

current = next;

}

return out;

}

// Info: Filter this list by removing the elements where function f returns false

// Parameters: f - the predicate function to filter by

template<class F>

ImmutableList<T> \* Where(F f, ArenaAllocator \* a)

{

auto out = ImmutableList<T>::Empty();

auto tail = out;

auto current = this;

while(!current->IsEmpty())

{

auto head = current->First();

if (f(head))

{

out = out->Append(head, a, &tail);

}

current = current->next;

}

return out;

}

// Info: Filter this list by removing the elements where function f returns false <Modifies existing list with removal of nodes that dont satisfy the condition f()>

// Parameters: f - the predicate function to filter by

template<class F>

ImmutableList<T> \* WhereInPlace(F f)

{

auto out = ImmutableList<T>::Empty();

auto tail = out;

auto current = this;

while(!current->IsEmpty())

{

auto head = current->First();

if (f(head))

{

if (out->IsEmpty())

{

// Need to add to the head

out = current;

}

else

{

// Add to the tail

Assert(!tail->IsEmpty());

tail->next = current;

}

tail = current;

}

current = current->next;

}

if (!tail->IsEmpty())

{

tail->next = ImmutableList<T>::Empty();

}

return out;

}

// Info: Return a new list by calling function f called on element of this list that satisfy predicate wf

// Parameters: wf - the predicate function to filter by

// f - the mapping function

template<class TOut, class selectF, class whereF>

ImmutableList<TOut> \* WhereSelect(whereF wf, selectF sf, ArenaAllocator \* a)

{

auto out = ImmutableList<TOut>::Empty();

auto tail = out;

auto current = this;

while(!current->IsEmpty())

{

auto head = current->First();

if (wf(head))

{

out = out->Append(sf(current->First()), a, &tail);

}

current = current->next;

}

return out;

}

// Info: Run through current list to find the 1 or 0 item that satisfied predicate f

// Parameters: f - the predicate function to filter by

// Info: If there are 0 or 1 elements, return an option. Throw otherwise.

template<class TOption, class F>

Option<TOption> WhereToOption(F f)

{

auto out = ImmutableList<T>::Empty();

auto current = this;

while(!current->IsEmpty())

{

auto head = current->First();

if (f(head))

{

if (out->IsEmpty())

{

out = current;

}

else

{

// Cannot convert Enumerable to Option because there is more than 1 item in the Enumerable

Js::Throw::FatalProjectionError();

}

}

current = current->next;

}

if (out->IsEmpty())

{

return nullptr;

}

return out->First();

}

// Info: Filter the list down to exactly one item.

// Parameters: f - the predicate function to filter by

template<class F>

T WhereSingle(F f)

{

auto current = this;

T \* result = nullptr;

while(!current->IsEmpty())

{

auto head = current->First();

if (f(head))

{

Js::VerifyCatastrophic(result==nullptr); // There are multiple matching items.

result = &current->value;

}

current = current->next;

}

Js::VerifyCatastrophic(result!=nullptr); // There are no matching items.

return \*result;

}

// Info: Return the first matching item

// Parameters: f - the predicate function to filter by

template<class F>

Option<T> WhereFirst(F f)

{

auto current = this;

while(!current->IsEmpty())

{

auto head = current->First();

if (f(head))

{

return &current->value;

}

current = current->next;

}

return Option<T>();

}

// Info: Return the count of matching items

// Parameters: f - the predicate function to filter by

template<class F>

size\_t CountWhere(F f)

{

auto current = this;

size\_t count = 0;

while(!current->IsEmpty())

{

auto head = current->First();

if (f(head))

{

++count;

}

current = current->next;

}

return count;

}

// Info: Return true if any item returns true for f

// Parameters: f - the predicate function to filter by

template<class F>

bool ContainsWhere(F f)

{

return WhereFirst(f).HasValue();

}

// Info: Remove all instances of the given value from current list <Modifies existing list with removal of nodes that has 'value'>

// Parameters: value - the value to remove from the list

ImmutableList<T> \* RemoveValueInPlace(T value)

{

return WhereInPlace([&](T seen) {return seen != value;});

}

// Info: Return the value from a list with exacly one element. Throw if there are 0 or 2+ elements.

T ToSingle()

{

Js::VerifyCatastrophic(Count()==1);

return value;

}

// Info: If there are 0 or 1 elements, return an option. Throw otherwise.

template<class TOption>

Option<TOption> ToOption()

{

auto en = this;

if (en == Empty())

{

return nullptr;

}

Js::VerifyCatastrophic(en->next == Empty()); // Cannot convert Enumerable to Option because there is more than 1 item in the Enumerable

return First();

}

// Info: Return the first element

T& First()

{

Js::VerifyCatastrophic(!IsEmpty());

return value;

}

// Info: Return the last item. Throw if there are none.

T& Last()

{

Js::VerifyCatastrophic(!IsEmpty());

if (next->IsEmpty())

{

return value;

}

return next->Last(); // Warning: possible stack usage

}

// Info: Return the nth item from the list. Throw if this list isn't long enough.

T& Nth(size\_t n)

{

Js::VerifyCatastrophic(!IsEmpty());

if(n==0)

{

return value;

}

return next->Nth(n-1); // Warning: possible stack usage

}

// Info: Return the rest of the list

ImmutableList<T> \* GetTail()

{

return next;

}

// Info: Return the empty list

static ImmutableList<T> \* Empty()

{

return nullptr;

}

// Info: Iterate over the elements of an enumerable calling f1 for each and f2 between each.

// Parameters: value - the value to remove from the list

template <class F1, class F2>

void IterateBetween(F1 f1, F2 f2)

{

auto en = this;

bool more = en!=Empty();

while(more)

{

auto last = en->First();

f1(last);

en = en->next;

more = en!=Empty();

if (more)

{

auto next = en->First();

f2(last,next);

}

}

}

// Info: Return the size of the list

size\_t Count()

{

size\_t count = 0;

auto current = this;

while(!current->IsEmpty())

{

++count;

current = current->next;

}

return count;

}

// Info: Return this list converted to an array in the heap. Get the size by calling Count().

T \*\* ToReferenceArrayInHeap(size\_t size)

{

Assert(size == Count());

auto result = new T \*[size];

IfNullThrowOutOfMemory(result);

auto current = this;

for(size\_t index = 0; index<size; ++index)

{

#pragma warning(push)

#pragma warning(disable:22102)

result[index] = &current->value;

#pragma warning(pop)

current = current->next;

}

Js::VerifyCatastrophic(current->IsEmpty());

return result;

}

// Info: Sort the list by the given reference comparer <Modifies existing list such that it is ordered using comparer>

// Parameters: comparer - comparer to sort with

ImmutableList<T> \* SortCurrentList(regex::Comparer<T \*> \* comparer)

{

auto size = Count();

if (size == 0 || size == 1)

{

return this;

}

auto arr = ToReferenceArrayInHeap(size);

QuickSort<T \*, true>::Sort(arr, arr+(size-1), comparer);

// Converet the reference Array into the List

auto result = (ImmutableList<T>\*)(arr[0] - offsetof(ImmutableList<T>, value));

auto current = result;

for(size\_t i = 1; i<size; ++i)

{

current->next = (ImmutableList<T>\*)(arr[i] - offsetof(ImmutableList<T>, value));

current = current->next;

}

current->next = nullptr;

delete []arr;

return result;

}

// Info: Sort the list by the given reference comparer in reverse order <Modifies existing list such that it is ordered using comparer>

// Parameters: comparer - comparer to sort with

ImmutableList<T> \* ReverseSortCurrentList(regex::Comparer<T \*> \* comparer)

{

auto size = Count();

if (size == 0 || size == 1)

{

return this;

}

auto arr = ToReferenceArrayInHeap(size);

QuickSort<T \*, true>::Sort(arr, arr+(size-1), comparer);

// Converet the reference Array into the List

auto result = (ImmutableList<T>\*)(arr[0] - offsetof(ImmutableList<T>, value));

result->next = nullptr;

for(size\_t i = 1; i<size; ++i)

{

auto current = (ImmutableList<T>\*)(arr[i] - offsetof(ImmutableList<T>, value));

current->next = result;

result = current;

}

delete []arr;

return result;

}

// Info: Return true if the list is empty.

bool IsEmpty()

{

return this==Empty();

}

// Info: Return a list containing the given single value

// Parameters: value - the value

static ImmutableList<T> \* OfSingle(T value, ArenaAllocator \* a)

{

return Anew(a, ImmutableList, value, nullptr);

}

// Info: Group elements that are equal to each other into sublists.

// NOTE: This function is N^2 and is currently only suitable for small

// lists. If larger lists are needed, please change this function

// to group into an intermediate hash table.

// Parameters: equals - the value

template<class FEquals>

ImmutableList<ImmutableList<T> \*> \* GroupBy(FEquals equals, ArenaAllocator \* a)

{

auto out = ImmutableList<ImmutableList<T>\*>::Empty();

auto currentIn = this;

while(!currentIn->IsEmpty())

{

auto currentOut = out;

bool found = false;

while(!currentOut->IsEmpty())

{

ImmutableList<T>\* currentOutValue = currentOut->First();

if (equals(currentOutValue->First(),currentIn->value))

{

found = true;

currentOut->First() = currentOutValue->Prepend(currentIn->First(),a);

break;

}

currentOut = currentOut->GetTail();

}

if (!found)

{

out = out->Prepend(OfSingle(currentIn->value,a),a);

}

currentIn = currentIn->next;

}

return out;

}

// Info: Create groups where FEquals returns true for sets of adjacent element. If the list is sorted, this is equivalent to GroupBy.

// Parameters: equals - the value

template<class FEquals>

ImmutableList<ImmutableList<T>\*> \* GroupByAdjacentOnCurrentList(FEquals equals, ArenaAllocator \* a)

{

auto out = ImmutableList<ImmutableList<T>\*>::Empty();

auto tail = out;

if(!IsEmpty())

{

auto current = this;

auto set = current;

auto setTail = set;

while(current)

{

if (!equals(current->First(), set->First()))

{

Assert(!set->IsEmpty() && !setTail->IsEmpty());

setTail->next = nullptr;

out = out->Append(set, a, &tail);

set = current;

}

setTail = current;

current = current->GetTail();

}

Assert(!set->IsEmpty() && !setTail->IsEmpty());

setTail->next = nullptr;

out = out->Append(set, a, &tail);

}

return out;

}

};

// Info: Return a list containing the given single value

// Parameters: value - the value

template<typename T>

ImmutableList<const T \*> \* ToImmutableList(const T \* value, ArenaAllocator \* a)

{

return ImmutableList<const T \*>::OfSingle(value,a);

}

template<class T, class TAllocator, int chunkSize = 4>

class ImmutableArrayBuilder

{

private:

TAllocator \*allocator;

T \* arrayData;

size\_t currentIndex;

size\_t arraySize;

bool fAutoDeleteArray;

public:

ImmutableArrayBuilder(TAllocator \*alloc) : allocator(alloc), arrayData(nullptr), currentIndex(0), arraySize(0), fAutoDeleteArray(true)

{

}

~ImmutableArrayBuilder()

{

if (fAutoDeleteArray)

{

AllocatorDeleteArray(TAllocator, allocator, arraySize, arrayData);

}

}

void Append(T newEntry)

{

// Genreate new chunk

if (currentIndex == arraySize)

{

T \* newChunk = AllocatorNewArray(TAllocator, allocator, T, arraySize + chunkSize);

memcpy\_s(newChunk, (arraySize + chunkSize) \* sizeof(T), arrayData, arraySize \* sizeof(T));

if (arrayData)

{

AllocatorDeleteArray(TAllocator, allocator, arraySize, arrayData);

}

arrayData = newChunk;

arraySize = arraySize + chunkSize;

}

Assert(arrayData != nullptr);

arrayData[currentIndex] = newEntry;

currentIndex++;

}

size\_t GetCount()

{

return currentIndex;

}

T \* Get()

{

return arrayData;

}

void DisableAutoDelete()

{

fAutoDeleteArray = false;

}

};

template<int chunkSize>

class ImmutableStringBuilder

{

private:

struct StringChunk

{

LPCWSTR dataPtr[chunkSize];

StringChunk \*next;

StringChunk() : next(nullptr)

{

}

};

StringChunk \*head;

StringChunk \*tail;

int currentIndex;

size\_t stringSize;

// tracking allocated strings based on non-Append(String) calls

struct AllocatedStringChunk

{

LPCWSTR dataPtr;

AllocatedStringChunk \*next;

AllocatedStringChunk() : next(nullptr)

{

}

};

AllocatedStringChunk\* allocatedStringChunksHead;

public:

ImmutableStringBuilder() : head(nullptr), tail(nullptr), currentIndex(chunkSize), stringSize(1), allocatedStringChunksHead(nullptr)

{

}

~ImmutableStringBuilder()

{

// unallocate strings

AllocatedStringChunk\* allocatedStringChunk = this->allocatedStringChunksHead;

AllocatedStringChunk\* nextAllocatedStringChunk;

while (allocatedStringChunk != nullptr)

{

nextAllocatedStringChunk = allocatedStringChunk->next;

delete[] allocatedStringChunk->dataPtr;

delete allocatedStringChunk;

allocatedStringChunk = nextAllocatedStringChunk;

}

while (head != nullptr)

{

auto current = head;

head = head->next;

delete current;

}

}

void AppendInt32(int32 value);

void AppendUInt64(uint64 value);

void AppendWithCopy(\_In\_z\_ LPCWSTR str);

void AppendBool(bool value)

{

this->Append(value ? L"true" : L"false");

}

void Append(LPCWSTR str)

{

// silently ignore nullptr usage pattern, to avoid cluttering codebase

if (str == nullptr)

return;

size\_t newStrSize = stringSize + wcslen(str);

if (newStrSize < stringSize)

{

// Overflow

Js::Throw::OutOfMemory();

}

// Genreate new chunk

if (currentIndex == chunkSize)

{

StringChunk \*newChunk = new StringChunk();

IfNullThrowOutOfMemory(newChunk);

if (tail == nullptr)

{

Assert(head == nullptr);

head = newChunk;

tail = newChunk;

}

else

{

tail->next = newChunk;

tail = newChunk;

}

currentIndex = 0;

}

Assert(tail != nullptr);

tail->dataPtr[currentIndex] = str;

currentIndex++;

stringSize = newStrSize;

}

template<class TAllocator>

LPCWSTR Get(TAllocator \*allocator)

{

wchar\_t \*str = AllocatorNewArray(TAllocator, allocator, wchar\_t, stringSize);

str[0] = L'\0';

auto current = head;

while (current != nullptr)

{

int lastIndex = (current == tail) ? currentIndex : chunkSize;

for (int index = 0; index < lastIndex; index++)

{

wcscat\_s(str, stringSize, current->dataPtr[index]);

}

current = current->next;

}

return str;

}

// Free a string returned by Get()

template<class TAllocator>

void FreeString(LPCWSTR str)

{

ImmutableList<chunkSize>::FreeString(allocator, str, stringSize);

}

template<class TAllocator>

static void FreeString(TAllocator \*allocator, LPCWSTR str, size\_t strLength)

{

AssertMsg(allocator != nullptr, "allocator != nullptr");

AssertMsg(str != nullptr, "str != nullptr");

AllocatorDeleteArray(TAllocator, allocator, strLength, str);

}

};

typedef ImmutableStringBuilder<8> DefaultImmutableStringBuilder;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

#include "DataStructures\CharacterBuffer.h"

#include "DataStructures\InternalString.h"

namespace Js

{

InternalString::InternalString(const wchar\_t\* content, charcount\_t length, unsigned char offset):

m\_content(content),

m\_charLength(length),

m\_offset(offset)

{

AssertMsg(length < INT\_MAX, "Length should be a valid string length");

}

// This will make a copy of the entire buffer

InternalString \*InternalString::New(ArenaAllocator\* alloc, const wchar\_t\* content, charcount\_t length)

{

size\_t bytelength = sizeof(wchar\_t) \* length;

DWORD\* allocbuffer = (DWORD\*)alloc->Alloc(sizeof(DWORD) + bytelength + sizeof(wchar\_t));

allocbuffer[0] = (DWORD) bytelength;

wchar\_t\* buffer = (wchar\_t\*)(allocbuffer+1);

js\_memcpy\_s(buffer, bytelength, content, bytelength);

buffer[length] = L'\0';

InternalString\* newInstance = Anew(alloc, InternalString, buffer, length);

return newInstance;

}

// This will make a copy of the entire buffer

// Allocated using recycler memory

InternalString \*InternalString::New(Recycler\* recycler, const wchar\_t\* content, charcount\_t length)

{

size\_t bytelength = sizeof(wchar\_t) \* length;

// Allocate 3 extra bytes, two for the first DWORD with the size, the third for the null character

// This is so that we can pretend that internal strings are BSTRs for purposes of clients who want to use

// it as thus

const unsigned char offset = sizeof(DWORD)/sizeof(wchar\_t);

InternalString\* newInstance = RecyclerNewPlusLeaf(recycler, bytelength + (sizeof(DWORD) + sizeof(wchar\_t)), InternalString, nullptr, length, offset);

DWORD\* allocbuffer = (DWORD\*) (newInstance + 1);

allocbuffer[0] = (DWORD) bytelength;

wchar\_t\* buffer = (wchar\_t\*)(allocbuffer + 1);

js\_memcpy\_s(buffer, bytelength, content, bytelength);

buffer[length] = L'\0';

newInstance->m\_content = (const wchar\_t\*) allocbuffer;

return newInstance;

}

// This will only store the pointer and length, not making a copy of the buffer

InternalString \*InternalString::NewNoCopy(ArenaAllocator\* alloc, const wchar\_t\* content, charcount\_t length)

{

InternalString\* newInstance = Anew(alloc, InternalString, const\_cast<wchar\_t\*> (content), length);

return newInstance;

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

class InternalString

{

charcount\_t m\_charLength;

unsigned char m\_offset;

const wchar\_t\* m\_content;

public:

InternalString() : m\_charLength(0), m\_content(NULL), m\_offset(0) { };

InternalString(const wchar\_t\* content, charcount\_t charLength, unsigned char offset = 0);

static InternalString\* New(ArenaAllocator\* alloc, const wchar\_t\* content, charcount\_t length);

static InternalString\* New(Recycler\* recycler, const wchar\_t\* content, charcount\_t length);

static InternalString\* NewNoCopy(ArenaAllocator\* alloc, const wchar\_t\* content, charcount\_t length);

inline charcount\_t GetLength() const

{

return m\_charLength;

}

inline const wchar\_t\* GetBuffer() const

{

return m\_content + m\_offset;

}

};

struct InternalStringComparer

{

\_\_inline static bool Equals(InternalString const& str1, InternalString const& str2)

{

return str1.GetLength() == str2.GetLength() &&

JsUtil::CharacterBuffer<WCHAR>::StaticEquals(str1.GetBuffer(), str2.GetBuffer(), str1.GetLength());

}

\_\_inline static hash\_t GetHashCode(InternalString const& str)

{

return JsUtil::CharacterBuffer<WCHAR>::StaticGetHashCode(str.GetBuffer(), str.GetLength());

}

};

}

template<>

struct DefaultComparer<Js::InternalString> : public Js::InternalStringComparer {};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

#include "DataStructures\CharacterBuffer.h"

#include "DataStructures\InternalStringNoCaseComparer.h"

namespace JsUtil

{

bool NoCaseComparer<JsUtil::CharacterBuffer<WCHAR>>::Equals(JsUtil::CharacterBuffer<WCHAR> const& s1, JsUtil::CharacterBuffer<WCHAR> const& s2)

{

return (s1.GetLength() == s2.GetLength()) && (NoCaseComparer<JsUtil::CharacterBuffer<WCHAR>>::Compare(s1, s2)==0);

}

uint NoCaseComparer<JsUtil::CharacterBuffer<WCHAR>>::GetHashCode(JsUtil::CharacterBuffer<WCHAR> const& s1)

{

const wchar\_t\* s = s1.GetBuffer();

size\_t length = s1.GetLength();

uint hash = 0;

for (size\_t i = 0; i < length; i++)

{

hash = \_rotl(hash, 7);

hash ^= tolower(s[i]);

}

return ((hash & 0x7fffffff) << 1) | 1;

}

int NoCaseComparer<JsUtil::CharacterBuffer<WCHAR>>::Compare(JsUtil::CharacterBuffer<WCHAR> const& s1, JsUtil::CharacterBuffer<WCHAR> const& s2)

{

if (s1.GetLength() != s2.GetLength()) return +1;

int count = s1.GetLength();

const wchar\_t\* buf1 = s1.GetBuffer();

const wchar\_t\* buf2 = s2.GetBuffer();

for (int i=0; i < count; i++)

{

if (tolower(buf1[i]) != tolower(buf2[i]))

{

return (buf1[i] < buf2[i] ? -1 : +1);

}

}

return (0);

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template <typename T>

class NoCaseComparer;

template <>

class NoCaseComparer<JsUtil::CharacterBuffer<WCHAR>>

{

public:

static bool Equals(JsUtil::CharacterBuffer<WCHAR> const& x, JsUtil::CharacterBuffer<WCHAR> const& y);

static uint GetHashCode(JsUtil::CharacterBuffer<WCHAR> const& i);

private:

static int Compare(JsUtil::CharacterBuffer<WCHAR> const& x, JsUtil::CharacterBuffer<WCHAR> const& y);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

#include "DataStructures\Interval.h"

namespace regex

{

bool Interval::Includes(int value) const

{

return (begin <= value) && (end >= value);

}

bool Interval::Includes(Interval other) const

{

return (Includes(other.Begin()) && (Includes(other.End())));

}

int Interval::CompareTo(Interval other)

{

if (begin < other.begin)

{

return -1;

}

else if (begin == other.begin)

{

if (end < other.end)

{

return -1;

}

else if (end == other.end)

{

return 0;

}

else

{

return 1;

}

}

else

{

return 1;

}

}

int Interval::Compare(Interval x, Interval y)

{

return x.CompareTo(y);

}

bool Interval::Equals(Interval other)

{

return CompareTo(other) == 0;

}

bool Interval::Equals(Interval x, Interval y)

{

return x.CompareTo(y) == 0;

}

int Interval::GetHashCode()

{

return \_rotl(begin, 7) ^ end;

}

int Interval::GetHashCode(Interval item)

{

return item.GetHashCode();

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace regex

{

struct Interval

{

int begin;

int end;

public:

Interval(): begin(0), end(0)

{

}

Interval(int start) : begin(start), end(start)

{

}

Interval(int start, int end) : begin(start), end(end)

{

}

inline int Begin() { return begin; }

inline void Begin(int value) { begin = value; }

inline int End() { return end; }

inline void End(int value) { end = value; }

bool Includes(int value) const;

bool Includes(Interval other) const;

int CompareTo(Interval other);

static int Compare(Interval x, Interval y);

bool Equals(Interval other);

static bool Equals(Interval x, Interval y);

int GetHashCode();

static int GetHashCode(Interval item);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template <class TKey, class TValue> struct KeyValuePair

{

private:

TKey key;

TValue value;

public:

KeyValuePair()

{

}

KeyValuePair(TKey key, TValue value)

{

this->key = key;

this->value = value;

}

TKey Key() { return key; }

const TKey Key() const { return key; }

TValue Value() { return value; }

const TValue Value() const { return value; }

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

template <class T>

struct LargeStackBlock {

T\* items;

int index;

int itemCount;

static LargeStackBlock<T>\* Make(ArenaAllocator\* alloc,int itemCount) {

LargeStackBlock<T>\* block = AnewStruct(alloc, LargeStackBlock<T>);

block->itemCount=itemCount;

block->items = AnewArray(alloc, T, itemCount);

block->index=0;

return block;

}

BOOL Full() { return index>=itemCount; }

BOOL Empty() { return index==0; }

void Push(T item) {

AssertMsg(!Full(),"can't push to full stack block");

items[index++]=item;

}

T Pop() {

AssertMsg(!Empty(),"can't pop empty stack block");

index--;

return items[index];

}

};

template <class T>

class LargeStack {

SList<LargeStackBlock<T>\*>\* blockStack;

static const int BlockSize=8;

static const int GrowSize=128;

ArenaAllocator\* alloc;

LargeStack(ArenaAllocator\* alloc) : alloc(alloc) {

blockStack=Anew(alloc,SList<LargeStackBlock<T>\*>,alloc);

blockStack->Push(LargeStackBlock<T>::Make(alloc,BlockSize));

}

public:

static LargeStack \* New(ArenaAllocator\* alloc)

{

return Anew(alloc, LargeStack, alloc);

}

void Push(T item) {

LargeStackBlock<T>\* top=blockStack->Top();

if (top->Full()) {

top=LargeStackBlock<T>::Make(alloc,top->itemCount+GrowSize);

blockStack->Push(top);

}

top->Push(item);

}

BOOL Empty() {

LargeStackBlock<T>\* top=blockStack->Top();

if (top->Empty()) {

if (blockStack->HasOne()) {

// Avoid popping the last empty block to reduce freelist overhead.

return true;

}

blockStack->Pop();

return blockStack->Empty();

}

else return false;

}

T Pop() {

LargeStackBlock<T>\* top=blockStack->Top();

if (top->Empty()) {

blockStack->Pop();

AssertMsg(!blockStack->Empty(),"can't pop empty block stack");

top=blockStack->Top();

}

return top->Pop();

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template <

class TKey,

class TValue,

class SizePolicy = PowerOf2SizePolicy,

template <typename ValueOrKey> class Comparer = DefaultComparer,

template <typename K, typename V> class Entry = SimpleDictionaryEntry,

class LockPolicy = Js::DefaultListLockPolicy, // Controls lock policy for read/map/write/add/remove items

class SyncObject = CriticalSection

>

struct LeafValueDictionary

{

typedef JsUtil::SynchronizedDictionary<TKey,

TValue,

RecyclerLeafAllocator,

SizePolicy,

Comparer,

Entry,

LockPolicy,

SyncObject

> Type;

};

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template <typename TAllocator>

class LineOffsetCache

{

public:

// Stores line offset information for a line in the source info. The index of this

// item in the LineOffsetCacheList determines which line it is.

struct LineOffsetCacheItem

{

// The character offset of where the line begins starting from the start of the source info.

charcount\_t characterOffset;

// The byte offset of where the line begins starting from the start of the source info's character buffer (UTF8).

charcount\_t byteOffset;

};

private:

typedef List<LineOffsetCacheItem, TAllocator, true /\*isLeaf\*/> LineOffsetCacheList;

typedef ReadOnlyList<LineOffsetCacheItem> LineOffsetCacheReadOnlyList;

public:

static int FindLineForCharacterOffset(

\_In\_z\_ LPCUTF8 sourceStartCharacter,

\_In\_z\_ LPCUTF8 sourceEndCharacter,

charcount\_t &inOutLineCharOffset,

charcount\_t &inOutByteOffset,

charcount\_t characterOffset)

{

int lastLine = 0;

while (FindNextLine(sourceStartCharacter, sourceEndCharacter, inOutLineCharOffset, inOutByteOffset, characterOffset))

{

lastLine++;

}

return lastLine;

}

LineOffsetCache(TAllocator\* allocator,

\_In\_z\_ LPCUTF8 sourceStartCharacter,

\_In\_z\_ LPCUTF8 sourceEndCharacter,

charcount\_t startingCharacterOffset = 0,

charcount\_t startingByteOffset = 0) :

allocator(allocator),

isCacheBuilt(false)

{

AssertMsg(allocator, "An allocator must be supplied to the cache for allocation of items.");

LineOffsetCacheList \*list = AllocatorNew(TAllocator, allocator, LineOffsetCacheList, allocator);

this->lineOffsetCacheList = list;

this->BuildCache(list, sourceStartCharacter, sourceEndCharacter, startingCharacterOffset, startingByteOffset);

}

LineOffsetCache(TAllocator \*allocator,

\_In\_reads\_(numberOfLines) const LineOffsetCacheItem \*lines,

\_\_in int numberOfLines) :

allocator(allocator),

isCacheBuilt(false)

{

this->lineOffsetCacheList = LineOffsetCacheReadOnlyList::New(allocator, (LineOffsetCacheItem \*)lines, numberOfLines);

}

~LineOffsetCache()

{

if (this->lineOffsetCacheList != nullptr)

{

this->lineOffsetCacheList->Delete();

}

}

// outLineCharOffset - The character offset of the start of the line returned

int GetLineForCharacterOffset(charcount\_t characterOffset, charcount\_t \*outLineCharOffset, charcount\_t \*outByteOffset)

{

Assert(this->lineOffsetCacheList->Count() > 0);

// The list is sorted, so binary search to find the line info.

int closestIndex = -1;

int minRange = INT\_MAX;

this->lineOffsetCacheList->BinarySearch([&](const LineOffsetCacheItem& item, int index)

{

int offsetRange = characterOffset - item.characterOffset;

if (offsetRange >= 0)

{

if (offsetRange < minRange)

{

// There are potentially many lines with starting offsets greater than the one we're searching

// for. As a result, we should track which index we've encountered so far that is the closest

// to the offset we're looking for without going under. This will find the line that contains

// the offset.

closestIndex = index;

minRange = offsetRange;

}

// Search lower to see if we can find a closer index.

return -1;

}

else

{

// Search higher to get into a range that is greater than the offset.

return 1;

}

// Note that we purposely don't return 0 (==) here. We want the search to end in failure (-1) because

// we're searching for the closest element, not necessarily an exact element offset. Exact offsets

// are possible when the offset we're searching for is the first character of the line, but that will

// be handled by the if statement above.

});

if (closestIndex >= 0)

{

LineOffsetCacheItem lastItem = this->lineOffsetCacheList->Item(closestIndex);

if (outLineCharOffset != nullptr)

{

\*outLineCharOffset = lastItem.characterOffset;

}

if (outByteOffset != nullptr)

{

\*outByteOffset = lastItem.byteOffset;

}

}

return closestIndex;

}

charcount\_t GetCharacterOffsetForLine(charcount\_t line, charcount\_t \*outByteOffset) const

{

AssertMsg(line < this->GetLineCount(), "Invalid line value passed in.");

LineOffsetCacheItem item = this->lineOffsetCacheList->Item(line);

if (outByteOffset != nullptr)

{

\*outByteOffset = item.byteOffset;

}

return item.characterOffset;

}

uint32 GetLineCount() const

{

AssertMsg(this->lineOffsetCacheList != nullptr, "The list was either not set from the ByteCode or not created.");

return this->lineOffsetCacheList->Count();

}

const LineOffsetCacheItem\* GetItems()

{

return this->lineOffsetCacheList->GetBuffer();

}

private:

static bool FindNextLine(\_In\_z\_ LPCUTF8 &currentSourcePosition, \_In\_z\_ LPCUTF8 sourceEndCharacter, charcount\_t &inOutCharacterOffset, charcount\_t &inOutByteOffset, charcount\_t maxCharacterOffset = MAXUINT32)

{

charcount\_t currentCharacterOffset = inOutCharacterOffset;

charcount\_t currentByteOffset = inOutByteOffset;

utf8::DecodeOptions options = utf8::doAllowThreeByteSurrogates;

while (currentSourcePosition < sourceEndCharacter)

{

LPCUTF8 previousCharacter = currentSourcePosition;

// Decode from UTF8 to wide char. Note that Decode will advance the current character by 1 at least.

wchar\_t decodedCharacter = utf8::Decode(currentSourcePosition, sourceEndCharacter, options);

bool wasLineEncountered = false;

switch (decodedCharacter)

{

case L'\r':

// Check if the next character is a '\n'. If so, consume that character as well

// (consider as one line).

if (\*currentSourcePosition == '\n')

{

++currentSourcePosition;

++currentCharacterOffset;

}

// Intentional fall-through.

case L'\n':

case 0x2028:

case 0x2029:

// Found a new line.

wasLineEncountered = true;

break;

}

// Move to the next character offset.

++currentCharacterOffset;

// Count the current byte offset we're at in the UTF-8 buffer.

// The character size can be > 1 for unicode characters.

currentByteOffset += static\_cast<int>(currentSourcePosition - previousCharacter);

if (wasLineEncountered)

{

inOutCharacterOffset = currentCharacterOffset;

inOutByteOffset = currentByteOffset;

return true;

}

else if (currentCharacterOffset >= maxCharacterOffset)

{

return false;

}

}

return false;

}

// Builds the cache of line offsets from the passed in source.

void BuildCache(\_In\_ LineOffsetCacheList \*list, \_In\_z\_ LPCUTF8 sourceStartCharacter,

\_In\_z\_ LPCUTF8 sourceEndCharacter,

charcount\_t startingCharacterOffset = 0,

charcount\_t startingByteOffset = 0)

{

AssertMsg(sourceStartCharacter, "The source start character passed in is null.");

AssertMsg(sourceEndCharacter, "The source end character passed in is null.");

AssertMsg(sourceStartCharacter <= sourceEndCharacter, "The source start character should not be beyond the source end character.");

AssertMsg(!this->isCacheBuilt, "The cache is already built.");

// Add the first line in the cache list.

this->AddLine(list, startingCharacterOffset, startingByteOffset);

while (FindNextLine(sourceStartCharacter, sourceEndCharacter, startingCharacterOffset, startingByteOffset))

{

this->AddLine(list, startingCharacterOffset, startingByteOffset);

}

isCacheBuilt = true;

}

// Tracks a new line offset in the cache.

void AddLine(\_In\_ LineOffsetCacheList \*list, int characterOffset, int byteOffset)

{

AssertMsg(characterOffset >= 0, "The character offset is invalid.");

AssertMsg(byteOffset >= 0, "The byte offset is invalid.");

LineOffsetCacheItem item;

item.characterOffset = characterOffset;

item.byteOffset = byteOffset;

list->Add(item);

#if DBG

if (list->Count() > 1)

{

// Ensure that the list remains sorted during insertion.

LineOffsetCacheItem previousItem = list->Item(list->Count() - 2);

AssertMsg(item.characterOffset > previousItem.characterOffset, "The character offsets must be inserted in increasing order per line.");

AssertMsg(item.byteOffset > previousItem.byteOffset, "The byte offsets must be inserted in increasing order per line.");

}

#endif // DBG

}

private:

TAllocator\* allocator;

// Line offset cache list used for quickly finding line/column offsets.

LineOffsetCacheReadOnlyList\* lineOffsetCacheList;

bool isCacheBuilt;

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

template <class TListType, bool clearOldEntries>

class CopyRemovePolicy;

template <typename TListType, bool clearOldEntries>

class FreeListedRemovePolicy;

template <typename TListType, bool clearOldEntries>

class WeakRefFreeListedRemovePolicy;

};

namespace JsUtil

{

template <

class T,

class TAllocator = Recycler,

template <typename Value> class TComparer = DefaultComparer>

class ReadOnlyList

{

public:

typedef TComparer<T> TComparerType;

protected:

T\* buffer;

int count;

TAllocator\* alloc;

ReadOnlyList(TAllocator\* alloc)

: buffer(nullptr),

count(0),

alloc(alloc)

{

}

public:

virtual bool IsReadOnly() const

{

return true;

}

virtual void Delete()

{

AllocatorDelete(TAllocator, alloc, this);

}

const T\* GetBuffer() const

{

return this->buffer;

}

template<class TList>

bool Equals(TList list)

{

CompileAssert(sizeof(T) == sizeof(\*list->GetBuffer()));

return list->Count() == this->Count()

&& memcmp(this->buffer, list->GetBuffer(), sizeof(T)\* this->Count()) == 0;

}

template<class TAllocator>

static ReadOnlyList \* New(TAllocator\* alloc, \_\_in\_ecount(count) T\* buffer, int count)

{

return AllocatorNew(TAllocator, alloc, ReadOnlyList, buffer, count, alloc);

}

ReadOnlyList(\_\_in\_ecount(count) T\* buffer, int count, TAllocator\* alloc)

: buffer(buffer),

count(count),

alloc(alloc)

{

}

virtual ~ReadOnlyList()

{

}

int Count() const

{

return count;

}

bool Empty() const

{

return Count() == 0;

}

// Gets the count of items using the specified criteria for considering an item.

template <typename TConditionalFunction>

int CountWhere(TConditionalFunction function) const

{

int conditionalCount = 0;

for (int i = 0; i < this->count; ++i)

{

if (function(this->buffer[i]))

{

++conditionalCount;

}

}

return conditionalCount;

}

const T& Item(int index) const

{

Assert(index >= 0 && index < count);

return buffer[index];

}

bool Contains(const T& item) const

{

for (int i = 0; i < count; i++)

{

if (TComparerType::Equals(item, buffer[i]))

{

return true;

}

}

return false;

}

// Checks if any of the elements satisfy the condition in the passed in function.

template <typename TConditionalFunction>

bool Any(TConditionalFunction function)

{

for (int i = 0; i < count; ++i)

{

if (function(this->buffer[i]))

{

return true;

}

}

return false;

}

// Checks if all of the elements satisfy the condition in the passed in function.

template <typename TConditionalFunction>

bool All(TConditionalFunction function)

{

for (int i = 0; i < count; ++i)

{

if (!function(this->buffer[i]))

{

return false;

}

}

return true;

}

// Performs a binary search on a range of elements in the list (assumes the list is sorted).

template <typename TComparisonFunction>

int BinarySearch(TComparisonFunction compare, int fromIndex, int toIndex)

{

AssertMsg(fromIndex >= 0, "Invalid starting index for binary searching.");

AssertMsg(toIndex < this->count, "Invalid ending index for binary searching.");

while (fromIndex <= toIndex)

{

int midIndex = fromIndex + (toIndex - fromIndex) / 2;

T item = this->Item(midIndex);

int compareResult = compare(item, midIndex);

if (compareResult > 0)

{

toIndex = midIndex - 1;

}

else if (compareResult < 0)

{

fromIndex = midIndex + 1;

}

else

{

return midIndex;

}

}

return -1;

}

// Performs a binary search on the elements in the list (assumes the list is sorted).

template <typename TComparisonFunction>

int BinarySearch(TComparisonFunction compare)

{

return BinarySearch<TComparisonFunction>(compare, 0, this->Count() - 1);

}

};

template <

class T,

class TAllocator = Recycler,

bool isLeaf = false,

template <class TListType, bool clearOldEntries> class TRemovePolicy = Js::CopyRemovePolicy,

template <typename Value> class TComparer = DefaultComparer>

class List : public ReadOnlyList<T, TAllocator, TComparer>

{

public:

typedef ReadOnlyList<T, TAllocator, TComparer> ParentType;

typedef T TElementType; // For TRemovePolicy

static const int DefaultIncrement = 4;

private:

typedef List<T, TAllocator, isLeaf, TRemovePolicy, TComparer> TListType;

friend TRemovePolicy<TListType, true>;

typedef TRemovePolicy<TListType, true /\* clearOldEntries \*/> TRemovePolicyType;

typedef ListTypeAllocatorFunc<TAllocator, isLeaf> AllocatorInfo;

int length;

int increment;

TRemovePolicyType removePolicy;

template <bool isLeaf> T \* AllocArray(int size);

template <> T \* AllocArray<true>(int size) { return AllocatorNewArrayLeaf(TAllocator, alloc, T, size); }

template <> T \* AllocArray<false>(int size) { return AllocatorNewArray(TAllocator, alloc, T, size); }

PREVENT\_COPY(List); // Disable copy constructor and operator=

public:

virtual bool IsReadOnly() const override

{

return false;

}

virtual void Delete() override

{

AllocatorDelete(TAllocator, alloc, this);

}

void EnsureArray()

{

EnsureArray(0);

}

void EnsureArray(int32 requiredCapacity)

{

if (buffer == nullptr)

{

int32 newSize = max(requiredCapacity, increment);

buffer = AllocArray<isLeaf>(newSize);

count = 0;

length = newSize;

}

else if (count == length || requiredCapacity > length)

{

int32 newLength = 0, newBufferSize = 0, oldBufferSize = 0;

if (Int32Math::Add(length, 1u, &newLength)

|| Int32Math::Shl(newLength, 1u, &newLength))

{

JsUtil::ExternalApi::RaiseOnIntOverflow();

}

newLength = max(requiredCapacity, newLength);

if (Int32Math::Mul(sizeof(T), newLength, &newBufferSize)

|| Int32Math::Mul(sizeof(T), length, &oldBufferSize))

{

JsUtil::ExternalApi::RaiseOnIntOverflow();

}

T\* newbuffer = AllocArray<isLeaf>(newLength);

js\_memcpy\_s(newbuffer, newBufferSize, buffer, oldBufferSize);

auto freeFunc = AllocatorInfo::GetFreeFunc();

AllocatorFree(this->alloc, freeFunc, buffer, oldBufferSize);

length = newLength;

buffer = newbuffer;

}

}

template<class T>

void Copy(const T\* list)

{

CompileAssert(sizeof(TElementType) == sizeof(typename T::TElementType));

if (list->Count() > 0)

{

this->EnsureArray(list->Count());

js\_memcpy\_s(this->buffer, UInt32Math::Mul(sizeof(TElementType), this->length), list->GetBuffer(), UInt32Math::Mul(sizeof(TElementType), list->Count()));

}

this->count = list->Count();

}

static List \* New(TAllocator \* alloc, int increment = DefaultIncrement)

{

return AllocatorNew(TAllocator, alloc, List, alloc, increment);

}

List(TAllocator\* alloc, int increment = DefaultIncrement) : increment(increment), removePolicy(this), ReadOnlyList(alloc)

{

buffer = nullptr;

count = 0;

length = 0;

}

virtual ~List() override

{

this->Reset();

}

TAllocator \* GetAllocator() const

{

return this->alloc;

}

const T& Item(int index) const

{

return ReadOnlyList::Item(index);

}

T& Item(int index)

{

Assert(index >= 0 && index < count);

return buffer[index];

}

T& Last()

{

Assert(count >= 1);

return this->Item(count - 1);

}

// Finds the last element that satisfies the condition in the passed in function.

// Returns true if the element was found; false otherwise.

template <typename TConditionalFunction>

bool Last(TConditionalFunction function, T& outElement)

{

for (int i = count - 1; i >= 0; --i)

{

if (function(this->buffer[i]))

{

outElement = this->buffer[i];

return true;

}

}

return false;

}

void Item(int index, const T& item)

{

Assert(index >= 0 && index < count);

buffer[index] = item;

}

void SetItem(int index, const T& item)

{

EnsureArray(index + 1);

buffer[index] = item;

count = max(count, index + 1);

}

void SetExistingItem(int index, const T& item)

{

Item(index, item);

}

bool IsItemValid(int index)

{

return removePolicy.IsItemValid(this, index);

}

int SetAtFirstFreeSpot(const T& item)

{

int indexToSetAt = removePolicy.GetFreeItemIndex(this);

if (indexToSetAt == -1)

{

return Add(item);

}

buffer[indexToSetAt] = item;

return indexToSetAt;

}

int Add(const T& item)

{

EnsureArray();

buffer[count] = item;

int pos = count;

count++;

return pos;

}

int32 AddRange(\_\_readonly \_In\_reads\_(count) const T\* items, int32 count)

{

Assert(items != nullptr);

Assert(count > 0);

int32 requiredSize = 0, availableByteSpace = 0, givenBufferSize = 0;

if (Int32Math::Add(this->count, count, &requiredSize))

{

JsUtil::ExternalApi::RaiseOnIntOverflow();

}

EnsureArray(requiredSize);

if (Int32Math::Sub(this->length, this->count, &availableByteSpace)

|| Int32Math::Mul(sizeof(T), availableByteSpace, &availableByteSpace)

|| Int32Math::Mul(sizeof(T), count, &givenBufferSize))

{

JsUtil::ExternalApi::RaiseOnIntOverflow();

}

js\_memcpy\_s(buffer + this->count, availableByteSpace, items, givenBufferSize);

this->count = requiredSize;

return requiredSize; //Returns count

}

void AddRange(TListType const& list)

{

list.Map([this](int index, T const& item)

{

this->Add(item);

});

}

// Trims the end of the list

template <bool weaklyRefItems>

T CompactEnd()

{

while (count != 0)

{

AnalysisAssert(!weaklyRefItems || (buffer[count - 1] != nullptr));

if ((weaklyRefItems ? buffer[count - 1]->Get() != nullptr : buffer[count - 1] != nullptr))

{

return buffer[count - 1];

}

count--;

buffer[count] = nullptr;

}

return nullptr;

}

void Remove(const T& item)

{

removePolicy.Remove(this, item);

}

T RemoveAtEnd()

{

Assert(count >= 1);

T item = this->Item(count - 1);

RemoveAt(count - 1);

return item;

}

void RemoveAt(int index)

{

removePolicy.RemoveAt(this, index);

}

void Clear()

{

count = 0;

}

void ClearAndZero()

{

if(count == 0)

{

return;

}

memset(buffer, 0, count \* sizeof(T));

Clear();

}

void Sort()

{

// We can call QSort only if the remove policy for this list is CopyRemovePolicy

CompileAssert((IsSame<TRemovePolicyType, Js::CopyRemovePolicy<TListType, false> >::IsTrue) ||

(IsSame<TRemovePolicyType, Js::CopyRemovePolicy<TListType, true> >::IsTrue));

if(count)

{

JsUtil::QuickSort<T, TComparerType>::Sort(buffer, buffer + (count-1));

}

}

void Sort(int(\_\_cdecl \* \_PtFuncCompare)(void \*, const void \*, const void \*), void \*\_Context)

{

// We can call QSort only if the remove policy for this list is CopyRemovePolicy

CompileAssert((IsSame<TRemovePolicyType, Js::CopyRemovePolicy<TListType, false> >::IsTrue) ||

(IsSame<TRemovePolicyType, Js::CopyRemovePolicy<TListType, true> >::IsTrue));

if (count)

{

qsort\_s(buffer, count, sizeof(T), \_PtFuncCompare, \_Context);

}

}

template<class DebugSite, class TMapFunction>

HRESULT Map(DebugSite site, TMapFunction map) const // external debugging version

{

return Js::Map(site, this->buffer, count, map);

}

template<class TMapFunction>

bool MapUntil(TMapFunction map) const

{

return MapUntilFrom(0, map);

}

template<class TMapFunction>

bool MapUntilFrom(int start, TMapFunction map) const

{

for (int i = start; i < count; i++)

{

if (TRemovePolicyType::IsItemValid(this->buffer[i]))

{

if (map(i, this->buffer[i]))

{

return true;

}

}

}

return false;

}

template<class TMapFunction>

void Map(TMapFunction map) const

{

MapFrom(0, map);

}

template<class TMapFunction>

void MapAddress(TMapFunction map) const

{

for (int i = 0; i < count; i++)

{

if (TRemovePolicyType::IsItemValid(this->buffer[i]))

{

map(i, &this->buffer[i]);

}

}

}

template<class TMapFunction>

void MapFrom(int start, TMapFunction map) const

{

for (int i = start; i < count; i++)

{

if (TRemovePolicyType::IsItemValid(this->buffer[i]))

{

map(i, this->buffer[i]);

}

}

}

template<class TMapFunction>

void ReverseMap(TMapFunction map)

{

for (int i = count - 1; i >= 0; i--)

{

if (TRemovePolicyType::IsItemValid(this->buffer[i]))

{

map(i, this->buffer[i]);

}

}

}

void Reset()

{

if (this->buffer != nullptr)

{

auto freeFunc = AllocatorInfo::GetFreeFunc();

AllocatorFree(this->alloc, freeFunc, buffer, sizeof(T) \* length); // TODO: Better version of DeleteArray?

this->buffer = nullptr;

count = 0;

length = 0;

}

}

};

}

namespace Js

{

class DefaultListLockPolicy

{

public:

class NoLock

{

public:

template <class SyncObject>

NoLock(SyncObject\*)

{

// No lock, do nothing.

}

};

typedef AutoCriticalSection DefaultLock;

typedef NoLock ReadLock; // To read/map items. Default to no lock.

typedef NoLock WriteLock; // To write an item. Default to no lock.

typedef DefaultLock AddRemoveLock; // To add/remove item

};

//

// A simple wrapper on List to synchronize access.

// Note that this wrapper class only exposes a few methods of List (through "private" inheritance).

// It applies proper lock policy to exposed methods.

//

template <

class T, // Item type in the list

class ListType,

class LockPolicy = DefaultListLockPolicy, // Controls lock policy for read/map/write/add/remove items

class SyncObject = CriticalSection

>

class SynchronizableList sealed: private ListType // Make base class private to lock down exposed methods

{

private:

SyncObject\* syncObj;

public:

template <class Arg1>

SynchronizableList(Arg1 arg1, SyncObject\* syncObj)

: ListType(arg1), syncObj(syncObj)

{

}

template <class Arg1, class Arg2>

SynchronizableList(Arg1 arg1, Arg2 arg2, SyncObject\* syncObj)

: ListType(arg1, arg2), syncObj(syncObj)

{

}

template <class Arg1, class Arg2, class Arg3>

SynchronizableList(Arg1 arg1, Arg2 arg2, Arg3 arg3, SyncObject\* syncObj)

: ListType(arg1, arg2, arg3), syncObj(syncObj)

{

}

int Count() const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::Count();

}

const T& Item(int index) const

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::Item(index);

}

void Item(int index, const T& item)

{

LockPolicy::WriteLock autoLock(syncObj);

\_\_super::Item(index, item);

}

void SetExistingItem(int index, const T& item)

{

LockPolicy::WriteLock autoLock(syncObj);

\_\_super::SetExistingItem(index, item);

}

bool IsItemValid(int index)

{

LockPolicy::ReadLock autoLock(syncObj);

return \_\_super::IsItemValid(index);

}

int SetAtFirstFreeSpot(const T& item)

{

LockPolicy::WriteLock autoLock(syncObj);

return \_\_super::SetAtFirstFreeSpot(item);

}

void ClearAndZero()

{

LockPolicy::WriteLock autoLock(syncObj);

\_\_super::ClearAndZero();

}

void RemoveAt(int index)

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::RemoveAt(index);

}

int Add(const T& item)

{

LockPolicy::AddRemoveLock autoLock(syncObj);

return \_\_super::Add(item);

}

template<class TMapFunction>

void Map(TMapFunction map) const

{

LockPolicy::ReadLock autoLock(syncObj);

\_\_super::Map(map);

}

template<class DebugSite, class TMapFunction>

HRESULT Map(DebugSite site, TMapFunction map) const // external debugging version

{

// No lock needed. Threads are suspended during external debugging.

return \_\_super::Map(site, map);

}

};

template <typename TListType, bool clearOldEntries = false>

class CopyRemovePolicy

{

typedef typename TListType::TElementType TElementType;

typedef typename TListType::TComparerType TComparerType;

public:

CopyRemovePolicy(TListType \* list) {};

void Remove(TListType\* list, const TElementType& item)

{

TElementType\* buffer = list->buffer;

int& count = list->count;

for (int i = 0; i < count; i++)

{

if (TComparerType::Equals(buffer[i], item))

{

for (int j = i + 1; j < count; i++, j++)

{

buffer[i] = buffer[j];

}

count--;

if (clearOldEntries)

{

memset(buffer + count, 0, sizeof(TElementType));

}

break;

}

}

}

int GetFreeItemIndex(TListType\* list)

{

return -1;

}

void RemoveAt(TListType\* list, int index)

{

Assert(index >= 0 && index < list->count);

for (int j = index + 1; j < list->count; index++, j++)

{

list->buffer[index] = list->buffer[j];

}

list->count--;

if (clearOldEntries)

{

memset(list->buffer + list->count, 0, sizeof(TElementType));

}

}

static bool IsItemValid(const TElementType& item)

{

return true;

}

bool IsItemValid(TListType\* list, int index)

{

Assert(index >= 0 && index < list->count);

return true;

}

};

template <typename TListType, bool clearOldEntries = false>

class FreeListedRemovePolicy

{

protected:

typedef typename TListType::TElementType TElementType;

typedef typename TListType::TComparerType TComparerType;

int freeItemIndex;

public:

FreeListedRemovePolicy(TListType \* list):

freeItemIndex(-1)

{

CompileAssert(IsPointer<TElementType>::IsTrue);

}

static bool IsItemValid(const TElementType& item)

{

return (item != nullptr && (::Math::PointerCastToIntegralTruncate<unsigned int>(item) & 1) == 0);

}

bool IsItemValid(TListType\* list, int index)

{

const TElementType& item = list->Item(index);

return IsItemValid(item);

}

void Remove(TListType\* list, const TElementType& item)

{

TElementType\* buffer = list->buffer;

int& count = list->count;

for (int i = 0; i < count; i++)

{

if (TComparerType::Equals(buffer[i], item))

{

RemoveAt(list, i);

break;

}

}

}

int GetFreeItemIndex(TListType\* list)

{

int currentFreeIndex = this->freeItemIndex;

if (currentFreeIndex != -1)

{

unsigned int nextFreeIndex = ::Math::PointerCastToIntegralTruncate<unsigned int>(list->Item(currentFreeIndex));

if (nextFreeIndex != ((unsigned int) -1))

{

// Since this is an unsigned shift, the sign bit is 0, which is what we want

this->freeItemIndex = (int) ((nextFreeIndex) >> 1);

}

else

{

this->freeItemIndex = -1;

}

return currentFreeIndex;

}

return -1;

}

void RemoveAt(TListType\* list, int index)

{

Assert(index >= 0 && index < list->Count());

Assert(IsItemValid(list, index));

unsigned int storedIndex = (unsigned int) this->freeItemIndex;

// Sentinel value, so leave that as is

// Otherwise, this has the range of all +ve integers

if (this->freeItemIndex != -1)

{

// Set a tag bit to indicate this is a free list index, rather than a list value

// Pointers will be aligned anyway

storedIndex = (storedIndex << 1) | 1;

}

list->SetExistingItem(index, (TElementType) (storedIndex));

this->freeItemIndex = index;

}

};

template <typename TListType, bool clearOldEntries = false>

class WeakRefFreeListedRemovePolicy : public FreeListedRemovePolicy<TListType, clearOldEntries>

{

private:

uint lastWeakReferenceCleanupId;

void CleanupWeakReference(TListType \* list)

{

list->Map([list](int i, TElementType weakRef)

{

if (weakRef->Get() == nullptr)

{

list->RemoveAt(i);

}

});

this->lastWeakReferenceCleanupId = list->alloc->GetWeakReferenceCleanupId();

}

public:

WeakRefFreeListedRemovePolicy(TListType \* list) : FreeListedRemovePolicy(list)

{

this->lastWeakReferenceCleanupId = list->alloc->GetWeakReferenceCleanupId();

}

int GetFreeItemIndex(TListType \* list)

{

if (list->alloc->GetWeakReferenceCleanupId() != this->lastWeakReferenceCleanupId)

{

CleanupWeakReference(list);

}

return \_\_super::GetFreeItemIndex(list);

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

// - Maintains an MRU linked list of fixed maximum length and a dictionary that contains all added values

// - RemoveRecentlyUnusedItems should be called to remove values that are not in the MRU list from the dictionary

// - Added values are linked to the beginning of the MRU list (most recently used) as part of an MruListEntry, and when the

// list is full, entries from the end of the list (least recently used) are reused for the new values

// - TryGetValue, if the key exists in the dictionary, adds the value to the MRU list as the most recently used value, and

// removes the least recently used value as necessary

template<

class TKey,

class TValue,

class TAllocator = Recycler,

class TSizePolicy = PowerOf2SizePolicy,

template<class ValueOrKey> class TComparer = DefaultComparer,

template<class K, class V> class TDictionaryEntry = SimpleDictionaryEntry>

class MruDictionary

{

private:

struct MruListEntry : public DoublyLinkedListElement<MruListEntry>

{

TValue value;

TKey key;

int dictionaryDataIndex;

MruListEntry(const TKey &key, const TValue &value) : key(key), value(value), dictionaryDataIndex(0)

{

}

PREVENT\_COPY(MruListEntry);

};

public:

class MruDictionaryData

{

private:

MruListEntry \*entry;

TValue value;

public:

MruDictionaryData() : entry(nullptr)

{

}

MruDictionaryData &operator =(const void \*const nullValue)

{

// Needed to support KeyValueEntry::Clear for dictionaries

Assert(!nullValue);

entry = nullptr;

value = nullptr; // TValue must also support this for the same reason

return \*this;

}

MruListEntry \*Entry() const

{

return entry;

}

const TValue &Value() const

{

Assert(!entry);

return value;

}

void OnAddedToMruList(MruListEntry \*const entry)

{

Assert(!this->entry);

this->entry = entry;

}

void OnRemovedFromMruList()

{

Assert(entry);

value = entry->value;

entry = nullptr;

}

};

private:

const int mruListCapacity;

int mruListCount;

DoublyLinkedList<MruListEntry> entries;

typedef

BaseDictionary<

TKey,

MruDictionaryData,

// MruDictionaryData always has pointer to GC pointer (MruEntry)

typename ForceNonLeafAllocator<TAllocator>::AllocatorType,

TSizePolicy,

TComparer,

TDictionaryEntry>

TDictionary;

TDictionary dictionary;

typedef typename TDictionary::AllocatorType AllocatorType;

public:

MruDictionary(AllocatorType \*const allocator, const int mruListCapacity)

: mruListCapacity(mruListCapacity), mruListCount(0), dictionary(allocator)

{

Assert(allocator);

Assert(mruListCapacity > 0);

}

static MruDictionary \*New(TAllocator \*const allocator, const int mruListCapacity)

{

return AllocatorNew(TAllocator, allocator, MruDictionary, allocator, mruListCapacity);

}

private:

void AddToDictionary(MruListEntry \*const entry)

{

const auto dictionaryDataIndex = dictionary.Add(entry->key, MruDictionaryData());

dictionary.GetReferenceAt(dictionaryDataIndex)->OnAddedToMruList(entry);

entry->dictionaryDataIndex = dictionaryDataIndex;

}

void ReuseLeastRecentlyUsedEntry(const TKey &key, const TValue &value, const int dictionaryDataIndex)

{

Assert(mruListCount == mruListCapacity);

// Reuse the least recently used entry for this key/value pair and make it the most recently used

const auto entry = entries.Tail();

dictionary.GetReferenceAt(dictionaryDataIndex)->OnAddedToMruList(entry);

dictionary.GetReferenceAt(entry->dictionaryDataIndex)->OnRemovedFromMruList();

entries.MoveToBeginning(entry);

entry->key = key;

entry->value = value;

entry->dictionaryDataIndex = dictionaryDataIndex;

}

public:

bool TryGetValue(const TKey &key, TValue \*const value)

{

MruDictionaryData \*dictionaryData;

int dictionaryDataIndex;

if(!dictionary.TryGetReference(key, &dictionaryData, &dictionaryDataIndex))

return false;

const auto entry = dictionaryData->Entry();

if(entry)

{

// Make this the most recently used entry

entries.MoveToBeginning(entry);

\*value = entry->value;

return true;

}

\*value = dictionaryData->Value();

// The key passed into this function may be temporary, and should not be placed in the MRU list or dictionary. Get

// the proper key to be used from the dictionary. That key should have the necessary lifetime.

ReuseLeastRecentlyUsedEntry(dictionary.GetKeyAt(dictionaryDataIndex), dictionaryData->Value(), dictionaryDataIndex);

return true;

}

void Add(const TKey &key, const TValue &value)

{

Assert(!dictionary.ContainsKey(key));

Assert(mruListCount <= mruListCapacity);

if(mruListCount == mruListCapacity)

{

ReuseLeastRecentlyUsedEntry(key, value, dictionary.Add(key, MruDictionaryData()));

return;

}

const auto entry = AllocatorNew(TAllocator, dictionary.GetAllocator(), MruListEntry, key, value);

AddToDictionary(entry);

entries.LinkToBeginning(entry);

++mruListCount;

}

void RemoveRecentlyUnusedItems()

{

if(dictionary.Count() == mruListCount)

return;

if(dictionary.Count() / 2 <= mruListCount)

{

dictionary.MapAndRemoveIf(

[](const TDictionary::EntryType &dictionaryEntry) -> bool

{

return !dictionaryEntry.Value().Entry();

});

return;

}

dictionary.Clear();

for(auto entry = entries.Head(); entry; entry = entry->Next())

AddToDictionary(entry);

}

PREVENT\_COPY(MruDictionary);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// Option represents a pointer with explicit null semantics. Use sites are

// required to deal with the null possibility.

//----------------------------------------------------------------------------

#pragma once

namespace regex

{

template<class T>

class Option

{

const T \* value;

public:

// Info: Construct an empty option of type T

Option()

: value(nullptr)

{ }

// Info: Create an empty or non-empty option of type T

// Parameters: value - the pointer to hold. May be null.

Option(const T \* value)

: value(value)

{ }

// Info: Get the held value if there is one. Assert otherwise.

const T \* GetValue() const

{

Assert(HasValue());

return value;

}

// Info: Returns true if there is value.

bool HasValue() const

{

return value!=nullptr;

}

// Info: Get the held value if there is one, otherwise call the given function

// to produce a value

// Parameters: f - function which produces a value of type T

template<class F>

const T \* GetValueOrDefault(F f) const

{

if(value==nullptr)

{

return f();

}

return value;

}

// Info: Get the held value if there is one, otherwise return the given default value

// Parameters: defaultValue - function which produces a value of type T

const T\* GetValueOrDefaultValue(const T \* defaultValue) const

{

if(value==nullptr)

{

return defaultValue;

}

return value;

}

// Info: Get the held value if there is one, otherwise return nullptr

const T\* GetValueOrNull() const

{

return value;

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

template <typename T>

class PageStack

{

private:

struct Chunk : public PagePoolPage

{

Chunk \* nextChunk;

T entries[];

};

static const size\_t EntriesPerChunk = (AutoSystemInfo::PageSize - sizeof(Chunk)) / sizeof(T);

public:

PageStack(PagePool \* pagePool);

~PageStack();

void Init(uint reservedPageCount = 0);

void Clear();

bool Pop(T \* item);

bool Push(T item);

uint Split(uint targetCount, \_\_in\_ecount(targetCount) PageStack<T> \*\* targetStacks);

void Abort();

void Release();

bool IsEmpty() const;

#if DBG

bool HasChunk() const

{

return this->currentChunk != nullptr;

}

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

void SetMaxPageCount(size\_t maxPageCount) { this->maxPageCount = max<size\_t>(maxPageCount, 1); }

#endif

static const uint MaxSplitTargets = 3; // Not counting original stack, so this supports 4-way parallel

private:

Chunk \* CreateChunk();

void FreeChunk(Chunk \* chunk);

private:

T \* nextEntry;

T \* chunkStart;

T \* chunkEnd;

Chunk \* currentChunk;

PagePool \* pagePool;

bool usesReservedPages;

#if DBG

size\_t count;

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

size\_t pageCount;

size\_t maxPageCount;

#endif

};

template <typename T>

\_\_inline

bool PageStack<T>::Pop(T \* item)

{

Assert(currentChunk != nullptr);

if (nextEntry == chunkStart)

{

// We're at the beginning of the chunk. Move to the previous chunk, if any

if (currentChunk->nextChunk == nullptr)

{

// All done

Assert(count == 0);

return false;

}

Chunk \* temp = currentChunk;

currentChunk = currentChunk->nextChunk;

FreeChunk(temp);

chunkStart = currentChunk->entries;

chunkEnd = &currentChunk->entries[EntriesPerChunk];

nextEntry = chunkEnd;

}

Assert(nextEntry > chunkStart && nextEntry <= chunkEnd);

nextEntry--;

\*item = \*nextEntry;

#if DBG

count--;

Assert(count == (nextEntry - chunkStart) + (pageCount - 1) \* EntriesPerChunk);

#endif

return true;

}

template <typename T>

\_\_inline

bool PageStack<T>::Push(T item)

{

if (nextEntry == chunkEnd)

{

Chunk \* newChunk = CreateChunk();

if (newChunk == nullptr)

{

return false;

}

newChunk->nextChunk = currentChunk;

currentChunk = newChunk;

chunkStart = currentChunk->entries;

chunkEnd = &currentChunk->entries[EntriesPerChunk];

nextEntry = chunkStart;

}

Assert(nextEntry >= chunkStart && nextEntry < chunkEnd);

\*nextEntry = item;

nextEntry++;

#if DBG

count++;

Assert(count == (nextEntry - chunkStart) + (pageCount - 1) \* EntriesPerChunk);

#endif

return true;

}

template <typename T>

PageStack<T>::PageStack(PagePool \* pagePool) :

pagePool(pagePool),

currentChunk(nullptr),

nextEntry(nullptr),

chunkStart(nullptr),

chunkEnd(nullptr),

usesReservedPages(false)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

pageCount = 0;

maxPageCount = (size\_t)-1; // Default to no limit

#endif

#if DBG

count = 0;

#endif

}

template <typename T>

PageStack<T>::~PageStack()

{

Assert(currentChunk == nullptr);

Assert(nextEntry == nullptr);

Assert(count == 0);

Assert(pageCount == 0);

}

template <typename T>

void PageStack<T>::Init(uint reservedPageCount)

{

if (reservedPageCount > 0)

{

this->usesReservedPages = true;

this->pagePool->ReservePages(reservedPageCount);

}

// Preallocate one chunk.

Assert(currentChunk == nullptr);

currentChunk = CreateChunk();

if (currentChunk == nullptr)

{

Js::Throw::OutOfMemory();

}

currentChunk->nextChunk = nullptr;

chunkStart = currentChunk->entries;

chunkEnd = &currentChunk->entries[EntriesPerChunk];

nextEntry = chunkStart;

}

template <typename T>

void PageStack<T>::Clear()

{

currentChunk = nullptr;

nextEntry = nullptr;

#if DBG

count = 0;

pageCount = 0;

#endif

}

template <typename T>

typename PageStack<T>::Chunk \* PageStack<T>::CreateChunk()

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (pageCount >= maxPageCount)

{

return nullptr;

}

#endif

Chunk \* newChunk = (Chunk \*)this->pagePool->GetPage(usesReservedPages);

if (newChunk == nullptr)

{

return nullptr;

}

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

pageCount++;

#endif

return newChunk;

}

template <typename T>

void PageStack<T>::FreeChunk(Chunk \* chunk)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

pageCount--;

#endif

this->pagePool->FreePage(chunk);

}

template <typename T>

uint PageStack<T>::Split(uint targetCount, \_\_in\_ecount(targetCount) PageStack<T> \*\* targetStacks)

{

// Split the current stack up to [targetCount + 1] ways.

// [targetStacks] contains the target stacks and must have [targetCount] elements.

Assert(targetCount > 0 && targetCount <= MaxSplitTargets);

Assert(targetStacks);

\_\_analysis\_assume(targetCount <= MaxSplitTargets);

Chunk \* mainCurrent;

Chunk \* targetCurrents[MaxSplitTargets];

// Do the initial split of first pages for each target stack.

// During this, if we run out of pages, we will return a value < maxSplit to

// indicate that the split was less than the maximum possible.

Chunk \* chunk = this->currentChunk;

Assert(chunk != nullptr);

// The first chunk is assigned to the main stack, and since it's already there,

// we just advance to the next chunk and start assigning to each target stack.

mainCurrent = chunk;

chunk = chunk->nextChunk;

uint targetIndex = 0;

while (targetIndex < targetCount)

{

if (chunk == nullptr)

{

// No more pages. Adjust targetCount down to what we were actually able to do.

// We'll return this number below so the caller knows.

targetCount = targetIndex;

break;

}

// Target stack should be empty.

// If it has a free page currently, release it.

Assert(targetStacks[targetIndex]->IsEmpty());

targetStacks[targetIndex]->Release();

targetStacks[targetIndex]->currentChunk = chunk;

targetStacks[targetIndex]->chunkStart = chunk->entries;

targetStacks[targetIndex]->chunkEnd = &chunk->entries[EntriesPerChunk];

targetStacks[targetIndex]->nextEntry = targetStacks[targetIndex]->chunkEnd;

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

this->pageCount--;

targetStacks[targetIndex]->pageCount = 1;

#endif

#if DBG

this->count -= EntriesPerChunk;

targetStacks[targetIndex]->count = EntriesPerChunk;

#endif

targetCurrents[targetIndex] = chunk;

chunk = chunk->nextChunk;

targetIndex++;

}

// Loop through the remaining chunks (if any),

// assigning each chunk to the main chunk and the target chunks in turn,

// and linking each chunk to the end of the respective list.

while (true)

{

if (chunk == nullptr)

{

break;

}

mainCurrent->nextChunk = chunk;

mainCurrent = chunk;

chunk = chunk->nextChunk;

targetIndex = 0;

while (targetIndex < targetCount)

{

if (chunk == nullptr)

{

break;

}

targetCurrents[targetIndex]->nextChunk = chunk;

targetCurrents[targetIndex] = chunk;

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

this->pageCount--;

targetStacks[targetIndex]->pageCount++;

#endif

#if DBG

this->count -= EntriesPerChunk;

targetStacks[targetIndex]->count += EntriesPerChunk;

#endif

chunk = chunk->nextChunk;

targetIndex++;

}

}

// Terminate all the split chunk lists with null

mainCurrent->nextChunk = nullptr;

targetIndex = 0;

while (targetIndex < targetCount)

{

targetCurrents[targetIndex]->nextChunk = nullptr;

targetIndex++;

}

// Return the actual split count we were able to do, which may have been lowered above.

return targetCount;

}

template <typename T>

void PageStack<T>::Abort()

{

// Abandon the current entries in the stack and reset to initialized state.

if (currentChunk == nullptr)

{

Assert(count == 0);

return;

}

// Free all the chunks except the first one

while (currentChunk->nextChunk != nullptr)

{

Chunk \* temp = currentChunk;

currentChunk = currentChunk->nextChunk;

FreeChunk(temp);

}

chunkStart = currentChunk->entries;

chunkEnd = &currentChunk->entries[EntriesPerChunk];

nextEntry = chunkStart;

#if DBG

count = 0;

#endif

}

template <typename T>

void PageStack<T>::Release()

{

Assert(IsEmpty());

// We may have a preallocated chunk still held; if so release it.

if (currentChunk != nullptr)

{

Assert(currentChunk->nextChunk == nullptr);

FreeChunk(currentChunk);

currentChunk = nullptr;

}

nextEntry = nullptr;

chunkStart = nullptr;

chunkEnd = nullptr;

}

template <typename T>

bool PageStack<T>::IsEmpty() const

{

if (currentChunk == nullptr)

{

Assert(count == 0);

Assert(nextEntry == nullptr);

return true;

}

if (nextEntry == chunkStart && currentChunk->nextChunk == nullptr)

{

Assert(count == 0);

return true;

}

Assert(count != 0);

return false;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template<class TFirst, class TSecond, template<class TValue> class Comparer = DefaultComparer>

class Pair

{

private:

TFirst first;

TSecond second;

#if DBG

bool initialized;

#endif

public:

Pair()

#if DBG

: initialized(false)

#endif

{

Assert(!IsValid());

}

Pair(const TFirst &first, const TSecond &second)

: first(first),

second(second)

#if DBG

,

initialized(true)

#endif

{

Assert(IsValid());

}

#if DBG

private:

bool IsValid() const

{

return initialized;

}

#endif

public:

const TFirst &First() const

{

Assert(IsValid());

return first;

}

const TSecond &Second() const

{

Assert(IsValid());

return second;

}

public:

bool operator ==(const Pair &other) const

{

return Comparer<TFirst>::Equals(first, other.first) && Comparer<TSecond>::Equals(second, other.second);

}

operator hash\_t() const

{

return Comparer<TFirst>::GetHashCode(first) + Comparer<TSecond>::GetHashCode(second);

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template <class T, class Allocator>

class Queue

{

private:

DList<T, Allocator> list;

public:

Queue(Allocator\* alloc) : list(alloc)

{

}

bool Empty() const

{

return list.Empty();

}

void Enqueue(const T& item)

{

list.Append(item);

}

T Dequeue()

{

T item = list.Head();

list.RemoveHead();

return item;

}

void Clear()

{

list.Clear();

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace regex

{

template <class T, bool partitionFromMiddle> class QuickSort

{

public:

static void Sort(T\* low, T\* high, Comparer<T> \* comp)

{

if ((low == NULL)||(high == NULL))

{

return;

}

if (high > low)

{

T\* pivot = Partition(low, high, comp);

Sort(low, pivot-1, comp);

Sort(pivot+1, high, comp);

}

}

private:

static T\* Partition(T\* l, T\* r, Comparer<T> \* comp)

{

if (partitionFromMiddle)

{

// Swap middle value to end to use as partition value

T\* mid = l + ((r - l) / 2);

swap(\*mid, \*r);

}

T\* i = l-1;

T\* j = r;

T v = \*r;

for (;;)

{

while (comp->Compare(\*(++i), v) < 0) ;

while (comp->Compare(v, \*(--j)) < 0) if (j == l) break;

if (i >= j) break;

swap(\*i, \*j);

}

swap(\*i, \*r);

return i;

}

inline static void swap(T& x, T& y)

{

T temp = x;

x = y;

y = temp;

}

};

}

namespace JsUtil

{

template <class T, class TComparer> class QuickSort

{

public:

static void Sort(T\* low, T\* high)

{

if ((low == NULL)||(high == NULL))

{

return;

}

if (high > low)

{

T\* pivot = Partition(low, high);

Sort(low, pivot-1);

Sort(pivot+1, high);

}

}

private:

static T\* Partition(T\* l, T\* r)

{

T\* i = l-1;

T\* j = r;

T v = \*r;

for (;;)

{

while (TComparer::Compare(\*(++i), v) < 0) ;

while (TComparer::Compare(v, \*(--j)) < 0) if (j == l) break;

if (i >= j) break;

swap(\*i, \*j);

}

swap(\*i, \*r);

return i;

}

inline static void swap(T& x, T& y)

{

T temp = x;

x = y;

y = temp;

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace UnifiedRegex

{

enum RegexFlags : uint8;

class RegexKey

{

private:

const wchar\_t \*source;

int length;

RegexFlags flags;

public:

RegexKey() : source(nullptr), length(0), flags(static\_cast<RegexFlags>(0))

{

}

RegexKey(const wchar\_t \*const source, const int length, const RegexFlags flags)

: source(source), length(length), flags(flags)

{

Assert(source);

Assert(length >= 0);

}

RegexKey &operator =(const void \*const nullValue)

{

// Needed to support KeyValueEntry::Clear for dictionaries

Assert(!nullValue);

source = nullptr;

length = 0;

flags = static\_cast<RegexFlags>(0);

return \*this;

}

const wchar\_t \*Source() const

{

return source;

}

int Length() const

{

return length;

}

RegexFlags Flags() const

{

return flags;

}

};

struct RegexKeyComparer

{

\_\_inline static bool Equals(const RegexKey &key1, const RegexKey &key2)

{

return

Js::InternalStringComparer::Equals(

Js::InternalString(key1.Source(), key1.Length()),

Js::InternalString(key2.Source(), key2.Length())) &&

key1.Flags() == key2.Flags();

}

\_\_inline static hash\_t GetHashCode(const RegexKey &key)

{

return Js::InternalStringComparer::GetHashCode(Js::InternalString(key.Source(), key.Length()));

}

};

}

template<>

struct DefaultComparer<UnifiedRegex::RegexKey> : public UnifiedRegex::RegexKeyComparer

{

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template<typename TKey, typename TData>

struct SimpleHashEntry {

TKey key;

TData value;

SimpleHashEntry \*next;

};

// Size should be a power of 2 for optimal performance

template<

typename TKey,

typename TData,

typename TAllocator = ArenaAllocator,

template <typename DataOrKey> class Comparer = DefaultComparer,

bool resize = false,

typename SizePolicy = PowerOf2Policy>

class SimpleHashTable

{

typedef SimpleHashEntry<TKey, TData> EntryType;

// REVIEW: Consider 5 or 7 as multiplication of these might be faster.

static const int MaxAverageChainLength = 6;

TAllocator \*allocator;

EntryType \*\*table;

EntryType \*free;

uint count;

uint size;

uint freecount;

bool disableResize;

#if PROFILE\_DICTIONARY

DictionaryStats \*stats;

#endif

public:

SimpleHashTable(TAllocator \*allocator) :

allocator(allocator),

count(0),

freecount(0)

{

this->size = SizePolicy::GetSize(64);

Initialize();

}

SimpleHashTable(uint size, TAllocator\* allocator) :

allocator(allocator),

count(0),

freecount(0)

{

this->size = SizePolicy::GetSize(size);

Initialize();

}

void Initialize()

{

disableResize = false;

free = nullptr;

table = AllocatorNewArrayZ(TAllocator, allocator, EntryType\*, size);

#if PROFILE\_DICTIONARY

stats = DictionaryStats::Create(typeid(this).name(), size);

#endif

}

~SimpleHashTable()

{

for (uint i = 0; i < size; i++)

{

EntryType \* entry = table[i];

while (entry != nullptr)

{

EntryType \* next = entry->next;

AllocatorDelete(TAllocator, allocator, entry);

entry = next;

}

}

while(free)

{

EntryType\* current = free;

free = current->next;

AllocatorDelete(TAllocator, allocator, current);

}

AllocatorDeleteArray(TAllocator, allocator, size, table);

}

void DisableResize()

{

Assert(!resize || !disableResize);

disableResize = true;

}

void EnableResize()

{

Assert(!resize || disableResize);

disableResize = false;

}

void Set(TKey key, TData data)

{

EntryType\* entry = FindOrAddEntry(key);

entry->value = data;

}

bool Add(TKey key, TData data)

{

uint targetBucket = HashKeyToBucket(key);

if(FindEntry(key, targetBucket) != nullptr)

{

return false;

}

AddInternal(key, data, targetBucket);

return true;

}

void ReplaceValue(TKey key,TData data)

{

EntryType \*current = FindEntry(key);

if (current != nullptr)

{

current->value = data;

}

}

void Remove(TKey key)

{

Remove(key, nullptr);

}

void Remove(TKey key, TData\* pOut)

{

uint val = HashKeyToBucket(key);

EntryType \*\*prev=&table[val];

for (EntryType \* current = \*prev ; current != nullptr; current = current->next)

{

if (Comparer<TKey>::Equals(key, current->key))

{

\*prev = current->next;

if (pOut != nullptr)

{

(\*pOut) = current->value;

}

count--;

FreeEntry(current);

#if PROFILE\_DICTIONARY

if (stats)

stats->Remove(table[val] == nullptr);

#endif

break;

}

prev = &current->next;

}

}

BOOL HasEntry(TKey key)

{

return (FindEntry(key) != nullptr);

}

uint Count() const

{

return(count);

}

// If density is a compile-time constant, then we can optimize (avoids division)

// Sometimes the compiler can also make this optimization, but this way it is guaranteed.

template< uint density > bool IsDenserThan() const

{

return count > (size \* density);

}

TData Lookup(TKey key)

{

EntryType \*current = FindEntry(key);

if (current != nullptr)

{

return current->value;

}

return TData();

}

TData LookupIndex(int index)

{

EntryType \*current;

int j=0;

for (uint i=0; i < size; i++)

{

for (current = table[i] ; current != nullptr; current = current->next)

{

if (j==index)

{

return current->value;

}

j++;

}

}

return nullptr;

}

bool TryGetValue(TKey key, TData \*dataReference)

{

EntryType \*current = FindEntry(key);

if (current != nullptr)

{

\*dataReference = current->value;

return true;

}

return false;

}

TData& GetReference(TKey key)

{

EntryType \* current = FindOrAddEntry(key);

return current->value;

}

TData \* TryGetReference(TKey key)

{

EntryType \* current = FindEntry(key);

if (current != nullptr)

{

return &current->value;

}

return nullptr;

}

template <class Fn>

void Map(Fn fn)

{

EntryType \*current;

for (uint i=0;i<size;i++) {

for (current = table[i] ; current != nullptr; current = current->next) {

fn(current->key,current->value);

}

}

}

template <class Fn>

void MapAndRemoveIf(Fn fn)

{

for (uint i=0; i<size; i++)

{

EntryType \*\* prev = &table[i];

while (EntryType \* current = \*prev)

{

if (fn(current->key,current->value))

{

\*prev = current->next;

FreeEntry(current);

}

else

{

prev = &current->next;

}

}

}

}

private:

uint HashKeyToBucket(TKey hashKey)

{

return HashKeyToBucket(hashKey, size);

}

uint HashKeyToBucket(TKey hashKey, int size)

{

uint hashCode = Comparer<TKey>::GetHashCode(hashKey);

return SizePolicy::GetBucket(hashCode, size);

}

EntryType \* FindEntry(TKey key)

{

uint targetBucket = HashKeyToBucket(key);

return FindEntry(key, targetBucket);

}

EntryType \* FindEntry(TKey key, uint targetBucket)

{

for (EntryType \* current = table[targetBucket] ; current != nullptr; current = current->next)

{

if (Comparer<TKey>::Equals(key, current->key))

{

return current;

}

}

return nullptr;

}

EntryType \* FindOrAddEntry(TKey key)

{

uint targetBucket = HashKeyToBucket(key);

EntryType \* entry = FindEntry(key, targetBucket);

if (entry == nullptr)

{

entry = AddInternal(key, TData(), targetBucket);

}

return entry;

}

void FreeEntry(EntryType\* current)

{

if ( freecount < 10 )

{

current->key = nullptr;

current->value = NULL;

current->next = free;

free = current;

freecount++;

}

else

{

AllocatorDelete(TAllocator, allocator, current);

}

}

EntryType\* GetFreeEntry()

{

EntryType\* retFree = free;

if (nullptr == retFree )

{

retFree = AllocatorNewStruct(TAllocator, allocator, EntryType);

}

else

{

free = retFree->next;

freecount--;

}

return retFree;

}

EntryType\* AddInternal(TKey key, TData data, uint targetBucket)

{

if(resize && !disableResize && IsDenserThan<MaxAverageChainLength>())

{

Resize(SizePolicy::GetSize(size\*2));

// After resize - we will need to recalculate the bucket

targetBucket = HashKeyToBucket(key);

}

EntryType\* entry = GetFreeEntry();

entry->key = key;

entry->value = data;

entry->next = table[targetBucket];

table[targetBucket] = entry;

count++;

#if PROFILE\_DICTIONARY

uint depth = 0;

for (EntryType \* current = table[targetBucket] ; current != nullptr; current = current->next)

{

++depth;

}

if (stats)

stats->Insert(depth);

#endif

return entry;

}

void Resize(int newSize)

{

Assert(!this->disableResize);

EntryType\*\* newTable = AllocatorNewArrayZ(TAllocator, allocator, EntryType\*, newSize);

for (uint i=0; i < size; i++)

{

EntryType\* current = table[i];

while (current != nullptr)

{

int targetBucket = HashKeyToBucket(current->key, newSize);

EntryType\* next = current->next; // Cache the next pointer

current->next = newTable[targetBucket];

newTable[targetBucket] = current;

current = next;

}

}

AllocatorDeleteArray(TAllocator, allocator, this->size, this->table);

this->size = newSize;

this->table = newTable;

#if PROFILE\_DICTIONARY

if (stats)

{

uint emptyBuckets = 0 ;

for (uint i=0; i < size; i++)

{

if(table[i] == nullptr)

{

emptyBuckets++;

}

}

stats->Resize(newSize, emptyBuckets);

}

#endif

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

static const uint primes[] = {

3, 7, 11, 17, 23, 29, 37, 47, 59, 71, 89, 107, 131, 163, 197, 239, 293, 353, 431, 521, 631, 761, 919,

1103, 1327, 1597, 1931, 2333, 2801, 3371, 4049, 4861, 5839, 7013, 8419, 10103, 12143, 14591,

17519, 21023, 25229, 30293, 36353, 43627, 52361, 62851, 75431, 90523, 108631, 130363, 156437,

187751, 225307, 270371, 324449, 389357, 467237, 560689, 672827, 807403, 968897, 1162687, 1395263,

1674319, 2009191, 2411033, 2893249, 3471899, 4166287, 4999559, 5999471, 7199369

};

bool

PrimePolicy::IsPrime(uint candidate)

{

if ((candidate & 1) != 0)

{

int limit = (uint)sqrt((FLOAT)candidate);

for (int divisor = 3; divisor <= limit; divisor += 2)

{

if ((candidate % divisor) == 0)

return false;

}

return true;

}

return (candidate == 2);

}

uint

PrimePolicy::GetPrime(uint min)

{

if (min <= 0)

return 17;

for (int i = 0; i < sizeof(primes)/sizeof(uint); i++)

{

uint prime = primes[i];

if (prime >= min) return prime;

}

//outside of our predefined table.

//compute the hard way.

for (uint i = (min | 1); i < 0x7FFFFFFF; i += 2)

{

if (IsPrime(i))

{

return i;

}

}

return min;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

struct PrimePolicy

{

\_\_inline static uint GetBucket(hash\_t hashCode, int size)

{

uint targetBucket = hashCode % size;

return targetBucket;

}

\_\_inline static uint GetSize(uint capacity)

{

return GetPrime(capacity);

}

private:

static bool IsPrime(uint candidate);

static uint GetPrime(uint min);

};

struct PowerOf2Policy

{

\_\_inline static uint GetBucket(hash\_t hashCode, int size)

{

AssertMsg(Math::IsPow2(size), "Size is not a power of 2.");

uint targetBucket = hashCode & (size-1);

return targetBucket;

}

/// Returns a size that is power of 2 and

/// greater than specified capacity.

\_\_inline static uint GetSize(size\_t minCapacity\_t)

{

AssertMsg(minCapacity\_t <= MAXINT32, "the next higher power of 2 must fit in uint32");

uint minCapacity = static\_cast<uint>(minCapacity\_t);

if(minCapacity <= 0)

{

return 4;

}

if (Math::IsPow2(minCapacity))

{

return minCapacity;

}

else

{

return 1 << (Math::Log2(minCapacity) + 1);

}

}

};

#ifndef JD\_PRIVATE

template <class SizePolicy, uint averageChainLength = 2, uint growthRateNumerator = 2, uint growthRateDenominator = 1, uint minBucket = 4>

struct DictionarySizePolicy

{

CompileAssert(growthRateNumerator > growthRateDenominator);

CompileAssert(growthRateDenominator != 0);

\_\_inline static uint GetBucket(hash\_t hashCode, uint bucketCount)

{

return SizePolicy::GetBucket(hashCode, bucketCount);

}

\_\_inline static uint GetNextSize(uint minCapacity)

{

uint nextSize = minCapacity \* growthRateNumerator / growthRateDenominator;

return (growthRateDenominator != 1 && nextSize <= minCapacity)? minCapacity + 1 : nextSize;

}

\_\_inline static uint GetBucketSize(uint size)

{

if (minBucket \* averageChainLength >= size)

{

return SizePolicy::GetSize(minBucket);

}

return SizePolicy::GetSize((size + (averageChainLength - 1)) / averageChainLength);

}

};

typedef DictionarySizePolicy<PrimePolicy> PrimeSizePolicy;

typedef DictionarySizePolicy<PowerOf2Policy> PowerOf2SizePolicy;

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

//----------------------------------------------------------------------------

//

// File: SList.h

//

// Template for Singly Linked List

//

//----------------------------------------------------------------------------

class FakeCount

{

protected:

void IncrementCount() {}

void DecrementCount() {}

void SetCount(uint count) {}

void AddCount(FakeCount& c) {}

};

class RealCount

{

protected:

RealCount() : count(0) {}

void IncrementCount() { count++; }

void DecrementCount() { count--; }

void SetCount(uint count) { this->count = count; }

void AddCount(RealCount const& c) { this->count += c.Count(); }

public:

uint Count() const { return count; }

private:

uint count;

};

#if DBG

typedef RealCount DefaultCount;

#else

typedef FakeCount DefaultCount;

#endif

template <typename TData, typename TCount = DefaultCount> class SListBase;

template <typename TData> class SListNode;

template <typename TData>

class SListNodeBase

{

public:

SListNodeBase<TData> \* Next() const { return next.base; }

SListNodeBase<TData> \*& Next() { return next.base; }

protected:

// The next node can be a real node with data, or it point back to the start of the list

// Use a union to show it in the debugger (instead of casting everywhere)

union

{

SListNodeBase<TData> \* base;

SListNode<TData> \* node;

SListBase<TData> \* list;

} next;

};

template <typename TData>

class SListNode : public SListNodeBase<TData>

{

friend class SListBase<TData, FakeCount>;

friend class SListBase<TData, RealCount>;

private:

SListNode() : data() {}

// Constructing with parameter

template <typename TParam>

SListNode(TParam param) : data(param) {}

// Constructing with parameter

template <typename TParam1, typename TParam2>

SListNode(TParam1 param1, TParam2 param2) : data(param1, param2) {}

// Constructing using copy constructor

SListNode(TData const& data) : data(data) {};

TData data;

};

template<typename TData, typename TCount>

class SListBase : protected SListNodeBase<TData>, public TCount

{

private:

typedef SListNodeBase<TData> NodeBase;

typedef SListNode<TData> Node;

bool IsHead(NodeBase const \* node) const

{

return (node == this);

}

public:

class Iterator

{

public:

Iterator() : list(nullptr), current(nullptr) {}

Iterator(SListBase const \* list) : list(list), current(list) {};

bool IsValid() const

{

return (current != nullptr && !list->IsHead(current));

}

void Reset()

{

current = list;

}

// forceinline only needed for SListBase<FlowEdge \*, RealCount>::Iterator::Next()

\_\_forceinline

bool Next()

{

Assert(current != nullptr);

if (list->IsHead(current->Next()))

{

current = nullptr;

return false;

}

current = current->Next();

return true;

}

TData const& Data() const

{

Assert(this->IsValid());

return ((Node \*)current)->data;

}

TData& Data()

{

Assert(this->IsValid());

return ((Node \*)current)->data;

}

protected:

SListBase const \* list;

NodeBase const \* current;

};

class EditingIterator : public Iterator

{

public:

EditingIterator() : Iterator(), last(nullptr) {};

EditingIterator(SListBase \* list) : Iterator(list), last(nullptr) {};

bool Next()

{

if (last != nullptr && last->Next() != current)

{

current = last;

}

else

{

last = current;

}

return Iterator::Next();

}

void UnlinkCurrent()

{

UnlinkCurrentNode();

}

template <typename TAllocator>

void RemoveCurrent(TAllocator \* allocator)

{

const NodeBase \*dead = current;

UnlinkCurrent();

auto freeFunc = TypeAllocatorFunc<TAllocator, TData>::GetFreeFunc();

AllocatorFree(allocator, freeFunc, (Node \*) dead, sizeof(Node));

}

template <typename TAllocator>

TData \* InsertNodeBefore(TAllocator \* allocator)

{

Assert(last != nullptr);

Node \* newNode = AllocatorNew(TAllocator, allocator, Node);

if (newNode)

{

newNode->Next() = last->Next();

const\_cast<NodeBase \*>(last)->Next() = newNode;

const\_cast<SListBase \*>(list)->IncrementCount();

last = newNode;

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator>

TData \* InsertNodeBeforeNoThrow(TAllocator \* allocator)

{

Assert(last != nullptr);

Node \* newNode = AllocatorNewNoThrow(TAllocator, allocator, Node);

if (newNode)

{

newNode->Next() = last->Next();

const\_cast<NodeBase \*>(last)->Next() = newNode;

const\_cast<SListBase \*>(list)->IncrementCount();

last = newNode;

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator>

bool InsertBefore(TAllocator \* allocator, TData const& data)

{

Assert(last != nullptr);

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, data);

if (newNode)

{

newNode->Next() = last->Next();

const\_cast<NodeBase \*>(last)->Next() = newNode;

const\_cast<SListBase \*>(list)->IncrementCount();

last = newNode;

return true;

}

return false;

}

void MoveCurrentTo(SListBase \* toList)

{

NodeBase \* node = UnlinkCurrentNode();

node->Next() = toList->Next();

toList->Next() = node;

toList->IncrementCount();

}

private:

NodeBase const \* last;

NodeBase \* UnlinkCurrentNode()

{

NodeBase \* unlinkedNode = const\_cast<NodeBase \*>(current);

Assert(current != nullptr);

Assert(!list->IsHead(current));

Assert(last != nullptr);

const\_cast<NodeBase \*>(last)->Next() = current->Next();

current = last;

last = nullptr;

const\_cast<SListBase \*>(list)->DecrementCount();

return unlinkedNode;

}

};

explicit SListBase()

{

Reset();

}

~SListBase()

{

AssertMsg(this->Empty(), "SListBase need to be cleared explicitly with an allocator");

}

void Reset()

{

this->Next() = this;

this->SetCount(0);

}

template <typename TAllocator>

\_\_forceinline

void Clear(TAllocator \* allocator)

{

NodeBase \* current = this->Next();

while (!this->IsHead(current))

{

NodeBase \* next = current->Next();

auto freeFunc = TypeAllocatorFunc<TAllocator, TData>::GetFreeFunc();

AllocatorFree(allocator, freeFunc, (Node \*)current, sizeof(Node));

current = next;

}

this->Reset();

}

bool Empty() const { return this->IsHead(this->Next()); }

bool HasOne() const { return !Empty() && this->IsHead(this->Next()->Next()); }

bool HasTwo() const { return !Empty() && this->IsHead(this->Next()->Next()->Next()); }

TData const& Head() const { Assert(!Empty()); return ((Node \*)this->Next())->data; }

TData& Head()

{

Assert(!Empty());

Node \* node = this->next.node;

return node->data;

}

template <typename TAllocator>

bool Prepend(TAllocator \* allocator, TData const& data)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, data);

if (newNode)

{

newNode->Next() = this->Next();

this->Next() = newNode;

this->IncrementCount();

return true;

}

return false;

}

template <typename TAllocator>

bool PrependNoThrow(TAllocator \* allocator, TData const& data)

{

Node \* newNode = AllocatorNewNoThrow(TAllocator, allocator, Node, data);

if (newNode)

{

newNode->Next() = this->Next();

this->Next() = newNode;

this->IncrementCount();

return true;

}

return false;

}

template <typename TAllocator>

TData \* PrependNode(TAllocator \* allocator)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node);

if (newNode)

{

newNode->Next() = this->Next();

this->Next() = newNode;

this->IncrementCount();

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator, typename TParam>

TData \* PrependNode(TAllocator \* allocator, TParam param)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, param);

if (newNode)

{

newNode->Next() = this->Next();

this->Next() = newNode;

this->IncrementCount();

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator, typename TParam1, typename TParam2>

TData \* PrependNode(TAllocator \* allocator, TParam1 param1, TParam2 param2)

{

Node \* newNode = AllocatorNew(TAllocator, allocator, Node, param1, param2);

if (newNode)

{

newNode->Next() = this->Next();

this->Next() = newNode;

this->IncrementCount();

return &newNode->data;

}

return nullptr;

}

template <typename TAllocator>

void RemoveHead(TAllocator \* allocator)

{

Assert(!this->Empty());

NodeBase \* node = this->Next();

this->Next() = node->Next();

auto freeFunc = TypeAllocatorFunc<TAllocator, TData>::GetFreeFunc();

AllocatorFree(allocator, freeFunc, (Node \*) node, sizeof(Node));

this->DecrementCount();

}

template <typename TAllocator>

bool Remove(TAllocator \* allocator, TData const& data)

{

EditingIterator iter(this);

while (iter.Next())

{

if (iter.Data() == data)

{

iter.RemoveCurrent(allocator);

return true;

}

}

return false;

}

bool Has(TData data) const

{

Iterator iter(this);

while (iter.Next())

{

if (iter.Data() == data)

{

return true;

}

}

return false;

}

void MoveTo(SListBase \* list)

{

while (!Empty())

{

this->MoveHeadTo(list);

}

}

void MoveHeadTo(SListBase \* list)

{

Assert(!this->Empty());

NodeBase \* node = this->Next();

this->Next() = node->Next();

node->Next() = list->Next();

list->Next() = node;

list->IncrementCount();

this->DecrementCount();

}

// Moves the first element that satisfies the predicate to the toList

template<class Fn>

TData\* MoveTo(SListBase\* toList, Fn predicate)

{

Assert(this != toList);

EditingIterator iter(this);

while (iter.Next())

{

if (predicate(iter.Data()))

{

TData\* data = &iter.Data();

iter.MoveCurrentTo(toList);

return data;

}

}

return nullptr;

}

template<class Fn>

TData\* Find(Fn predicate)

{

Iterator iter(this);

while(iter.Next())

{

if(predicate(iter.Data()))

{

return &iter.Data();

}

}

return nullptr;

}

template<class Fn>

void Iterate(Fn fn)

{

Iterator iter(this);

while(iter.Next())

{

fn(iter.Data());

}

}

void Reverse()

{

NodeBase \* prev = this;

NodeBase \* current = this->Next();

while (!this->IsHead(current))

{

NodeBase \* next = current->Next();

current->Next() = prev;

prev = current;

current = next;

}

current->Next() = prev;

}

bool Equals(SListBase const& other)

{

SListBase<TData>::Iterator iter(this);

SListBase<TData>::Iterator iter2(&other);

while (iter.Next())

{

if (!iter2.Next() || iter.Data() != iter2.Data())

{

return false;

}

}

return !iter2.Next();

}

template <typename TAllocator>

bool CopyTo(TAllocator \* allocator, SListBase& to) const

{

return CopyTo<DefaultCopyElement>(allocator, to);

}

template <void (\*CopyElement)(TData const& from, TData& to), typename TAllocator>

bool CopyTo(TAllocator \* allocator, SListBase& to) const

{

to.Clear(allocator);

SListBase::Iterator iter(this);

NodeBase \*\* next = &to.Next();

while (iter.Next())

{

Node \* node = AllocatorNew(TAllocator, allocator, Node);

if (node == nullptr)

{

return false;

}

CopyElement(iter.Data(), node->data);

\*next = node;

next = &node->Next();

\*next = &to; // Do this every time, in case an OOM exception occurs, to keep the list correct

to.IncrementCount();

}

return true;

}

template <class Fn>

void Map(Fn fn) const

{

MapUntil([fn](TData& data) { fn(data); return false; });

}

template <class Fn>

bool MapUntil(Fn fn) const

{

Iterator iter(this);

while (iter.Next())

{

if (fn(iter.Data()))

{

return true;

}

}

return false;

}

private:

static void DefaultCopyElement(TData const& from, TData& to) { to = from; }

// disable copy constructor

SListBase(SListBase const& list);

};

template <typename TData>

class SListBaseCounted : public SListBase<TData, RealCount>

{

};

template <typename TData, typename TAllocator = ArenaAllocator, typename TCount = DefaultCount>

class SList : public SListBase<TData, TCount>

{

public:

class EditingIterator : public SListBase::EditingIterator

{

public:

EditingIterator() : SListBase::EditingIterator() {}

EditingIterator(SList \* list) : SListBase::EditingIterator(list) {}

void RemoveCurrent()

{

\_\_super::RemoveCurrent(Allocator());

}

TData \* InsertNodeBefore()

{

return \_\_super::InsertNodeBefore(Allocator());

}

bool InsertBefore(TData const& data)

{

return \_\_super::InsertBefore(Allocator(), data);

}

private:

TAllocator \* Allocator() const

{

return ((SList const \*)list)->allocator;

}

};

explicit SList(TAllocator \* allocator) : allocator(allocator) {}

~SList()

{

Clear();

}

void Clear()

{

\_\_super::Clear(allocator);

}

bool Prepend(TData const& data)

{

return \_\_super::Prepend(allocator, data);

}

TData \* PrependNode()

{

return \_\_super::PrependNode(allocator);

}

template <typename TParam>

TData \* PrependNode(TParam param)

{

return \_\_super::PrependNode(allocator, param);

}

template <typename TParam1, typename TParam2>

TData \* PrependNode(TParam1 param1, TParam2 param2)

{

return \_\_super::PrependNode(allocator, param1, param2);

}

void RemoveHead()

{

\_\_super::RemoveHead(allocator);

}

bool Remove(TData const& data)

{

return \_\_super::Remove(allocator, data);

}

// Stack like interface

bool Push(TData const& data)

{

return Prepend(data);

}

TData Pop()

{

TData data = Head();

RemoveHead();

return data;

}

TData const& Top() const

{

return Head();

}

TData& Top()

{

return Head();

}

private:

TAllocator \* allocator;

};

template <typename TData, typename TAllocator = ArenaAllocator>

class SListCounted : public SList<TData, TAllocator, RealCount>

{

public:

explicit SListCounted(TAllocator \* allocator) : SList(allocator) {}

};

#define \_FOREACH\_LIST\_ENTRY\_EX(List, T, Iterator, iter, data, list) \

List<T>::Iterator iter(list); \

while (iter.Next()) \

{ \

T& data = iter.Data();

#define \_NEXT\_LIST\_ENTRY\_EX \

}

#define \_FOREACH\_LIST\_ENTRY(List, T, data, list) { \_FOREACH\_LIST\_ENTRY\_EX(List, T, Iterator, \_\_iter, data, list)

#define \_NEXT\_LIST\_ENTRY \_NEXT\_LIST\_ENTRY\_EX }

#define FOREACH\_SLISTBASE\_ENTRY(T, data, list) \_FOREACH\_LIST\_ENTRY(SListBase, T, data, list)

#define NEXT\_SLISTBASE\_ENTRY \_NEXT\_LIST\_ENTRY

#define FOREACH\_SLISTBASE\_ENTRY\_EDITING(T, data, list, iter) \_FOREACH\_LIST\_ENTRY\_EX(SListBase, T, EditingIterator, iter, data, list)

#define NEXT\_SLISTBASE\_ENTRY\_EDITING \_NEXT\_LIST\_ENTRY\_EX

#define FOREACH\_SLISTBASECOUNTED\_ENTRY(T, data, list) \_FOREACH\_LIST\_ENTRY(SListBaseCounted, T, data, list)

#define NEXT\_SLISTBASECOUNTED\_ENTRY \_NEXT\_LIST\_ENTRY

#define FOREACH\_SLISTBASECOUNTED\_ENTRY\_EDITING(T, data, list, iter) \_FOREACH\_LIST\_ENTRY\_EX(SListBaseCounted, T, EditingIterator, iter, data, list)

#define NEXT\_SLISTBASECOUNTED\_ENTRY\_EDITING \_NEXT\_LIST\_ENTRY\_EX

#define FOREACH\_SLIST\_ENTRY(T, data, list) \_FOREACH\_LIST\_ENTRY(SList, T, data, list)

#define NEXT\_SLIST\_ENTRY \_NEXT\_LIST\_ENTRY

#define FOREACH\_SLIST\_ENTRY\_EDITING(T, data, list, iter) \_FOREACH\_LIST\_ENTRY\_EX(SList, T, EditingIterator, iter, data, list)

#define NEXT\_SLIST\_ENTRY\_EDITING \_NEXT\_LIST\_ENTRY\_EX

#define FOREACH\_SLISTCOUNTED\_ENTRY(T, data, list) \_FOREACH\_LIST\_ENTRY(SListCounted, T, data, list)

#define NEXT\_SLISTCOUNTED\_ENTRY \_NEXT\_LIST\_ENTRY

#define FOREACH\_SLISTCOUNTED\_ENTRY\_EDITING(T, data, list, iter) \_FOREACH\_LIST\_ENTRY\_EX(SListCounted, T, EditingIterator, iter, data, list)

#define NEXT\_SLISTCOUNTED\_ENTRY\_EDITING \_NEXT\_LIST\_ENTRY\_EX

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template<class T>

class SAChunk

{

public:

SAChunk<T> \* next;

uint32 startIndex;

T \* data[];

};

template<class T>

class SparseArray

{

private:

ArenaAllocator \* alloc;

uint32 chunkSize;

SAChunk<T> \* firstChunk;

public:

static SparseArray<T> \* New(ArenaAllocator \*allocator, uint32 chunkSize)

{

SparseArray<T> \* array;

if (!Math::IsPow2(chunkSize))

{

chunkSize = Math::NextPowerOf2(chunkSize);

}

// Throw early if this overflows, since chunkSize never changes, subsequent operations will be safe

UInt32Math::MulAdd<sizeof(T\*), sizeof(SAChunk<T>)>(chunkSize);

array = Anew(allocator, SparseArray<T>);

array->alloc = allocator;

array->chunkSize = chunkSize;

array->firstChunk = NULL;

return array;

}

void Set(uint32 index, T \*element)

{

SAChunk<T> \* chunk, \*\*pPrev = &(this->firstChunk);

uint32 indexInChunk = (index % this->chunkSize);

for (chunk = this->firstChunk; chunk; chunk = chunk->next)

{

if (index < chunk->startIndex)

{

// Need a new chunk...

chunk = NULL;

break;

}

if (index < chunk->startIndex + this->chunkSize)

{

break;

}

pPrev = &(chunk->next);

}

if (chunk == NULL)

{

chunk = (SAChunk<T> \*)this->alloc->AllocZero(sizeof(SAChunk<T>) + (chunkSize \* sizeof(T \*)));

chunk->startIndex = index - indexInChunk;

// Since startIndex and chunkSize don't change, check now if this overflows.

// Cache the result or save memory ?

UInt32Math::Add(chunk->startIndex, chunkSize);

chunk->next = \*pPrev;

\*pPrev = chunk;

}

chunk->data[indexInChunk] = element;

}

T \* Get(uint32 index)

{

SAChunk<T> \* chunk;

uint32 indexInChunk = (index % this->chunkSize);

for (chunk = this->firstChunk; chunk; chunk = chunk->next)

{

if (index < chunk->startIndex)

{

return NULL;

}

if (index < chunk->startIndex + this->chunkSize)

{

return chunk->data[indexInChunk];

}

}

return NULL;

}

SparseArray<T> \* Copy()

{

SparseArray<T> \* newSA = SparseArray<T>::New(this->alloc, this->chunkSize);

SAChunk<T> \* chunk, \*pred = NULL;

for (chunk = this->firstChunk; chunk; chunk = chunk->next)

{

SAChunk<T> \*newChunk = (SAChunk<T> \*)this->alloc->Alloc(sizeof(SAChunk<T>) + (sizeof(T \*) \* this->chunkSize));

newChunk->startIndex = chunk->startIndex;

js\_memcpy\_s(newChunk->data, sizeof(T \*) \* this->chunkSize, chunk->data, sizeof(T \*) \* this->chunkSize);

if (pred)

{

pred->next = newChunk;

}

else

{

newSA->firstChunk = newChunk;

}

pred = newChunk;

}

if (pred)

{

pred->next = NULL;

}

else

{

newSA->firstChunk = NULL;

}

return newSA;

}

void And(SparseArray<T> \*this2)

{

SAChunk<T> \* chunk, \*pred = NULL;

SAChunk<T> \* chunk2;

AssertMsg(this->chunkSize == this2->chunkSize, "Anding incompatible arrays");

chunk2 = this2->firstChunk;

for (chunk = this->firstChunk; chunk; chunk = chunk->next)

{

while (chunk2 && chunk->startIndex > chunk2->startIndex)

{

chunk2 = chunk2->next;

}

if (chunk2 == NULL || chunk->startIndex < chunk2->startIndex)

{

if (pred)

{

pred->next = chunk->next;

}

else

{

this->firstChunk = chunk->next;

}

continue;

}

AssertMsg(chunk->startIndex == chunk2->startIndex, "Huh??");

for (int i = 0; i < this->chunkSize; i++)

{

if (chunk->data[i])

{

if (chunk2->data[i])

{

if (\*(chunk->data[i]) == \*(chunk2->data[i]))

{

continue;

}

}

chunk->data[i] = NULL;

}

}

chunk2 = chunk2->next;

pred = chunk;

}

}

void Clear()

{

this->firstChunk = NULL;

}

#if DBG\_DUMP

void Dump()

{

for (SAChunk<T> \*chunk = this->firstChunk; chunk; chunk = chunk->next)

{

for (int index = chunk->startIndex; index < this->chunkSize; index++)

{

if (chunk->data[index])

{

Output::Print(L"Index %4d => ", index);

chunk->data[index]->Dump();

Output::Print(L"\n");

}

}

}

}

#endif

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonDataStructuresPch.h"

BVSparseNode::BVSparseNode(BVIndex beginIndex, BVSparseNode \* nextNode) :

startIndex(beginIndex),

data(0),

next(nextNode)

{

}

void BVSparseNode::init(BVIndex beginIndex, BVSparseNode \* nextNode)

{

this->startIndex = beginIndex;

this->data = 0;

this->next = nextNode;

}

bool BVSparseNode::ToString(

\_\_out\_ecount(strSize) char \*const str,

const size\_t strSize,

size\_t \*const writtenLengthRef,

const bool isInSequence,

const bool isFirstInSequence,

const bool isLastInSequence) const

{

Assert(str);

Assert(!isFirstInSequence || isInSequence);

Assert(!isLastInSequence || isInSequence);

if (strSize == 0)

{

if (writtenLengthRef)

{

\*writtenLengthRef = 0;

}

return false;

}

str[0] = '\0';

const size\_t reservedLength = \_countof(", ...}");

if (strSize <= reservedLength)

{

if (writtenLengthRef)

{

\*writtenLengthRef = 0;

}

return false;

}

size\_t length = 0;

if (!isInSequence || isFirstInSequence)

{

str[length++] = '{';

}

bool insertComma = isInSequence && !isFirstInSequence;

char tempStr[13];

for (BVIndex i = data.GetNextBit(); i != BVInvalidIndex; i = data.GetNextBit(i + 1))

{

const size\_t copyLength = sprintf\_s(tempStr, insertComma ? ", %u" : "%u", startIndex + i);

Assert(static\_cast<int>(copyLength) > 0);

Assert(strSize > length);

Assert(strSize - length > reservedLength);

if (strSize - length - reservedLength <= copyLength)

{

strcpy\_s(&str[length], strSize - length, insertComma ? ", ...}" : "...}");

if (writtenLengthRef)

{

\*writtenLengthRef = length + (insertComma ? \_countof(", ...}") : \_countof("...}"));

}

return false;

}

strcpy\_s(&str[length], strSize - length - reservedLength, tempStr);

length += copyLength;

insertComma = true;

}

if (!isInSequence || isLastInSequence)

{

Assert(\_countof("}") < strSize - length);

strcpy\_s(&str[length], strSize - length, "}");

length += \_countof("}");

}

if (writtenLengthRef)

{

\*writtenLengthRef = length;

}

return true;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#if defined(\_M\_ARM64) || defined(\_M\_X64)

typedef BVUnit32 SparseBVUnit;

#else

typedef BVUnit64 SparseBVUnit;

#endif

#define FOREACH\_BITSET\_IN\_SPARSEBV(index, bv) \

{ \

BVIndex index; \

for(BVSparseNode \* \_curNode = (bv)->head; \_curNode != 0 ; \_curNode = \_curNode->next) \

{ \

BVIndex \_offset; \

BVIndex \_startIndex = \_curNode->startIndex; \

SparseBVUnit \_unit = \_curNode->data; \

for(\_offset = \_unit.GetNextBit(); \_offset != -1; \_offset = \_unit.GetNextBit()) \

{ \

index = \_startIndex + \_offset; \

\_unit.Clear(\_offset); \

\

#define BREAK\_BITSET\_IN\_SPARSEBV \

\_curNode = 0; \

break;

#define NEXT\_BITSET\_IN\_SPARSEBV \

} \

if(\_curNode == 0) \

{ \

break; \

} \

} \

}

#define FOREACH\_BITSET\_IN\_SPARSEBV\_EDITING(index, bv) \

{ \

BVIndex index; \

BVSparseNode \* \_curNodeEdit = (bv)->head; \

while (\_curNodeEdit != nullptr) \

{ \

BVSparseNode \* \_next = \_curNodeEdit->next; \

BVIndex \_offset; \

BVIndex \_startIndex = \_curNodeEdit->startIndex; \

SparseBVUnit \_unit = \_curNodeEdit->data; \

for(\_offset = \_unit.GetNextBit(); \_offset != -1; \_offset = \_unit.GetNextBit()) \

{ \

index = \_startIndex + \_offset; \

\_unit.Clear(\_offset); \

\

#define NEXT\_BITSET\_IN\_SPARSEBV\_EDITING \

} \

\_curNodeEdit = \_next; \

} \

}

#define SPARSEBV\_CLEAR\_CURRENT\_BIT() \_curNodeEdit->data.Clear(\_offset)

struct BVSparseNode

{

BVIndex startIndex;

#if defined(\_M\_ARM64) || defined(\_M\_X64)

//64-bit: the order is changed to make sure it fits in 16 bytes

SparseBVUnit data;

BVSparseNode \* next;

#else //\_M\_IX86 and \_M\_ARM32

BVSparseNode \* next;

SparseBVUnit data;

#endif

BVSparseNode(BVIndex beginIndex, BVSparseNode \* nextNode);

void init(BVIndex beginIndex, BVSparseNode \* nextNode);

bool ToString(

\_\_out\_ecount(strSize) char \*const str,

const size\_t strSize,

size\_t \*const writtenLengthRef = nullptr,

const bool isInSequence = false,

const bool isFirstInSequence = false,

const bool isLastInSequence = false) const;

};

CompileAssert(sizeof(BVSparseNode) == 16); // Performance assert, BVSparseNode is heavily used in the backend, do perf measurement before changing this.

template <class TAllocator>

class BVSparse

{

// Data

public:

BVSparseNode \* head;

private:

TAllocator \* alloc;

BVSparseNode \*\* lastUsedNodePrevNextField;

static const SparseBVUnit s\_EmptyUnit;

// Constructor

public:

BVSparse(TAllocator\* allocator);

~BVSparse();

// Implementation

protected:

template <class TOtherAllocator>

static void AssertBV(const BVSparse<TOtherAllocator> \* bv);

SparseBVUnit \* BitsFromIndex(BVIndex i, bool create = true);

BVSparseNode\* NodeFromIndex(BVIndex i, BVSparseNode \*\*\* prevNextFieldOut, bool create = true);

BVSparseNode \* DeleteNode(BVSparseNode \*node, bool bResetLastUsed = true);

void QueueInFreeList(BVSparseNode\* node);

BVSparseNode \* Allocate(const BVIndex searchIndex, BVSparseNode \*prevNode);

template<void (SparseBVUnit::\*callback)(SparseBVUnit)>

void for\_each(const BVSparse<TAllocator> \*bv2);

template<void (SparseBVUnit::\*callback)(SparseBVUnit)>

void for\_each(const BVSparse<TAllocator> \*bv1, const BVSparse<TAllocator> \*bv2);

// Methods

public:

BOOLEAN operator[](BVIndex i) const;

BOOLEAN Test(BVIndex i);

BVIndex GetNextBit(BVIndex i) const;

BVIndex GetNextBit(BVSparseNode \* node) const;

BOOLEAN TestEmpty() const;

BOOLEAN TestAndSet(BVIndex i);

BOOLEAN TestAndClear(BVIndex i);

void Set(BVIndex i);

void Clear(BVIndex i);

void Compliment(BVIndex i);

// this |= bv;

void Or(const BVSparse<TAllocator> \*bv);

// this = bv1 | bv2;

void Or(const BVSparse<TAllocator> \*bv1, const BVSparse<TAllocator> \*bv2);

// newBv = this | bv;

BVSparse<TAllocator> \* OrNew(const BVSparse<TAllocator> \*bv, TAllocator\* allocator) const;

BVSparse<TAllocator> \* OrNew(const BVSparse<TAllocator> \*bv) const { return this->OrNew(bv, this->alloc); }

// this &= bv;

void And(const BVSparse<TAllocator> \*bv);

// this = bv1 & bv2;

void And(const BVSparse<TAllocator> \*bv1, const BVSparse<TAllocator> \*bv2);

// newBv = this & bv;

BVSparse<TAllocator> \* AndNew(const BVSparse<TAllocator> \*bv, TAllocator\* allocator) const;

BVSparse<TAllocator> \* AndNew(const BVSparse<TAllocator> \*bv) const { return this->AndNew(bv, this->alloc); }

// this ^= bv;

void Xor(const BVSparse<TAllocator> \*bv);

// this = bv1 ^ bv2;

void Xor(const BVSparse<TAllocator> \*bv1, const BVSparse<TAllocator> \*bv2);

// newBv = this ^ bv;

BVSparse<TAllocator> \* XorNew(const BVSparse<TAllocator> \*bv, TAllocator\* allocator) const;

BVSparse<TAllocator> \* XorNew(const BVSparse<TAllocator> \*bv) const { return this->XorNew(bv, this->alloc); }

// this -= bv;

void Minus(const BVSparse<TAllocator> \*bv);

// this = bv1 - bv2;

void Minus(const BVSparse<TAllocator> \*bv1, const BVSparse<TAllocator> \*bv2);

// newBv = this - bv;

BVSparse<TAllocator> \* MinusNew(const BVSparse<TAllocator> \*bv, TAllocator\* allocator) const;

BVSparse<TAllocator> \* MinusNew(const BVSparse<TAllocator> \*bv) const { return this->MinusNew(bv, this->alloc); }

template <class TSrcAllocator>

void Copy(const BVSparse<TSrcAllocator> \*bv);

BVSparse<TAllocator> \* CopyNew(TAllocator\* allocator) const;

BVSparse<TAllocator> \* CopyNew() const;

void ComplimentAll();

void ClearAll();

BVIndex Count() const;

bool IsEmpty() const;

bool Equal(BVSparse<TAllocator> const \* bv) const;

// this & bv != empty

bool Test(BVSparse const \* bv) const;

void ToString(\_\_out\_ecount(strSize) char \*const str, const size\_t strSize) const;

template<class F> void ToString(\_\_out\_ecount(strSize) char \*const str, const size\_t strSize, const F ReadNode) const;

TAllocator \* GetAllocator() const { return alloc; }

#if DBG\_DUMP

void Dump() const;

#endif

};

#if DBG\_DUMP

template <typename T> void Dump(T const& t);

namespace Memory{ class JitArenaAllocator; }

template<>

inline void Dump(BVSparse<JitArenaAllocator> \* const& bv)

{

bv->Dump();

}

namespace Memory { class Recycler; }

template<>

inline void Dump(BVSparse<Recycler> \* const& bv)

{

bv->Dump();

}

#endif

template <class TAllocator>

const SparseBVUnit BVSparse<TAllocator>::s\_EmptyUnit(0);

template <class TAllocator>

BVSparse<TAllocator>::BVSparse(TAllocator\* allocator) :

alloc(allocator),

head(nullptr)

{

this->lastUsedNodePrevNextField = &this->head;

}

template <class TAllocator>

void

BVSparse<TAllocator>::QueueInFreeList(BVSparseNode \*curNode)

{

AllocatorDelete(TAllocator, this->alloc, curNode);

}

template <class TAllocator>

BVSparseNode \*

BVSparse<TAllocator>::Allocate(const BVIndex searchIndex, BVSparseNode \*nextNode)

{

return AllocatorNew(TAllocator, this->alloc, BVSparseNode, searchIndex, nextNode);

}

template <class TAllocator>

BVSparse<TAllocator>::~BVSparse()

{

BVSparseNode \* curNode = this->head;

while (curNode != nullptr)

{

curNode = this->DeleteNode(curNode);

}

}

// Searches for a node which would contain the required bit. If not found, then it inserts

// a new node in the appropriate position.

//

template <class TAllocator>

BVSparseNode \*

BVSparse<TAllocator>::NodeFromIndex(BVIndex i, BVSparseNode \*\*\* prevNextFieldOut, bool create)

{

const BVIndex searchIndex = SparseBVUnit::Floor(i);

BVSparseNode \*\* prevNextField = this->lastUsedNodePrevNextField;

BVSparseNode \* curNode = (\*prevNextField);

if (curNode != nullptr)

{

if (curNode->startIndex == searchIndex)

{

\*prevNextFieldOut = prevNextField;

return curNode;

}

if (curNode->startIndex > searchIndex)

{

prevNextField = &this->head;

curNode = this->head;

}

}

else

{

prevNextField = &this->head;

curNode = this->head;

}

for (; curNode && searchIndex > curNode->startIndex; curNode = curNode->next)

{

prevNextField = &curNode->next;

}

if(curNode && searchIndex == curNode->startIndex)

{

\*prevNextFieldOut = prevNextField;

this->lastUsedNodePrevNextField = prevNextField;

return curNode;

}

if(!create)

{

return nullptr;

}

BVSparseNode \* newNode = Allocate(searchIndex, \*prevNextField);

\*prevNextField = newNode;

\*prevNextFieldOut = prevNextField;

this->lastUsedNodePrevNextField = prevNextField;

return newNode;

}

template <class TAllocator>

SparseBVUnit \*

BVSparse<TAllocator>::BitsFromIndex(BVIndex i, bool create)

{

BVSparseNode \*\* prevNextField;

BVSparseNode \* node = NodeFromIndex(i, &prevNextField, create);

if (node)

{

return &node->data;

}

else

{

return (SparseBVUnit \*)&BVSparse::s\_EmptyUnit;

}

}

template <class TAllocator>

BVSparseNode \*

BVSparse<TAllocator>::DeleteNode(BVSparseNode \*node, bool bResetLastUsed)

{

BVSparseNode \*next = node->next;

QueueInFreeList(node);

if (bResetLastUsed)

{

this->lastUsedNodePrevNextField = &this->head;

}

else

{

Assert(this->lastUsedNodePrevNextField != &node->next);

}

return next;

}

template <class TAllocator>

BVIndex

BVSparse<TAllocator>::GetNextBit(BVSparseNode \*node) const

{

while(0 != node)

{

BVIndex ret = node->data.GetNextBit();

if(-1 != ret)

{

return ret + node->startIndex;

}

}

return -1;

}

template <class TAllocator>

BVIndex

BVSparse<TAllocator>::GetNextBit(BVIndex i) const

{

const BVIndex startIndex = SparseBVUnit::Floor(i);

for(BVSparseNode \* node = this->head; node != 0 ; node = node->next)

{

if(startIndex == node->startIndex)

{

BVIndex ret = node->data.GetNextBit(SparseBVUnit::Offset(i));

if(-1 != ret)

{

return ret + node->startIndex;

}

else

{

return GetNextBit(node->next);

}

}

else if(startIndex < node->startIndex)

{

return GetNextBit(node->next);

}

}

return -1;

}

template <class TAllocator>

template <class TOtherAllocator>

void

BVSparse<TAllocator>::AssertBV(const BVSparse<TOtherAllocator> \*bv)

{

AssertMsg(nullptr != bv, "Cannot operate on NULL bitvector");

}

template <class TAllocator>

void

BVSparse<TAllocator>::ClearAll()

{

BVSparseNode\* nextNode;

for(BVSparseNode \* node = this->head; node != 0 ; node = nextNode)

{

nextNode = node->next;

QueueInFreeList(node);

}

this->head = nullptr;

this->lastUsedNodePrevNextField = &this->head;

}

template <class TAllocator>

void

BVSparse<TAllocator>::Set(BVIndex i)

{

this->BitsFromIndex(i)->Set(SparseBVUnit::Offset(i));

}

template <class TAllocator>

void

BVSparse<TAllocator>::Clear(BVIndex i)

{

BVSparseNode \*\* prevNextField;

BVSparseNode \* current = this->NodeFromIndex(i, &prevNextField, false /\* create \*/);

if(current)

{

current->data.Clear(SparseBVUnit::Offset(i));

if (current->data.IsEmpty())

{

\*prevNextField = this->DeleteNode(current, false);

}

}

}

template <class TAllocator>

void

BVSparse<TAllocator>::Compliment(BVIndex i)

{

this->BitsFromIndex(i)->Complement(SparseBVUnit::Offset(i));

}

template <class TAllocator>

BOOLEAN

BVSparse<TAllocator>::TestEmpty() const

{

return this->head != nullptr;

}

template <class TAllocator>

BOOLEAN

BVSparse<TAllocator>::Test(BVIndex i)

{

return this->BitsFromIndex(i, false)->Test(SparseBVUnit::Offset(i));

}

template <class TAllocator>

BOOLEAN

BVSparse<TAllocator>::TestAndSet(BVIndex i)

{

SparseBVUnit \* bvUnit = this->BitsFromIndex(i);

BVIndex bvIndex = SparseBVUnit::Offset(i);

BOOLEAN bit = bvUnit->Test(bvIndex);

bvUnit->Set(bvIndex);

return bit;

}

template <class TAllocator>

BOOLEAN

BVSparse<TAllocator>::TestAndClear(BVIndex i)

{

BVSparseNode \*\* prevNextField;

BVSparseNode \* current = this->NodeFromIndex(i, &prevNextField);

BVIndex bvIndex = SparseBVUnit::Offset(i);

BOOLEAN bit = current->data.Test(bvIndex);

current->data.Clear(bvIndex);

if (current->data.IsEmpty())

{

\*prevNextField = this->DeleteNode(current, false);

}

return bit;

}

template <class TAllocator>

BOOLEAN

BVSparse<TAllocator>::operator[](BVIndex i) const

{

return this->Test(i);

}

template<class TAllocator>

template<void (SparseBVUnit::\*callback)(SparseBVUnit)>

void BVSparse<TAllocator>::for\_each(const BVSparse \*bv2)

{

Assert(callback == &SparseBVUnit::And || callback == &SparseBVUnit::Or || callback == &SparseBVUnit::Xor || callback == &SparseBVUnit::Minus);

AssertBV(bv2);

BVSparseNode \* node1 = this->head;

const BVSparseNode \* node2 = bv2->head;

BVSparseNode \*\* prevNodeNextField = &this->head;

while(node1 != nullptr && node2 != nullptr)

{

if(node2->startIndex == node1->startIndex)

{

(node1->data.\*callback)(node2->data);

prevNodeNextField = &node1->next;

node1 = node1->next;

node2 = node2->next;

}

else if(node2->startIndex > node1->startIndex)

{

if (callback == &SparseBVUnit::And)

{

node1 = this->DeleteNode(node1);

\*prevNodeNextField = node1;

}

else

{

prevNodeNextField = &node1->next;

node1 = node1->next;

}

}

else

{

if (callback == &SparseBVUnit::Or || callback == &SparseBVUnit::Xor)

{

BVSparseNode \* newNode = Allocate(node2->startIndex, node1);

(newNode->data.\*callback)(node2->data);

\*prevNodeNextField = newNode;

prevNodeNextField = &newNode->next;

}

node2 = node2->next;

}

}

if (callback == &SparseBVUnit::And)

{

while (node1 != nullptr)

{

node1 = this->DeleteNode(node1);

}

\*prevNodeNextField = nullptr;

}

else if (callback == &SparseBVUnit::Or || callback == &SparseBVUnit::Xor)

{

while(node2 != 0)

{

Assert(\*prevNodeNextField == nullptr);

BVSparseNode \* newNode = Allocate(node2->startIndex, nullptr);

\*prevNodeNextField = newNode;

(newNode->data.\*callback)(node2->data);

node2 = node2->next;

prevNodeNextField = &newNode->next;

}

}

}

template<class TAllocator>

template<void (SparseBVUnit::\*callback)(SparseBVUnit)>

void BVSparse<TAllocator>::for\_each(const BVSparse \*bv1, const BVSparse \*bv2)

{

Assert(callback == &SparseBVUnit::And || callback == &SparseBVUnit::Or || callback == &SparseBVUnit::Xor || callback == &SparseBVUnit::Minus);

Assert(this->IsEmpty());

AssertBV(bv1);

AssertBV(bv2);

BVSparseNode \* node1 = bv1->head;

const BVSparseNode \* node2 = bv2->head;

BVSparseNode \* lastNode = nullptr;

BVSparseNode \*\* prevNextField = &this->head;

while(node1 != nullptr && node2 != nullptr)

{

lastNode = node1;

BVIndex startIndex;

SparseBVUnit bvUnit1;

SparseBVUnit bvUnit2;

if (node2->startIndex == node1->startIndex)

{

startIndex = node1->startIndex;

bvUnit1 = node1->data;

bvUnit2 = node2->data;

node1 = node1->next;

node2 = node2->next;

}

else if (node2->startIndex > node1->startIndex)

{

startIndex = node1->startIndex;

bvUnit1 = node1->data;

node1 = node1->next;

}

else

{

startIndex = node2->startIndex;

bvUnit2 = node2->data;

node2 = node2->next;

}

(bvUnit1.\*callback)(bvUnit2);

if (!bvUnit1.IsEmpty())

{

BVSparseNode \* newNode = Allocate(startIndex, nullptr);

newNode->data = bvUnit1;

\*prevNextField = newNode;

prevNextField = &newNode->next;

}

}

if (callback == &SparseBVUnit::Minus || callback == &SparseBVUnit::Or || callback == &SparseBVUnit::Xor)

{

BVSparseNode const \* copyNode = (callback == &SparseBVUnit::Minus || node1 != nullptr)? node1 : node2;

while (copyNode != nullptr)

{

if (!copyNode->data.IsEmpty())

{

BVSparseNode \* newNode = Allocate(copyNode->startIndex, nullptr);

newNode->data = copyNode->data;

\*prevNextField = newNode;

prevNextField = &newNode->next;

}

copyNode = copyNode->next;

}

}

}

template <class TAllocator>

void

BVSparse<TAllocator>::Or(const BVSparse\*bv)

{

this->for\_each<&SparseBVUnit::Or>(bv);

}

template <class TAllocator>

void

BVSparse<TAllocator>::Or(const BVSparse \* bv1, const BVSparse \* bv2)

{

this->ClearAll();

this->for\_each<&SparseBVUnit::Or>(bv1, bv2);

}

template <class TAllocator>

BVSparse<TAllocator> \*

BVSparse<TAllocator>::OrNew(const BVSparse\* bv, TAllocator\* allocator) const

{

BVSparse \* newBv = AllocatorNew(TAllocator, allocator, BVSparse, allocator);

newBv->for\_each<&SparseBVUnit::Or>(this, bv);

return newBv;

}

template <class TAllocator>

void

BVSparse<TAllocator>::And(const BVSparse\*bv)

{

this->for\_each<&SparseBVUnit::And>(bv);

}

template <class TAllocator>

void

BVSparse<TAllocator>::And(const BVSparse \* bv1, const BVSparse \* bv2)

{

this->ClearAll();

this->for\_each<&SparseBVUnit::And>(bv1, bv2);

}

template <class TAllocator>

BVSparse<TAllocator> \*

BVSparse<TAllocator>::AndNew(const BVSparse\* bv, TAllocator\* allocator) const

{

BVSparse \* newBv = AllocatorNew(TAllocator, allocator, BVSparse, allocator);

newBv->for\_each<&SparseBVUnit::And>(this, bv);

return newBv;

}

template <class TAllocator>

void

BVSparse<TAllocator>::Xor(const BVSparse\*bv)

{

this->for\_each<&SparseBVUnit::Xor>(bv);

}

template <class TAllocator>

void

BVSparse<TAllocator>::Xor(const BVSparse \* bv1, const BVSparse \* bv2)

{

this->ClearAll();

this->for\_each<&SparseBVUnit::Xor>(bv1, bv2);

}

template <class TAllocator>

BVSparse<TAllocator> \*

BVSparse<TAllocator>::XorNew(const BVSparse\* bv, TAllocator\* allocator) const

{

BVSparse \* newBv = AllocatorNew(TAllocator, allocator, BVSparse, allocator);

newBv->for\_each<&SparseBVUnit::Xor>(this, bv);

return newBv;

}

template <class TAllocator>

void

BVSparse<TAllocator>::Minus(const BVSparse\*bv)

{

this->for\_each<&SparseBVUnit::Minus>(bv);

}

template <class TAllocator>

void

BVSparse<TAllocator>::Minus(const BVSparse \* bv1, const BVSparse \* bv2)

{

this->ClearAll();

this->for\_each<&SparseBVUnit::Minus>(bv1, bv2);

}

template <class TAllocator>

BVSparse<TAllocator> \*

BVSparse<TAllocator>::MinusNew(const BVSparse\* bv, TAllocator\* allocator) const

{

BVSparse \* newBv = AllocatorNew(TAllocator, allocator, BVSparse, allocator);

newBv->for\_each<&SparseBVUnit::Minus>(this, bv);

return newBv;

}

template <class TAllocator>

template <class TSrcAllocator>

void

BVSparse<TAllocator>::Copy(const BVSparse<TSrcAllocator> \* bv2)

{

AssertBV(bv2);

BVSparseNode \* node1 = this->head;

const BVSparseNode \* node2 = bv2->head;

BVSparseNode \*\* prevNextField = &this->head;

while (node1 != nullptr && node2 != nullptr)

{

if (!node2->data.IsEmpty())

{

node1->startIndex = node2->startIndex;

node1->data.Copy(node2->data);

prevNextField = &node1->next;

node1 = node1->next;

}

node2 = node2->next;

}

if (node1 != nullptr)

{

while (node1 != nullptr)

{

node1 = this->DeleteNode(node1);

}

\*prevNextField = nullptr;

}

else

{

while (node2 != nullptr)

{

if (!node2->data.IsEmpty())

{

BVSparseNode \* newNode = Allocate(node2->startIndex, nullptr);

newNode->data.Copy(node2->data);

\*prevNextField = newNode;

prevNextField = &newNode->next;

}

node2 = node2->next;

}

}

}

template <class TAllocator>

BVSparse<TAllocator> \*

BVSparse<TAllocator>::CopyNew(TAllocator\* allocator) const

{

BVSparse \* bv = AllocatorNew(TAllocator, allocator, BVSparse, allocator);

bv->Copy(this);

return bv;

}

template <class TAllocator>

BVSparse<TAllocator> \*

BVSparse<TAllocator>::CopyNew() const

{

return this->CopyNew(this->alloc);

}

template <class TAllocator>

void

BVSparse<TAllocator>::ComplimentAll()

{

for(BVSparseNode \* node = this->head; node != 0 ; node = node->next)

{

node->data.ComplimentAll();

}

}

template <class TAllocator>

BVIndex

BVSparse<TAllocator>::Count() const

{

BVIndex sum = 0;

for(BVSparseNode \* node = this->head; node != 0 ; node = node->next)

{

sum += node->data.Count();

}

return sum;

}

template <class TAllocator>

bool

BVSparse<TAllocator>::IsEmpty() const

{

for(BVSparseNode \* node = this->head; node != 0 ; node = node->next)

{

if (!node->data.IsEmpty())

{

return false;

}

}

return true;

}

template <class TAllocator>

bool

BVSparse<TAllocator>::Equal(BVSparse const \* bv) const

{

BVSparseNode const \* bvNode1 = this->head;

BVSparseNode const \* bvNode2 = bv->head;

while (true)

{

while (bvNode1 != nullptr && bvNode1->data.IsEmpty())

{

bvNode1 = bvNode1->next;

}

while (bvNode2 != nullptr && bvNode2->data.IsEmpty())

{

bvNode2 = bvNode2->next;

}

if (bvNode1 == nullptr)

{

return (bvNode2 == nullptr);

}

if (bvNode2 == nullptr)

{

return false;

}

if (bvNode1->startIndex != bvNode2->startIndex)

{

return false;

}

if (!bvNode1->data.Equal(bvNode2->data))

{

return false;

}

bvNode1 = bvNode1->next;

bvNode2 = bvNode2->next;

}

}

template <class TAllocator>

bool

BVSparse<TAllocator>::Test(BVSparse const \* bv) const

{

BVSparseNode const \* bvNode1 = this->head;

BVSparseNode const \* bvNode2 = bv->head;

while (bvNode1 != nullptr && bvNode2 != nullptr)

{

if (bvNode1->data.IsEmpty() || bvNode1->startIndex < bvNode2->startIndex)

{

bvNode1 = bvNode1->next;

continue;

}

if (bvNode2->data.IsEmpty() || bvNode1->startIndex > bvNode2->startIndex)

{

bvNode2 = bvNode2->next;

continue;

}

Assert(bvNode1->startIndex == bvNode2->startIndex);

if (bvNode1->data.Test(bvNode2->data))

{

return true;

}

bvNode1 = bvNode1->next;

bvNode2 = bvNode2->next;

}

return false;

}

template<class TAllocator>

template<class F>

void BVSparse<TAllocator>::ToString(\_\_out\_ecount(strSize) char \*const str, const size\_t strSize, const F ReadNode) const

{

Assert(str);

if(strSize == 0)

{

return;

}

str[0] = '\0';

bool empty = true;

bool isFirstInSequence = true;

size\_t length = 0;

BVSparseNode \*nodePtr = head;

while(nodePtr)

{

bool readSuccess;

const BVSparseNode node(ReadNode(nodePtr, &readSuccess));

if(!readSuccess)

{

str[0] = '\0';

return;

}

if(node.data.IsEmpty())

{

nodePtr = node.next;

continue;

}

empty = false;

size\_t writtenLength;

if(!node.ToString(&str[length], strSize - length, &writtenLength, true, isFirstInSequence, !node.next))

{

return;

}

length += writtenLength;

isFirstInSequence = false;

nodePtr = node.next;

}

if(empty && \_countof("{}") < strSize)

{

strcpy\_s(str, strSize, "{}");

}

}

template<class TAllocator>

void BVSparse<TAllocator>::ToString(\_\_out\_ecount(strSize) char \*const str, const size\_t strSize) const

{

ToString(

str,

strSize,

[](BVSparseNode \*const nodePtr, bool \*const successRef) -> BVSparseNode

{

Assert(nodePtr);

Assert(successRef);

\*successRef = true;

return \*nodePtr;

});

}

#if DBG\_DUMP

template <class TAllocator>

void

BVSparse<TAllocator>::Dump() const

{

bool hasBits = false;

Output::Print(L"[ ");

for(BVSparseNode \* node = this->head; node != 0 ; node = node->next)

{

hasBits = node->data.Dump(node->startIndex, hasBits);

}

Output::Print(L"]\n");

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

template <

class T,

class Allocator = ArenaAllocator,

bool isLeaf = false,

template <typename Value> class TComparer = DefaultComparer>

class Stack

{

private:

List<T, Allocator, isLeaf, Js::CopyRemovePolicy, TComparer> list;

public:

Stack(Allocator\* alloc) : list(alloc)

{

}

int Count() const { return list.Count(); }

bool Empty() const { return Count() == 0; }

void Clear()

{

list.Clear();

}

bool Contains(const T& item) const

{

return list.Contains(item);

}

const T& Top() const

{

return list.Item(list.Count() - 1);

}

const T& Peek(int stepsBack = 0) const

{

return list.Item(list.Count() - 1 - stepsBack);

}

T Pop()

{

T item = list.Item(list.Count() - 1);

list.RemoveAt(list.Count() - 1);

return item;

}

T Pop(int count)

{

T item = T();

while (count-- > 0)

{

item = Pop();

}

return item;

}

void Push(const T& item)

{

list.Add(item);

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

template <typename TAllocator>

class StringBuilder

{

private:

struct Data

{

public:

union {

struct st\_Single

{

wchar\_t buffer[];

} single;

struct st\_Chained

{

charcount\_t length;

Data \*next;

wchar\_t buffer[];

} chained;

}u;

};

private:

static const charcount\_t MaxLength = INT\_MAX - 1;

const static charcount\_t MaxRealloc = 64;

TAllocator\* alloc;

// First chunk is just a buffer, and which can be detached without copying.

Data \*firstChunk;

// Second chunk is a chained list of chunks. UnChain() needs to be called to copy the first chunk

// and the list of chained chunks to a single buffer on calls to GetBuffer().

Data \*secondChunk;

Data \*lastChunk;

wchar\_t \* appendPtr;

charcount\_t length; // Total capacity (allocated number of elements - 1), in all chunks. Note that we keep one allocated element which is not accounted in length for terminating '\0'.

charcount\_t count; // Total number of elements, in all chunks.

charcount\_t firstChunkLength;

charcount\_t initialSize;

bool IsChained() { return this->secondChunk != NULL; }

Data \*NewChainedChunk(charcount\_t bufLengthRequested)

{

CompileAssert(sizeof(charcount\_t) == sizeof(uint32));

// allocation = (bufLengthRequested \* sizeof(wchar\_t) + sizeof(Data)

charcount\_t alloc32 = UInt32Math::MulAdd<sizeof(wchar\_t), sizeof(Data)>(bufLengthRequested);

size\_t allocation = TAllocator::GetAlignedSize(alloc32);

size\_t size\_t\_length = (allocation - sizeof(Data)) / sizeof(wchar\_t);

charcount\_t bufLength = (charcount\_t)size\_t\_length;

Assert(bufLength == size\_t\_length);

Data \*newChunk = AllocatorNewStructPlus(TAllocator, this->alloc, allocation, Data);

newChunk->u.chained.length = bufLength;

newChunk->u.chained.next = NULL;

// Recycler gives zeroed memory, so rely on that instead of memsetting the tail

#if 0

// Align memset to machine register size for perf

bufLengthRequested &= ~(sizeof(size\_t) - 1);

memset(newChunk->u.chained.buffer + bufLengthRequested, 0, (bufLength - bufLengthRequested) \* sizeof(wchar\_t));

#endif

return newChunk;

}

Data \*NewSingleChunk(charcount\_t \*pBufLengthRequested)

{

Assert(\*pBufLengthRequested <= MaxLength);

// Let's just grow the current chunk in place

CompileAssert(sizeof(charcount\_t) == sizeof(uint32));

//// allocation = (bufLengthRequested+1) \* sizeof(wchar\_t)

charcount\_t alloc32 = UInt32Math::AddMul< 1, sizeof(wchar\_t) >(\*pBufLengthRequested);

size\_t allocation = HeapInfo::GetAlignedSize(alloc32);

size\_t size\_t\_newLength = allocation / sizeof(wchar\_t) - 1;

charcount\_t newLength = (charcount\_t)size\_t\_newLength;

Assert(newLength == size\_t\_newLength);

Assert(newLength <= MaxLength + 1);

if (newLength == MaxLength + 1)

{

// newLength could be MaxLength + 1 because of alignment.

// In this case alloc size is 2 elements more than newLength (normally 1 elements more for NULL), that's fine.

newLength = MaxLength;

}

Assert(newLength <= MaxLength);

Data\* newChunk = AllocatorNewStructPlus(TAllocator, this->alloc, allocation, Data);

newChunk->u.single.buffer[newLength] = L'\0';

\*pBufLengthRequested = newLength;

return newChunk;

}

\_\_declspec(noinline) void ExtendBuffer(charcount\_t newLength)

{

Data \*newChunk;

// To maintain this->length under MaxLength, check it here/throw, this is the only place we grow the buffer.

if (newLength > MaxLength)

{

Throw::OutOfMemory();

}

Assert(this->length <= MaxLength);

charcount\_t newLengthTryGrowPolicy = newLength + (this->length\*2/3); // Note: this would never result in uint32 overflow.

if (newLengthTryGrowPolicy <= MaxLength)

{

newLength = newLengthTryGrowPolicy;

}

Assert(newLength <= MaxLength);

// We already have linked chunks

if (this->IsChained() || (this->firstChunk != NULL && newLength - this->length > MaxRealloc))

{

newChunk = this->NewChainedChunk(newLength - this->count);

if (this->IsChained())

{

this->lastChunk->u.chained.next = newChunk;

// We're not going to use the extra space in the current chunk...

Assert(this->lastChunk->u.chained.length > this->length - this->count);

this->lastChunk->u.chained.length -= (this->length - this->count);

}

else

{

// Time to add our first linked chunk

Assert(this->secondChunk == NULL);

this->secondChunk = newChunk;

// We're not going to use the extra space in the current chunk...

this->firstChunkLength = this->count;

}

this->length = this->count + newChunk->u.chained.length;

this->lastChunk = newChunk;

this->appendPtr = newChunk->u.chained.buffer;

}

else

{

if (this->initialSize < MaxLength)

{

newLength = max(newLength, this->initialSize + 1);

}

else

{

newLength = MaxLength;

}

Assert(newLength <= MaxLength);

// Let's just grow the current chunk in place

newChunk = this->NewSingleChunk(&newLength);

if (this->count)

{

js\_memcpy\_s(newChunk->u.single.buffer, newLength \* sizeof(wchar\_t), this->firstChunk->u.single.buffer, sizeof(wchar\_t) \* this->count);

}

this->firstChunk = this->lastChunk = newChunk;

this->firstChunkLength = newLength;

this->length = newLength;

this->appendPtr = newChunk->u.single.buffer + this->count;

}

}

void EnsureBuffer(charcount\_t countNeeded)

{

if(countNeeded == 0) return;

if (countNeeded >= this->length - this->count)

{

if (countNeeded > MaxLength)

{

// Check upfront to prevent potential uint32 overflow caused by (this->count + countNeeded + 1).

Throw::OutOfMemory();

}

ExtendBuffer(this->count + countNeeded + 1);

}

}

public:

static StringBuilder<TAllocator> \*

New(TAllocator\* alloc, charcount\_t initialSize)

{

if (initialSize > MaxLength)

{

Throw::OutOfMemory();

}

return AllocatorNew(TAllocator, alloc, StringBuilder<TAllocator>, alloc, initialSize);

}

StringBuilder(TAllocator\* alloc)

{

new (this) StringBuilder(alloc, 0);

}

StringBuilder(TAllocator\* alloc, charcount\_t initialSize) : alloc(alloc), length(0), count(0), firstChunk(NULL),

secondChunk(NULL), appendPtr(NULL), initialSize(initialSize)

{

if (initialSize > MaxLength)

{

Throw::OutOfMemory();

}

}

void UnChain(\_\_out \_\_ecount(bufLen) wchar\_t \*pBuf, charcount\_t bufLen)

{

charcount\_t lastChunkCount = this->count;

Assert(this->IsChained());

Assert(bufLen >= this->count);

wchar\_t \*pSrcBuf = this->firstChunk->u.single.buffer;

Data \*next = this->secondChunk;

charcount\_t srcLength = this->firstChunkLength;

for (Data \*chunk = this->firstChunk; chunk != this->lastChunk; next = chunk->u.chained.next)

{

if (bufLen < srcLength)

{

Throw::FatalInternalError();

}

js\_memcpy\_s(pBuf, bufLen \* sizeof(wchar\_t), pSrcBuf, sizeof(wchar\_t) \* srcLength);

bufLen -= srcLength;

pBuf += srcLength;

lastChunkCount -= srcLength;

chunk = next;

pSrcBuf = chunk->u.chained.buffer;

srcLength = chunk->u.chained.length;

}

if (bufLen < lastChunkCount)

{

Throw::FatalInternalError();

}

js\_memcpy\_s(pBuf, bufLen \* sizeof(wchar\_t), this->lastChunk->u.chained.buffer, sizeof(wchar\_t) \* lastChunkCount);

}

void UnChain()

{

Assert(this->IsChained());

charcount\_t newLength = this->count;

Data \*newChunk = this->NewSingleChunk(&newLength);

this->length = newLength;

this->UnChain(newChunk->u.single.buffer, newLength);

this->firstChunk = this->lastChunk = newChunk;

this->secondChunk = NULL;

this->appendPtr = newChunk->u.single.buffer + this->count;

}

void Copy(\_\_out \_\_ecount(bufLen) wchar\_t \*pBuf, charcount\_t bufLen)

{

if (this->IsChained())

{

this->UnChain(pBuf, bufLen);

}

else

{

if (bufLen < this->count)

{

Throw::FatalInternalError();

}

js\_memcpy\_s(pBuf, bufLen \* sizeof(wchar\_t), this->firstChunk->u.single.buffer, this->count \* sizeof(wchar\_t));

}

}

inline wchar\_t\* Buffer()

{

if (this->IsChained())

{

this->UnChain();

}

if (this->firstChunk)

{

this->firstChunk->u.single.buffer[this->count] = L'\0';

return this->firstChunk->u.single.buffer;

}

else

{

return L"";

}

}

inline charcount\_t Count() { return this->count; }

void Append(wchar\_t c)

{

if (this->count == this->length)

{

ExtendBuffer(this->length+1);

}

\*(this->appendPtr++) = c;

this->count++;

}

void AppendSz(const wchar\_t \* str)

{

// WARNING!!

// Do not use this to append JavascriptStrings. They can have embedded

// nulls which obviously won't be handled correctly here. Instead use

// Append with a length, which will use memcpy and correctly include any

// embedded null characters.

// WARNING!!

while (\*str != L'\0')

{

Append(\*str++);

}

}

void Append(const wchar\_t \* str, charcount\_t countNeeded)

{

EnsureBuffer(countNeeded);

wchar\_t \*dst = this->appendPtr;

JavascriptString::CopyHelper(dst, str, countNeeded);

this->appendPtr += countNeeded;

this->count += countNeeded;

}

template <size\_t N>

void AppendCppLiteral(const wchar\_t(&str)[N])

{

// Need to account for the terminating null character in C++ string literals, hence N > 2 and N - 1 below

static\_assert(N > 2, "Use Append(wchar\_t) for appending literal single characters and do not append empty string literal");

Append(str, N - 1);

}

// If we expect str to be large - we should just use this version that uses memcpy directly instead of Append

void AppendLarge(const wchar\_t \* str, charcount\_t countNeeded)

{

EnsureBuffer(countNeeded);

wchar\_t \*dst = this->appendPtr;

js\_memcpy\_s(dst, sizeof(WCHAR) \* countNeeded, str, sizeof(WCHAR) \* countNeeded);

this->appendPtr += countNeeded;

this->count += countNeeded;

}

errno\_t AppendUint64(unsigned \_\_int64 value)

{

const int max\_length = 20; // maximum length of 64-bit value converted to base 10 string

const int radix = 10;

WCHAR buf[max\_length+1];

errno\_t result = \_ui64tow\_s(value, buf, max\_length+1, radix);

AssertMsg(result==0, "Failed to translate value to string");

if (result == 0)

{

AppendSz(buf);

}

return result;

}

wchar\_t \*AllocBufferSpace(charcount\_t countNeeded)

{

EnsureBuffer(countNeeded);

return this->appendPtr;

}

void IncreaseCount(charcount\_t countInc)

{

if(countInc == 0) return;

this->count += countInc;

this->appendPtr += countInc;

Assert(this->count < this->length);

}

wchar\_t\* Detach()

{

// NULL terminate the string

Append(L'\0');

// if there is a chain we need to account for that also, so that the new buffer will have the NULL at the end.

if (this->IsChained())

{

this->UnChain();

}

// Now decrement the count to adjust according to number of chars

this->count--;

wchar\_t\* result = this->firstChunk->u.single.buffer;

this->firstChunk = this->lastChunk = NULL;

return result;

}

void TrimTrailingNULL()

{

Assert(this->count);

if (this->IsChained())

{

Assert(this->lastChunk->u.chained.buffer[this->count - (this->length - this->lastChunk->u.chained.length) - 1] == L'\0');

}

else

{

Assert(this->lastChunk->u.single.buffer[this->count - 1] == L'\0');

}

this->appendPtr--;

this->count--;

}

void Reset()

{

this->length = 0;

this->count = 0;

this->firstChunk = NULL;

this->secondChunk = NULL;

this->lastChunk = NULL;

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

///

/// class TreeNode

///

/// General DataStructure for an N-ary tree.

///

///----------------------------------------------------------------------------

///----------------------------------------------------------------------------

template<class T, int N>

class TreeNode

{

// Data

private:

T value;

TreeNode \* children[N];

TreeNode<T, N> \* parent;

// Constructor

public:

TreeNode(TreeNode<T, N> \* parent = NULL)

{

this->parent = parent;

for(int i = 0; i < N; i++)

{

this->children[i] = NULL;

}

}

// Methods

public:

bool ChildExistsAt(int i)

{

return NULL != this->children[i];

}

TreeNode<T, N> \* GetChildAt(int i)

{

return this->children[i];

}

void SetChildAt(int i, TreeNode<T, N> \*node)

{

this->children[i] = node;

}

TreeNode<T, N> \* GetParent()

{

return this->parent;

}

void SetParent(TreeNode<T, N>\* parent)

{

this->parent = parent;

}

T \* GetValue()

{

return &this->value;

}

void SetValue(const T value)

{

this->value = value;

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace regex

{

struct Nothing { };

template<typename T0, typename T1, typename T2 = Nothing, typename T3 = Nothing>

class Tuple

{

T0 first;

T1 second;

T2 third;

T3 forth;

public:

Tuple(T0 first, T1 second)

: first(first), second(second)

{

CompileAssert(sizeof(T2)==sizeof(Nothing));

CompileAssert(sizeof(T3)==sizeof(Nothing));

}

Tuple(T0 first, T1 second, T2 third)

: first(first), second(second), third(third)

{

CompileAssert(sizeof(T3)==sizeof(Nothing));

}

T0 First() const

{

return first;

}

T1 Second() const

{

return second;

}

T2 Third() const

{

CompileAssert(sizeof(T2)!=sizeof(Nothing));

return third;

}

T3 Forth() const

{

CompileAssert(sizeof(T3)!=sizeof(Nothing));

return forth;

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#define FOREACH\_BITSET\_IN\_UNITBV(index, bv, BVUnitT) \

{ \

BVIndex index; \

BVUnitT \_unit = bv; \

for(index = \_unit.GetNextBit(); index != -1; index = \_unit.GetNextBit()) \

{ \

\_unit.Clear(index); \

\

#define NEXT\_BITSET\_IN\_UNITBV }}

// Typedef

typedef uint32 UnitWord32;

typedef uint64 UnitWord64;

inline BOOLEAN

GetFirstBitSet(DWORD \*Index, UnitWord32 Mask)

{

return \_BitScanForward(Index, Mask);

}

inline BOOLEAN

GetFirstBitSet(DWORD \*Index, UnitWord64 Mask)

{

#if defined(\_M\_X64\_OR\_ARM64)

return \_BitScanForward64(Index, Mask);

#else

//\_BitScanForward64 intrinsic is not available in x86 & ARM

if (\_BitScanForward(Index, (UnitWord32)Mask))

{

return true;

}

if(\_BitScanForward(Index, (UnitWord32) (Mask >> 32)))

{

\*Index = \*Index + 32;

return true;

}

return false;

#endif

}

inline BOOLEAN

GetLastBitSet(DWORD \*Index, UnitWord32 Mask)

{

return \_BitScanReverse(Index, Mask);

}

inline BOOLEAN

GetLastBitSet(DWORD \*Index, UnitWord64 Mask)

{

#if defined(\_M\_X64\_OR\_ARM64)

return \_BitScanReverse64(Index, Mask);

#else

//\_BitScanReverse64 intrinsic is not available in x86 & ARM

if (\_BitScanReverse(Index, (UnitWord32)(Mask >> 32)))

{

\*Index = \*Index + 32;

return true;

}

return \_BitScanReverse(Index, (UnitWord32)Mask);

#endif

}

template <typename T>

class BVUnitT

{

// Data

private:

T word;

// Constructor

public:

BVUnitT(T initial = 0)

{

word = initial;

}

typedef T BVUnitTContainer;

// Implementation

private:

static void AssertRange(BVIndex index)

{

AssertMsg(index < BitsPerWord, "index out of bound");

}

static UnitWord32 Reverse(UnitWord32 bitsToReverse) {

bitsToReverse = (bitsToReverse & 0x55555555) << 1 | (bitsToReverse & 0xAAAAAAAA) >> 1;

bitsToReverse = (bitsToReverse & 0x33333333) << 2 | (bitsToReverse & 0xCCCCCCCC) >> 2;

bitsToReverse = (bitsToReverse & 0x0F0F0F0F) << 4 | (bitsToReverse & 0xF0F0F0F0) >> 4;

bitsToReverse = (bitsToReverse & 0x00FF00FF) << 8 | (bitsToReverse & 0xFF00FF00) >> 8;

bitsToReverse = (bitsToReverse & 0x0000FFFF) << 16 | (bitsToReverse & 0xFFFF0000) >> 16;

return bitsToReverse;

}

static UnitWord64 Reverse(UnitWord64 bits)

{

UnitWord32 lower = (UnitWord32) bits;

UnitWord32 upper = (UnitWord32) (bits >> 32);

UnitWord64 result = ((UnitWord64) Reverse(lower)) << 32;

result |= Reverse(upper);

return result;

}

static BVIndex CountBit(UnitWord32 bits)

{

const uint \_5\_32 = 0x55555555;

const uint \_3\_32 = 0x33333333;

const uint \_F1\_32 = 0x0f0f0f0f;

// In-place adder tree: perform 16 1-bit adds, 8 2-bit adds, 4 4-bit adds,

// 2 8=bit adds, and 1 16-bit add.

// From Dr. Dobb's Nov. 2000 letters, from Phil Bagwell, on reducing

// the cost by removing some of the masks that can be "forgotten" dbitse

// to the max # of bits set (32) that will fit in a byte.

//

bits -= (bits >> 1) & \_5\_32;

bits = ((bits >> 2) & \_3\_32) + (bits & \_3\_32);

bits = ((bits >> 4) & \_F1\_32) + (bits & \_F1\_32);

bits += bits >> 8;

bits += bits >> 16;

return BVIndex(bits & 0xff);

}

static BVIndex CountBit(UnitWord64 bits)

{

#if DBG

unsigned countBits = CountBit((UnitWord32)bits) + CountBit((UnitWord32)(bits >> 32));

#endif

const uint64 \_5\_64 = 0x5555555555555555ui64;

const uint64 \_3\_64 = 0x3333333333333333ui64;

const uint64 \_F1\_64 = 0x0f0f0f0f0f0f0f0fui64;

// In-place adder tree: perform 32 1-bit adds, 16 2-bit adds, 8 4-bit adds,

// 4 8-bit adds, 2 16-bit adds, and 1 32-bit add.

// From Dr. Dobb's Nov. 2000 letters, from Phil Bagwell, on reducing

// the cost by removing some of the masks that can be "forgotten" due

// to the max # of bits set (64) that will fit in a byte.

//

bits -= (bits >> 1) & \_5\_64;

bits = ((bits >> 2) & \_3\_64) + (bits & \_3\_64);

bits = ((bits >> 4) & \_F1\_64) + (bits & \_F1\_64);

bits += bits >> 8;

bits += bits >> 16;

bits += bits >> 32;

AssertMsg(countBits == (bits & 0xff), "Wrong count?");

return (BVIndex)(bits & 0xff);

}

static unsigned int NumLeadingZeroes(UnitWord32 bits)

{

int n = 0;

if (bits == 0) return 32;

// Binary search to figure out the number of leading zeroes

if (bits <= 0x0000FFFF)

{

// At least 16 leading zeroes- so remove them, and

// let's figure out how many leading zeroes in the last 16 bits

n = n + 16;

bits = bits << 16;

}

if (bits <= 0x00FFFFFF)

{

// At least 8 more zeroes- remove them, and repeat process

n = n + 8;

bits = bits << 8;

}

if (bits <= 0x0FFFFFFF)

{

n = n + 4;

bits = bits << 4;

}

if (bits <= 0x3FFFFFFF)

{

n = n + 2;

bits = bits << 2;

}

if (bits <= 0x7FFFFFFF)

{

n = n + 1;

}

return n;

}

static unsigned int NumLeadingZeroes(UnitWord64 bits)

{

UnitWord32 lower = (UnitWord32) bits;

UnitWord32 upper = (UnitWord32) (bits >> 32);

if (upper == 0)

{

return 32 + NumLeadingZeroes(lower);

}

else

{

return NumLeadingZeroes(upper);

}

}

public:

enum

{

BitsPerWord = sizeof(T) \* MachBits,

BitMask = BitsPerWord - 1,

AllOnesMask = -1

};

//ShiftValue is essentially log(sizeof(T))

//Initialization is through template specialization

static const LONG ShiftValue;

static BVIndex Position(BVIndex index)

{

return index >> ShiftValue;

}

static BVIndex Offset(BVIndex index)

{

return index & BitMask;

}

static BVIndex Floor(BVIndex index)

{

return index & (~BitMask);

}

static T GetTopBitsClear(BVIndex len)

{

return ((T)1 << Offset(len)) - 1;

}

bool Equal(BVUnitT unit) const

{

return this->word == unit.word;

}

void Set(BVIndex index)

{

AssertRange(index);

this->word |= (T)1 << index;

}

void Clear(BVIndex index)

{

AssertRange(index);

this->word &= ~((T)1 << index);

}

void Complement(BVIndex index)

{

AssertRange(index);

this->word ^= (T)1 << index;

}

BOOLEAN Test(const BVIndex index) const

{

AssertRange(index);

return (this->word & ( (T)1 << index)) != 0;

}

BOOLEAN Test(BVUnitT const unit) const

{

return (this->word & unit.word) != 0;

}

BOOLEAN TestRange(const BVIndex index, uint length) const

{

T mask = ((T)AllOnesMask) >> (BitsPerWord - length) << index;

return (this->word & mask) == mask;

}

void SetRange(const BVIndex index, uint length)

{

T mask = ((T)AllOnesMask) >> (BitsPerWord - length) << index;

this->word |= mask;

}

void ClearRange(const BVIndex index, uint length)

{

T mask = ((T)AllOnesMask) >> (BitsPerWord - length) << index;

this->word &= ~mask;

}

BVIndex GetNextBit() const

{

DWORD index;

if(GetFirstBitSet(&index, this->word))

{

return index;

}

else

{

return BVInvalidIndex;

}

}

BVIndex GetNextBit(BVIndex index) const

{

AssertRange(index);

BVUnitT temp = \*this;

temp.ClearAllTill(index);

return temp.GetNextBit();

}

BVIndex GetPrevBit() const

{

DWORD index;

if(GetLastBitSet(&index, this->word))

{

return index;

}

else

{

return BVInvalidIndex;

}

}

inline unsigned int GetNumberOfLeadingZeroes()

{

return NumLeadingZeroes(this->word);

}

T GetWord() const

{

return this->word;

}

inline BVIndex FirstStringOfOnes(unsigned int l)

{

unsigned int leadingZeroes;

BVIndex i = 0;

if (this->word == 0)

{

return BVInvalidIndex;

}

T bitVector = Reverse(this->word);

while (bitVector != 0) {

// Find the number of leading zeroes

leadingZeroes = NumLeadingZeroes(bitVector);

// Skip over leading zeroes

bitVector = bitVector << leadingZeroes;

i = i + leadingZeroes;

// Invert the bit vector- leading number of leading zeroes = number of leading 1's in the original bit vector

leadingZeroes = NumLeadingZeroes(~bitVector); // Count first/next group of 1's.

if (leadingZeroes >= l)

return i;

// If there aren't enough ones, we skip over them, iterate again

bitVector = bitVector << leadingZeroes;

i = i + leadingZeroes;

}

// No sub-sequence of 1's of length l found in this bit vector

return BVInvalidIndex;

}

void ClearAllTill(BVIndex index)

{

AssertRange(index);

this->word &= ((T)BVUnitT::AllOnesMask) <<(T)index;

}

BVIndex Count() const

{

return CountBit(this->word);

}

bool IsEmpty() const

{

return 0 == this->word;

}

bool IsFull() const

{

return this->word == AllOnesMask;

}

void SetAll()

{

this->word = (T)AllOnesMask;

}

void ClearAll()

{

this->word = 0;

}

void ComplimentAll()

{

this->word = ~this->word;

}

void Or(BVUnitT x)

{

this->word |= x.word;

}

void OrComplimented(BVUnitT x)

{

this->word |= (~x.word);

}

void And(BVUnitT x)

{

this->word &= x.word;

}

void Xor(BVUnitT x)

{

this->word ^= x.word;

}

uint DiffCount(BVUnitT x) const

{

return CountBit(x.word ^ this->word);

}

void Minus(BVUnitT x)

{

this->word &= (~x.word);

}

void Copy(BVUnitT x)

{

this->word = x.word;

}

void ShiftLeft(BVIndex count)

{

this->word <<= count;

}

void ShiftRight(BVIndex count)

{

this->word >>= count;

}

#if DBG\_DUMP || defined(ENABLE\_IR\_VIEWER)

void DumpWord()

{

Output::Print(L"%p", this->word);

}

bool Dump(BVIndex base = 0, bool hasBits = false) const

{

FOREACH\_BITSET\_IN\_UNITBV(index, \*this, BVUnitT)

{

if (hasBits)

{

Output::Print(L", ");

}

Output::Print(L"%u", index + base);

hasBits = true;

}

NEXT\_BITSET\_IN\_UNITBV;

return hasBits;

}

#endif

};

typedef BVUnitT<UnitWord32> BVUnit32;

typedef BVUnitT<UnitWord64> BVUnit64;

const LONG BVUnitT<UnitWord32>::ShiftValue = 5;

const LONG BVUnitT<UnitWord64>::ShiftValue = 6;

#if defined(\_M\_X64\_OR\_ARM64)

typedef BVUnit64 BVUnit;

#else

typedef BVUnit32 BVUnit;

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace JsUtil

{

interface IWeakReferenceDictionary

{

virtual void Cleanup() = 0;

};

template <

class TKey,

class TValue,

class SizePolicy = PowerOf2SizePolicy,

template <typename ValueOrKey> class Comparer = DefaultComparer

>

class WeakReferenceDictionary: public BaseDictionary<TKey, RecyclerWeakReference<TValue>\*, RecyclerNonLeafAllocator, SizePolicy, Comparer, WeakRefValueDictionaryEntry>,

public IWeakReferenceDictionary

{

public:

WeakReferenceDictionary(Recycler\* recycler, int capacity = 0):

BaseDictionary(recycler, capacity)

{

Assert(reinterpret\_cast<void\*>(this) == reinterpret\_cast<void\*>((IWeakReferenceDictionary\*) this));

}

virtual void Cleanup() override

{

this->MapAndRemoveIf([](EntryType& entry)

{

return (EntryType::NeedsCleanup(entry));

});

}

private:

using BaseDictionary::Clone;

using BaseDictionary::Copy;

PREVENT\_COPY(WeakReferenceDictionary);

};

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

class AsmJsParseException : public ExceptionBase

{

};

} // namespace Js

<?xml version="1.0" encoding="utf-8"?>

<Project DefaultTargets="Build" ToolsVersion="12.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Import Condition="'$(ChakraBuildPathImported)'!='true'" Project="$(SolutionDir)Chakra.Build.Paths.props" />

<Import Project="$(BuildConfigPropsPath)Chakra.Build.ProjectConfiguration.props" />

<PropertyGroup Label="Globals">

<TargetName>Chakra.Common.Exceptions</TargetName>

<ProjectGuid>{FD8EEC40-4141-448A-BF4B-1589FBE4F60D}</ProjectGuid>

<RootNamespace>JS</RootNamespace>

<Keyword>Win32Proj</Keyword>

</PropertyGroup>

<PropertyGroup Label="Configuration">

<ConfigurationType>StaticLibrary</ConfigurationType>

</PropertyGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />

<Import Project="$(BuildConfigPropsPath)Chakra.Build.props" />

<PropertyGroup>

<\_ProjectFileVersion>10.0.30319.1</\_ProjectFileVersion>

</PropertyGroup>

<ItemDefinitionGroup>

<ClCompile>

<AdditionalIncludeDirectories>

$(MSBuildThisFileDirectory)..;

%(AdditionalIncludeDirectories)

</AdditionalIncludeDirectories>

<PrecompiledHeader>Use</PrecompiledHeader>

<PrecompiledHeaderFile>CommonExceptionsPch.h</PrecompiledHeaderFile>

</ClCompile>

</ItemDefinitionGroup>

<ItemGroup>

<ClCompile Include="$(MSBuildThisFileDirectory)ExceptionCheck.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)reporterror.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)ExceptionBase.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CommonExceptionsPch.cpp">

<PrecompiledHeader>Create</PrecompiledHeader>

</ClCompile>

<ClCompile Include="$(MSBuildThisFileDirectory)Throw.cpp">

<AdditionalIncludeDirectories>..\..\lib\runtime;%(AdditionalIncludeDirectories)</AdditionalIncludeDirectories>

</ClCompile>

</ItemGroup>

<ItemGroup>

<ClInclude Include="AsmJsParseException.h" />

<ClInclude Include="CommonExceptionsPch.h" />

<ClInclude Include="EvalDisabledException.h" />

<ClInclude Include="ExceptionBase.h" />

<ClInclude Include="ExceptionCheck.h" />

<ClInclude Include="InternalErrorException.h" />

<ClInclude Include="NotImplementedException.h" />

<ClInclude Include="OperationAbortedException.h" />

<ClInclude Include="OutOfMemoryException.h" />

<ClInclude Include="RejitException.h" />

<ClInclude Include="reporterror.h" />

<ClInclude Include="ScriptAbortException.h" />

<ClInclude Include="StackOverflowException.h" />

<ClInclude Include="Throw.h" />

</ItemGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.targets" Condition="exists('$(BuildConfigPropsPath)Chakra.Build.targets')" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />

</Project>

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonExceptionsPch.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonBasic.h"

#include "Exceptions\ExceptionBase.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

class EvalDisabledException : public ExceptionBase

{

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonExceptionsPch.h"

namespace Js

{

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

class ExceptionBase

{

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonExceptionsPch.h"

#ifdef EXCEPTION\_CHECK

#include "ExceptionCheck.h"

\_\_declspec(thread) ExceptionCheck::Data ExceptionCheck::data;

BOOL ExceptionCheck::IsEmpty()

{

return (data.handledExceptionType == ExceptionType\_None);

}

ExceptionCheck::Data ExceptionCheck::Save()

{

ExceptionCheck::Data savedData = data;

data = ExceptionCheck::Data();

return savedData;

}

void ExceptionCheck::Restore(ExceptionCheck::Data& savedData)

{

Assert(IsEmpty());

data = savedData;

}

ExceptionCheck::Data ExceptionCheck::GetData()

{

return data;

}

BOOL ExceptionCheck::CanHandleOutOfMemory()

{

return (data.handledExceptionType == ExceptionType\_DisableCheck) ||

JsUtil::ExternalApi::IsScriptActiveOnCurrentThreadContext() ||

(data.handledExceptionType & ExceptionType\_OutOfMemory);

}

BOOL ExceptionCheck::HasStackProbe()

{

return (data.handledExceptionType & ExceptionType\_HasStackProbe);

}

BOOL ExceptionCheck::CanHandleStackOverflow(bool isExternal)

{

return (JsUtil::ExternalApi::IsScriptActiveOnCurrentThreadContext() || isExternal) ||

(data.handledExceptionType & ExceptionType\_StackOverflow) ||

(data.handledExceptionType == ExceptionType\_DisableCheck);

}

void ExceptionCheck::SetHandledExceptionType(ExceptionType e)

{

Assert((e & ExceptionType\_DisableCheck) == 0 || e == ExceptionType\_DisableCheck);

Assert(IsEmpty());

#if DBG

if(!(e == ExceptionType\_None ||

e == ExceptionType\_DisableCheck ||

!JsUtil::ExternalApi::IsScriptActiveOnCurrentThreadContext() ||

(e & ExceptionType\_JavascriptException) == ExceptionType\_JavascriptException ||

e == ExceptionType\_HasStackProbe))

{

Assert(false);

}

#endif

data.handledExceptionType = e;

}

ExceptionType ExceptionCheck::ClearHandledExceptionType()

{

ExceptionType exceptionType = data.handledExceptionType;

data.handledExceptionType = ExceptionType\_None;

Assert(IsEmpty());

return exceptionType;

}

AutoHandledExceptionType::AutoHandledExceptionType(ExceptionType e)

{

ExceptionCheck::SetHandledExceptionType(e);

}

AutoHandledExceptionType::~AutoHandledExceptionType()

{

Assert(ExceptionCheck::GetData().handledExceptionType == ExceptionType\_DisableCheck ||

!JsUtil::ExternalApi::IsScriptActiveOnCurrentThreadContext() ||

ExceptionCheck::GetData().handledExceptionType == ExceptionType\_HasStackProbe ||

(ExceptionCheck::GetData().handledExceptionType & ExceptionType\_JavascriptException) == ExceptionType\_JavascriptException);

ExceptionCheck::ClearHandledExceptionType();

}

AutoNestedHandledExceptionType::AutoNestedHandledExceptionType(ExceptionType e)

{

savedData = ExceptionCheck::Save();

ExceptionCheck::SetHandledExceptionType(e);

}

AutoNestedHandledExceptionType::~AutoNestedHandledExceptionType()

{

Assert(ExceptionCheck::GetData().handledExceptionType == ExceptionType\_DisableCheck ||

!JsUtil::ExternalApi::IsScriptActiveOnCurrentThreadContext() ||

ExceptionCheck::GetData().handledExceptionType == ExceptionType\_HasStackProbe ||

(ExceptionCheck::GetData().handledExceptionType & ExceptionType\_JavascriptException) == ExceptionType\_JavascriptException);

ExceptionCheck::ClearHandledExceptionType();

ExceptionCheck::Restore(savedData);

}

AutoFilterExceptionRegion::AutoFilterExceptionRegion(ExceptionType e)

{

savedData = ExceptionCheck::Save();

ExceptionCheck::SetHandledExceptionType((ExceptionType)(~e & savedData.handledExceptionType));

}

AutoFilterExceptionRegion::~AutoFilterExceptionRegion()

{

ExceptionCheck::ClearHandledExceptionType();

ExceptionCheck::Restore(savedData);

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef EXCEPTION\_CHECK

enum ExceptionType

{

ExceptionType\_None = 0x00000000,

ExceptionType\_OutOfMemory = 0x00000001,

ExceptionType\_StackOverflow = 0x00000002,

ExceptionType\_JavascriptException = 0x00000004,

ExceptionType\_HasStackProbe = 0x00000008,

ExceptionType\_DisableCheck = 0x80000000,

ExceptionType\_All = 0x0FFFFFFF,

};

class ExceptionCheck

{

public:

static void SetHandledExceptionType(ExceptionType e);

static ExceptionType ClearHandledExceptionType();

static BOOL CanHandleOutOfMemory();

static BOOL CanHandleStackOverflow(bool isExternal);

static BOOL HasStackProbe();

struct Data

{

ExceptionType handledExceptionType;

};

static ExceptionCheck::Data Save();

static void Restore(ExceptionCheck::Data& savedData);

static ExceptionCheck::Data GetData();

private:

static BOOL IsEmpty();

\_\_declspec(thread) static Data data;

};

class AutoHandledExceptionType

{

public:

AutoHandledExceptionType(ExceptionType e);

~AutoHandledExceptionType();

};

class AutoNestedHandledExceptionType

{

public:

AutoNestedHandledExceptionType(ExceptionType e);

~AutoNestedHandledExceptionType();

private:

ExceptionCheck::Data savedData;

};

class AutoFilterExceptionRegion

{

public:

AutoFilterExceptionRegion(ExceptionType e);

~AutoFilterExceptionRegion();

private:

ExceptionCheck::Data savedData;

};

#define AssertCanHandleOutOfMemory() Assert(ExceptionCheck::CanHandleOutOfMemory())

#define AssertCanHandleStackOverflow() Assert(ExceptionCheck::CanHandleStackOverflow(false))

#define AssertCanHandleStackOverflowCall(isExternal) Assert(ExceptionCheck::CanHandleStackOverflow(isExternal))

#define DECLARE\_EXCEPTION\_CHECK\_DATA \

ExceptionCheck::Data \_\_exceptionCheck;

#define SAVE\_EXCEPTION\_CHECK \

\_\_exceptionCheck = ExceptionCheck::Save();

#define RESTORE\_EXCEPTION\_CHECK \

ExceptionCheck::Restore(\_\_exceptionCheck);

#define AUTO\_HANDLED\_EXCEPTION\_TYPE(type) AutoHandledExceptionType \_\_autoHandledExceptionType(type)

#define AUTO\_NESTED\_HANDLED\_EXCEPTION\_TYPE(type) AutoNestedHandledExceptionType \_\_autoNestedHandledExceptionType(type)

#define AUTO\_FILTER\_EXCEPTION\_REGION(type) AutoFilterExceptionRegion \_\_autoFilterExceptionRegion(type)

#define AUTO\_NO\_EXCEPTION\_REGION AUTO\_FILTER\_EXCEPTION\_REGION(ExceptionType\_All)

#else

#define AssertCanHandleOutOfMemory()

#define AssertCanHandleStackOverflow()

#define AssertCanHandleStackOverflowCall(isExternal)

#define DECLARE\_EXCEPTION\_CHECK\_DATA

#define SAVE\_EXCEPTION\_CHECK

#define RESTORE\_EXCEPTION\_CHECK

#define AUTO\_HANDLED\_EXCEPTION\_TYPE(type)

#define AUTO\_NESTED\_HANDLED\_EXCEPTION\_TYPE(type)

#define AUTO\_FILTER\_EXCEPTION\_REGION(type)

#define AUTO\_NO\_EXCEPTION\_REGION

#endif

#if DBG

class DebugCheckNoException

{

public:

DebugCheckNoException() : hasException(true) { SAVE\_EXCEPTION\_CHECK;}

~DebugCheckNoException() { Assert(!hasException); RESTORE\_EXCEPTION\_CHECK; }

DECLARE\_EXCEPTION\_CHECK\_DATA;

bool hasException;

};

#define BEGIN\_NO\_EXCEPTION { DebugCheckNoException \_\_debugCheckNoException;

#define END\_NO\_EXCEPTION \_\_debugCheckNoException.hasException = false; }

#else

#define BEGIN\_NO\_EXCEPTION

#define END\_NO\_EXCEPTION

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

class InternalErrorException: public ExceptionBase

{

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

class NotImplementedException : public ExceptionBase

{

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

class OperationAbortedException : public ExceptionBase

{

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

class OutOfMemoryException : public ExceptionBase

{

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

class RejitException : public ExceptionBase

{

private:

const RejitReason reason;

public:

RejitException(const RejitReason reason) : reason(reason)

{

}

public:

RejitReason Reason() const

{

return reason;

}

const char \*ReasonName() const

{

return RejitReasonNames[reason];

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonExceptionsPch.h"

\_\_inline void ReportFatalException(

\_\_in ULONG\_PTR context,

\_\_in HRESULT exceptionCode,

\_\_in ErrorReason reasonCode,

\_\_in ULONG scenario)

{

// avoid the error text methods to be optimized out.

UNREFERENCED\_PARAMETER(scenario);

if (IsDebuggerPresent())

{

DebugBreak();

}

\_\_try

{

ULONG\_PTR ExceptionInformation[2];

ExceptionInformation[0] = (ULONG\_PTR)reasonCode;

ExceptionInformation[1] = (ULONG\_PTR)context;

RaiseException(exceptionCode, EXCEPTION\_NONCONTINUABLE, 2, (ULONG\_PTR\*)ExceptionInformation);

}

\_\_except(FatalExceptionFilter(GetExceptionInformation()))

{

}

}

// Disable optimization make sure all the frames are still available in Dr. Watson bug reports.

#pragma optimize("", off)

\_\_declspec(noinline) void JavascriptDispatch\_OOM\_fatal\_error(

\_\_in ULONG\_PTR context)

{

int scenario = 1;

ReportFatalException(context, E\_OUTOFMEMORY, JavascriptDispatch\_OUTOFMEMORY, scenario);

};

\_\_declspec(noinline) void CustomHeap\_BadPageState\_fatal\_error(

\_\_in ULONG\_PTR context)

{

int scenario = 1;

ReportFatalException(context, E\_UNEXPECTED, CustomHeap\_MEMORYCORRUPTION, scenario);

};

\_\_declspec(noinline) void MarkStack\_OOM\_fatal\_error()

{

int scenario = 1;

ReportFatalException(NULL, E\_OUTOFMEMORY, MarkStack\_OUTOFMEMORY, scenario);

};

\_\_declspec(noinline) void Amd64StackWalkerOutOfContexts\_fatal\_error(

\_\_in ULONG\_PTR context)

{

int scenario = 1;

ReportFatalException(context, E\_UNEXPECTED, Fatal\_Amd64StackWalkerOutOfContexts, scenario);

}

\_\_declspec(noinline) void FailedToBox\_OOM\_fatal\_error(

\_\_in ULONG\_PTR context)

{

int scenario = 1;

ReportFatalException(context, E\_UNEXPECTED, Fatal\_FailedToBox\_OUTOFMEMORY, scenario);

}

#if defined(RECYCLER\_WRITE\_BARRIER) && defined(\_M\_X64\_OR\_ARM64)

\_\_declspec(noinline) void X64WriteBarrier\_OOM\_fatal\_error()

{

int scenario = 3;

ReportFatalException(NULL, E\_OUTOFMEMORY, WriteBarrier\_OUTOFMEMORY, scenario);

}

#endif

\_\_declspec(noinline) void DebugHeap\_OOM\_fatal\_error()

{

int scenario = 3;

ReportFatalException(NULL, E\_OUTOFMEMORY, Fatal\_Debug\_Heap\_OUTOFMEMORY, scenario);

}

\_\_declspec(noinline) void Binary\_Inconsistency\_fatal\_error()

{

int scenario = 4;

ReportFatalException(NULL, E\_UNEXPECTED, Fatal\_Binary\_Inconsistency, scenario);

}

\_\_declspec(noinline) void Version\_Inconsistency\_fatal\_error()

{

int scenario = 4;

ReportFatalException(NULL, E\_UNEXPECTED, Fatal\_Version\_Inconsistency, scenario);

}

#ifdef LARGEHEAPBLOCK\_ENCODING

\_\_declspec(noinline) void LargeHeapBlock\_Metadata\_Corrupted(

\_\_in ULONG\_PTR context, \_\_in unsigned char calculatedChecksum)

{

int scenario = calculatedChecksum; /\* For debugging purpose if checksum mismatch happen\*/

ReportFatalException(context, E\_UNEXPECTED, LargeHeapBlock\_Metadata\_Corrupt, scenario);

};

#endif

\_\_declspec(noinline) void FromDOM\_NoScriptScope\_fatal\_error()

{

int scenario = 5;

ReportFatalException(NULL, E\_UNEXPECTED, EnterScript\_FromDOM\_NoScriptScope, scenario);

}

#pragma optimize("",on)

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

enum ErrorReason

{

JavascriptDispatch\_OUTOFMEMORY = 1,

Fatal\_Internal\_Error = 2,

Fatal\_Debug\_Heap\_OUTOFMEMORY = 3,

Fatal\_Amd64StackWalkerOutOfContexts = 4,

// Unused = 5,

Fatal\_Binary\_Inconsistency = 6,

WriteBarrier\_OUTOFMEMORY = 7,

CustomHeap\_MEMORYCORRUPTION = 8,

LargeHeapBlock\_Metadata\_Corrupt = 9,

Fatal\_Version\_Inconsistency = 10,

MarkStack\_OUTOFMEMORY = 11,

Fatal\_FailedToBox\_OUTOFMEMORY = 12,

EnterScript\_FromDOM\_NoScriptScope = 13

};

extern "C" void ReportFatalException(

\_\_in ULONG\_PTR context,

\_\_in HRESULT exceptionCode,

\_\_in ErrorReason reasonCode,

\_\_in ULONG scenario);

// We can have other error handle code path with

// unique call stack so we can collect data in Dr. Watson.

void JavascriptDispatch\_OOM\_fatal\_error(

\_\_in ULONG\_PTR context);

void CustomHeap\_BadPageState\_fatal\_error(

\_\_in ULONG\_PTR context);

void Amd64StackWalkerOutOfContexts\_fatal\_error(

\_\_in ULONG\_PTR context);

void FailedToBox\_OOM\_fatal\_error(

\_\_in ULONG\_PTR context);

#if defined(RECYCLER\_WRITE\_BARRIER) && defined(\_M\_X64\_OR\_ARM64)

void X64WriteBarrier\_OOM\_fatal\_error();

#endif

void DebugHeap\_OOM\_fatal\_error();

void MarkStack\_OOM\_fatal\_error();

void Binary\_Inconsistency\_fatal\_error();

void Version\_Inconsistency\_fatal\_error();

#ifdef LARGEHEAPBLOCK\_ENCODING

void LargeHeapBlock\_Metadata\_Corrupted(

\_\_in ULONG\_PTR context, \_\_in unsigned char calculatedCheckSum);

#endif

void FromDOM\_NoScriptScope\_fatal\_error();

// RtlReportException is available on Vista and up, but we cannot use it for OOB release.

// Use UnhandleExceptionFilter to let the default handler handles it.

\_\_inline LONG FatalExceptionFilter(

\_\_in LPEXCEPTION\_POINTERS lpep)

{

LONG rc = UnhandledExceptionFilter(lpep);

// re == EXCEPTION\_EXECUTE\_HANDLER means there is no debugger attached, let's terminate

// the process. Otherwise give control to the debugger.

// Note: in case when postmortem debugger is registered but no actual debugger attached,

// rc will be 0 (and EXCEPTION\_EXECUTE\_HANDLER is 1), so it acts as if there is debugger attached.

if (rc == EXCEPTION\_EXECUTE\_HANDLER)

{

TerminateProcess(GetCurrentProcess(), (UINT)DBG\_TERMINATE\_PROCESS);

}

else

{

Assert(IsDebuggerPresent());

DebugBreak();

}

return EXCEPTION\_CONTINUE\_SEARCH;

}

template<class Fn>

static STDMETHODIMP DebugApiWrapper(Fn fn)

{

// If an assertion or AV is hit, it triggers a SEH exception. SEH exceptions escaped here will be eaten by PDM. To prevent assertions

// from getting unnoticed, we install a SEH exception filter and crash the process.

#if ENABLE\_DEBUG\_API\_WRAPPER

\_\_try

{

#endif

return fn();

#if ENABLE\_DEBUG\_API\_WRAPPER

}

\_\_except(FatalExceptionFilter(GetExceptionInformation()))

{

}

return E\_FAIL;

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

class ScriptAbortException : public ExceptionBase

{

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js {

class StackOverflowException : public ExceptionBase

{

};

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonExceptionsPch.h"

// === C Runtime Header Files ===

#pragma warning(push)

#pragma warning(disable: 4995) /\* 'function': name was marked as #pragma deprecated \*/

#include <strsafe.h>

#pragma warning(pop)

#include "StackOverflowException.h"

#include "AsmJsParseException.h"

#include "InternalErrorException.h"

#include "OutOfMemoryException.h"

#include "NotImplementedException.h"

// Header files required before including ConfigFlagsTable.h

#include "EnumHelp.h"

#include "Common\MathUtil.h"

#include "Core\AllocSizeMath.h"

#include "Core\FaultInjection.h"

#include "core\BasePtr.h"

#include "core\AutoFILE.h"

#include "core\Output.h"

// Memory Management

namespace Memory {}

using namespace Memory;

#include "Memory\Allocator.h"

#include "Memory\HeapAllocator.h"

// Data structure

#include "DataStructures\Comparer.h"

#include "DataStructures\SizePolicy.h"

#include "DataStructures\SList.h"

#include "DataStructures\KeyValuePair.h"

#include "DataStructures\BaseDictionary.h"

#include "core\ConfigFlagsTable.h"

#include "core\StackBackTrace.h"

// dbghelp.h is not clean with warning 4091

#pragma warning(push)

#pragma warning(disable: 4091) /\* warning C4091: 'typedef ': ignored on left of '' when no variable is declared \*/

#include <dbghelp.h>

#pragma warning(pop)

extern "C"{

BOOLEAN IsMessageBoxWPresent();

}

namespace Js {

#ifdef GENERATE\_DUMP

StackBackTrace \* Throw::stackBackTrace = nullptr;

#endif

void Throw::FatalInternalError()

{

int scenario = 2;

ReportFatalException(NULL, E\_FAIL, Fatal\_Internal\_Error, scenario);

}

void Throw::FatalProjectionError()

{

RaiseException((DWORD)DBG\_TERMINATE\_PROCESS, EXCEPTION\_NONCONTINUABLE, 0, NULL);

}

void Throw::InternalError()

{

AssertMsg(false, "Internal error!!");

throw InternalErrorException();

}

void Throw::OutOfMemory()

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (CONFIG\_FLAG(PrintSystemException))

{

Output::Print(L"SystemException: OutOfMemory\n");

Output::Flush();

}

#endif

if (JsUtil::ExternalApi::RaiseOutOfMemoryIfScriptActive())

{

AssertMsg(false, "We shouldn't be here");

}

throw OutOfMemoryException();

}

void Throw::CheckAndThrowOutOfMemory(BOOLEAN status)

{

if (!status)

{

OutOfMemory();

}

}

void Throw::StackOverflow(ScriptContext \*scriptContext, PVOID returnAddress)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (CONFIG\_FLAG(PrintSystemException))

{

Output::Print(L"SystemException: StackOverflow\n");

Output::Flush();

}

#endif

if (JsUtil::ExternalApi::RaiseStackOverflowIfScriptActive(scriptContext, returnAddress))

{

AssertMsg(false, "We shouldn't be here");

}

throw StackOverflowException();

}

void Throw::NotImplemented()

{

AssertMsg(false, "This functionality is not yet implemented");

throw NotImplementedException();

}

// Returns true when the process is either TE.exe or TE.processhost.exe

bool Throw::IsTEProcess()

{

wchar\_t fileName[\_MAX\_PATH];

wchar\_t moduleName[\_MAX\_PATH];

GetModuleFileName(0, moduleName, \_MAX\_PATH);

errno\_t err = \_wsplitpath\_s(moduleName, nullptr, 0, nullptr, 0, fileName, \_MAX\_PATH, nullptr, 0);

return err == 0 && \_wcsnicmp(fileName, L"TE", 2) == 0;

}

void Throw::GenerateDumpAndTerminateProcess(PEXCEPTION\_POINTERS exceptInfo)

{

if (!Throw::IsTEProcess()

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

&& !Js::Configuration::Global.flags.IsEnabled(Js::DumpOnCrashFlag)

#endif

)

{

return;

}

#ifdef GENERATE\_DUMP

Js::Throw::GenerateDump(exceptInfo, Js::Configuration::Global.flags.DumpOnCrash);

#endif

// For now let's terminate the process.

TerminateProcess(GetCurrentProcess(), (UINT)DBG\_TERMINATE\_PROCESS);

}

#ifdef GENERATE\_DUMP

CriticalSection Throw::csGenereateDump;

void Throw::GenerateDump(LPCWSTR filePath, bool terminate, bool needLock)

{

\_\_try

{

if (terminate)

{

RaiseException((DWORD)DBG\_TERMINATE\_PROCESS, EXCEPTION\_NONCONTINUABLE, 0, NULL);

}

else

{

RaiseException(0, 0, 0, NULL);

}

}

\_\_except(Throw::GenerateDump(GetExceptionInformation(), filePath,

terminate? EXCEPTION\_CONTINUE\_SEARCH : EXCEPTION\_EXECUTE\_HANDLER), needLock)

{

// we don't do anything interesting in this handler

}

}

void Throw::GenerateDumpForAssert(LPCWSTR filePath)

{

\_\_try

{

RaiseException(STATUS\_ASSERTION\_FAILURE, EXCEPTION\_NONCONTINUABLE, 0, NULL);

}

\_\_except (Throw::GenerateDump(GetExceptionInformation(), filePath, EXCEPTION\_CONTINUE\_SEARCH), false)

{

// no-op

}

}

int Throw::GenerateDump(PEXCEPTION\_POINTERS exceptInfo, LPCWSTR filePath, int ret, bool needLock)

{

WCHAR tempFilePath[MAX\_PATH];

WCHAR tempFileName[MAX\_PATH];

HANDLE hTempFile;

DWORD retVal;

if (filePath == NULL)

{

retVal = GetTempPath(MAX\_PATH, tempFilePath);

if (retVal > MAX\_PATH || (retVal == 0))

{

return ret;

}

filePath = tempFilePath;

}

StringCchPrintf(tempFileName, \_countof(tempFileName), L"%s\\CH\_%u\_%u.dmp", filePath, GetCurrentProcessId(), GetCurrentThreadId());

Output::Print(L"dump filename %s \n", tempFileName);

Output::Flush();

hTempFile = CreateFile(tempFileName, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hTempFile == INVALID\_HANDLE\_VALUE)

{

return GetLastError();

}

MINIDUMP\_EXCEPTION\_INFORMATION dumpExceptInfo;

dumpExceptInfo.ThreadId = GetCurrentThreadId();

dumpExceptInfo.ExceptionPointers = exceptInfo;

dumpExceptInfo.ClientPointers = FALSE;

{

MINIDUMP\_TYPE dumpType = static\_cast<MINIDUMP\_TYPE>(MiniDumpWithDataSegs | MiniDumpWithPrivateReadWriteMemory);

// Generating full dump for the TE process (reason : it contains both managed and native memory)

if (CONFIG\_FLAG(FullMemoryDump))

{

dumpType = static\_cast<MINIDUMP\_TYPE>(dumpType | MiniDumpWithFullMemory);

}

BOOL dumpGenerated = false;

if (needLock)

{

// the critical section might have been destructed at process shutdown time. At that time we don't need

// to lock.

AutoCriticalSection autocs(&csGenereateDump);

dumpGenerated = MiniDumpWriteDump(GetCurrentProcess(),

GetCurrentProcessId(),

hTempFile,

dumpType,

&dumpExceptInfo,

NULL,

NULL);

}

else

{

dumpGenerated = MiniDumpWriteDump(GetCurrentProcess(),

GetCurrentProcessId(),

hTempFile,

dumpType,

&dumpExceptInfo,

NULL,

NULL);

}

if (!dumpGenerated)

{

Output::Print(L"Unable to write minidump (0x%08X)\n", GetLastError());

Output::Flush();

}

}

FlushFileBuffers(hTempFile);

CloseHandle(hTempFile);

return ret;

}

#endif // GENERATE\_DUMP

#if DBG

// After assert the program should terminate. Sometime we saw the program continue somehow

// log the existence of assert for debugging.

void Throw::LogAssert()

{

IsInAssert = true;

// This should be the last thing to happen in the process. Therefore, leaks are not an issue.

stackBackTrace = StackBackTrace::Capture(&NoCheckHeapAllocator::Instance, Throw::StackToSkip, Throw::StackTraceDepth);

}

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

static const wchar\_t \* caption = L"CHAKRA ASSERT";

#endif

bool Throw::ReportAssert(\_\_in LPSTR fileName, uint lineNumber, \_\_in LPSTR error, \_\_in LPSTR message)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (Js::Configuration::Global.flags.IsEnabled(Js::AssertBreakFlag))

{

DebugBreak();

return false;

}

if (Js::Configuration::Global.flags.IsEnabled(Js::AssertIgnoreFlag))

{

return true;

}

#endif

if (AssertsToConsole)

{

fprintf(stderr, "ASSERTION %u: (%s, line %u) %s\n Failure: %s\n", GetCurrentProcessId(), fileName, lineNumber, message, error);

fflush(stderr);

#ifdef GENERATE\_DUMP

// force dump if we have assert in jc.exe. check build only.

if (!Js::Configuration::Global.flags.IsEnabled(Js::DumpOnCrashFlag))

{

return false;

}

Throw::GenerateDumpForAssert(Js::Configuration::Global.flags.DumpOnCrash);

#else

return false;

#endif

}

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

// Then if DumpOncrashFlag is not specified it directly returns,

// otherwise if will raise a non-continuable exception, generate the dump and terminate the process.

// the popup message box might be useful when testing in IE

if (Js::Configuration::Global.flags.AssertPopUp && IsMessageBoxWPresent())

{

wchar\_t buff[1024];

swprintf\_s(buff, \_countof(buff), L"%S (%u)\n%S\n%S", fileName, lineNumber, message, error);

buff[\_countof(buff)-1] = 0;

int ret = MessageBox(nullptr, buff, caption, MB\_ABORTRETRYIGNORE);

switch (ret)

{

case IDIGNORE:

return true;

case IDABORT:

Throw::FatalInternalError();

default:

return false;

}

}

#endif

return false;

}

#endif

} // namespace Js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

class StackBackTrace;

namespace Js {

class ScriptContext;

class Throw

{

public:

static void \_\_declspec(noreturn) OutOfMemory();

static void \_\_declspec(noreturn) StackOverflow(ScriptContext \*scriptContext, PVOID returnAddress);

static void \_\_declspec(noreturn) NotImplemented();

static void \_\_declspec(noreturn) InternalError();

static void \_\_declspec(noreturn) FatalInternalError();

static void \_\_declspec(noreturn) FatalProjectionError();

static bool IsTEProcess();

static void GenerateDumpAndTerminateProcess(PEXCEPTION\_POINTERS exceptInfo);

static void CheckAndThrowOutOfMemory(BOOLEAN status);

#ifdef GENERATE\_DUMP

static bool ReportAssert(\_\_in LPSTR fileName, uint lineNumber, \_\_in LPSTR error, \_\_in LPSTR message);

static void LogAssert();

static int GenerateDump(PEXCEPTION\_POINTERS exceptInfo, LPCWSTR filePath, int ret = EXCEPTION\_CONTINUE\_SEARCH, bool needLock = false);

static void GenerateDump(LPCWSTR filePath, bool terminate = false, bool needLock = false);

static void GenerateDumpForAssert(LPCWSTR filePath);

private:

static CriticalSection csGenereateDump;

\_\_declspec(thread) static StackBackTrace \* stackBackTrace;

static const int StackToSkip = 2;

static const int StackTraceDepth = 40;

#endif

};

// Info: Verify the result or throw catastrophic

// Parameters: HRESULT

inline void VerifyOkCatastrophic(\_\_in HRESULT hr)

{

if (hr == E\_OUTOFMEMORY)

{

Js::Throw::OutOfMemory();

}

else if (FAILED(hr))

{

Js::Throw::FatalProjectionError();

}

}

// Info: Verify the result or throw catastrophic

// Parameters: bool

template<typename TCheck>

inline void VerifyCatastrophic(\_\_in TCheck result)

{

if (!result)

{

Assert(false);

Js::Throw::FatalProjectionError();

}

}

} // namespace Js

#define BEGIN\_TRANSLATE\_TO\_HRESULT(type) \

{\

try \

{ \

AUTO\_HANDLED\_EXCEPTION\_TYPE(type);

#define BEGIN\_TRANSLATE\_TO\_HRESULT\_NESTED(type) \

{\

try \

{ \

AUTO\_NESTED\_HANDLED\_EXCEPTION\_TYPE(type);

#define BEGIN\_TRANSLATE\_OOM\_TO\_HRESULT BEGIN\_TRANSLATE\_TO\_HRESULT(ExceptionType\_OutOfMemory)

#define BEGIN\_TRANSLATE\_OOM\_TO\_HRESULT\_NESTED BEGIN\_TRANSLATE\_TO\_HRESULT\_NESTED(ExceptionType\_OutOfMemory)

#define END\_TRANSLATE\_OOM\_TO\_HRESULT(hr) \

} \

catch (Js::OutOfMemoryException) \

{ \

hr = E\_OUTOFMEMORY; \

}\

}

#define END\_TRANSLATE\_OOM\_TO\_HRESULT\_AND\_EXCEPTION\_OBJECT(hr, scriptContext, exceptionObject) \

} \

catch(Js::OutOfMemoryException) \

{ \

hr = E\_OUTOFMEMORY; \

\*exceptionObject = Js::JavascriptExceptionOperators::GetOutOfMemoryExceptionObject(scriptContext); \

} \

}

#define BEGIN\_TRANSLATE\_EXCEPTION\_TO\_HRESULT BEGIN\_TRANSLATE\_TO\_HRESULT((ExceptionType)(ExceptionType\_OutOfMemory | ExceptionType\_StackOverflow))

#define BEGIN\_TRANSLATE\_EXCEPTION\_TO\_HRESULT\_NESTED BEGIN\_TRANSLATE\_TO\_HRESULT\_NESTED((ExceptionType)(ExceptionType\_OutOfMemory | ExceptionType\_StackOverflow))

#define BEGIN\_TRANSLATE\_EXCEPTION\_AND\_ERROROBJECT\_TO\_HRESULT BEGIN\_TRANSLATE\_TO\_HRESULT((ExceptionType)(ExceptionType\_OutOfMemory | ExceptionType\_StackOverflow | ExceptionType\_JavascriptException))

#define BEGIN\_TRANSLATE\_EXCEPTION\_AND\_ERROROBJECT\_TO\_HRESULT\_NESTED BEGIN\_TRANSLATE\_TO\_HRESULT\_NESTED((ExceptionType)(ExceptionType\_OutOfMemory | ExceptionType\_StackOverflow | ExceptionType\_JavascriptException))

#define END\_TRANSLATE\_KNOWN\_EXCEPTION\_TO\_HRESULT(hr) \

} \

catch (Js::InternalErrorException) \

{ \

hr = E\_FAIL; \

} \

catch (Js::OutOfMemoryException) \

{ \

hr = E\_OUTOFMEMORY; \

} \

catch (Js::StackOverflowException) \

{ \

hr = VBSERR\_OutOfStack; \

} \

catch (Js::NotImplementedException) \

{ \

hr = E\_NOTIMPL; \

} \

catch (Js::ScriptAbortException) \

{ \

hr = E\_ABORT; \

} \

catch (Js::AsmJsParseException) \

{ \

hr = JSERR\_AsmJsCompileError; \

}

#define CATCH\_UNHANDLED\_EXCEPTION(hr) \

catch (...) \

{ \

AssertMsg(FALSE, "invalid exception thrown and didn't get handled"); \

hr = E\_FAIL; \

} \

}

#define END\_TRANSLATE\_EXCEPTION\_TO\_HRESULT(hr) \

END\_TRANSLATE\_KNOWN\_EXCEPTION\_TO\_HRESULT(hr)\

CATCH\_UNHANDLED\_EXCEPTION(hr)

#define END\_TRANSLATE\_EXCEPTION\_AND\_ERROROBJECT\_TO\_HRESULT(hr) \

Assert(!JsUtil::ExternalApi::IsScriptActiveOnCurrentThreadContext()); \

END\_TRANSLATE\_KNOWN\_EXCEPTION\_TO\_HRESULT(hr) \

END\_TRANSLATE\_ERROROBJECT\_TO\_HRESULT(hr) \

CATCH\_UNHANDLED\_EXCEPTION(hr)

// Use this version if execution is in script (use rarely)

#define END\_TRANSLATE\_EXCEPTION\_AND\_ERROROBJECT\_TO\_HRESULT\_INSCRIPT(hr) \

Assert(JsUtil::ExternalApi::IsScriptActiveOnCurrentThreadContext()); \

END\_TRANSLATE\_KNOWN\_EXCEPTION\_TO\_HRESULT(hr) \

END\_TRANSLATE\_ERROROBJECT\_TO\_HRESULT\_INSCRIPT(hr) \

CATCH\_UNHANDLED\_EXCEPTION(hr)

#define END\_TRANSLATE\_EXCEPTION\_AND\_ERROROBJECT\_TO\_HRESULT\_NOASSERT(hr) \

END\_TRANSLATE\_KNOWN\_EXCEPTION\_TO\_HRESULT(hr) \

END\_TRANSLATE\_ERROROBJECT\_TO\_HRESULT(hr) \

CATCH\_UNHANDLED\_EXCEPTION(hr)

#define END\_TRANSLATE\_ERROROBJECT\_TO\_HRESULT\_EX(hr, GetRuntimeErrorFunc) \

catch(Js::JavascriptExceptionObject \* exceptionObject) \

{ \

GET\_RUNTIME\_ERROR\_IMPL(hr, GetRuntimeErrorFunc, exceptionObject); \

}

#define GET\_RUNTIME\_ERROR\_IMPL(hr, GetRuntimeErrorFunc, exceptionObject) \

{ \

Js::Var errorObject = exceptionObject->GetThrownObject(nullptr); \

if (errorObject != nullptr && (Js::JavascriptError::Is(errorObject) || \

Js::JavascriptError::IsRemoteError(errorObject))) \

{ \

hr = GetRuntimeErrorFunc(Js::RecyclableObject::FromVar(errorObject), nullptr); \

} \

else \

{ \

AssertMsg(errorObject == nullptr, "errorObject should be NULL"); \

hr = E\_OUTOFMEMORY; \

} \

}

#define GET\_RUNTIME\_ERROR(hr, exceptionObject) \

GET\_RUNTIME\_ERROR\_IMPL(hr, Js::JavascriptError::GetRuntimeErrorWithScriptEnter, exceptionObject)

#define END\_TRANSLATE\_ERROROBJECT\_TO\_HRESULT(hr) \

END\_TRANSLATE\_ERROROBJECT\_TO\_HRESULT\_EX(hr, Js::JavascriptError::GetRuntimeErrorWithScriptEnter)

#define END\_GET\_ERROROBJECT(hr, scriptContext, exceptionObject) \

catch (Js::JavascriptExceptionObject \* \_exceptionObject) \

{ \

BEGIN\_TRANSLATE\_OOM\_TO\_HRESULT\_NESTED \

exceptionObject = \_exceptionObject; \

exceptionObject = exceptionObject->CloneIfStaticExceptionObject(scriptContext); \

END\_TRANSLATE\_OOM\_TO\_HRESULT(hr) \

}

#define CATCH\_STATIC\_JAVASCRIPT\_EXCEPTION\_OBJECT \

catch (Js::OutOfMemoryException) \

{ \

return JsErrorOutOfMemory; \

} catch (Js::StackOverflowException) \

{ \

return JsErrorOutOfMemory; \

} \

#define CATCH\_OTHER\_EXCEPTIONS \

catch (JsrtExceptionBase& e) \

{ \

return e.GetJsErrorCode(); \

} \

catch (Js::ExceptionBase) \

{ \

AssertMsg(false, "Unexpected engine exception."); \

return JsErrorFatal; \

} \

catch (...) \

{ \

AssertMsg(false, "Unexpected non-engine exception."); \

return JsErrorFatal; \

}

// Use this version if execution is in script (use rarely)

#define END\_TRANSLATE\_ERROROBJECT\_TO\_HRESULT\_INSCRIPT(hr) \

END\_TRANSLATE\_ERROROBJECT\_TO\_HRESULT\_EX(hr, Js::JavascriptError::GetRuntimeError)

#define TRANSLATE\_EXCEPTION\_TO\_HRESULT\_ENTRY(ex) \

} \

catch (ex) \

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

#define RAISE\_FATL\_INTERNAL\_ERROR\_IFFAILED(hr) if (hr != S\_OK) Js::Throw::FatalInternalError();

#else

#define RAISE\_FATL\_INTERNAL\_ERROR\_IFFAILED(hr)

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// Empty placeholder for ExternalHelperMethod.h

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// Empty placeholder for ExternalHelperMethodList.h

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

//

// AllocationPolicyManager allows a caller/host to disallow new page allocations

// and to track current page usage.

//

// NOTE: For now, we are only tracking reserved page count.

// Consider whether we should also (or maybe only) track committed page count.

class AllocationPolicyManager

{

public:

enum MemoryAllocateEvent

{

MemoryAllocate = 0,

MemoryFree = 1,

MemoryFailure = 2,

MemoryMax = 2,

};

typedef bool (\_\_stdcall \* PageAllocatorMemoryAllocationCallback)(\_\_in LPVOID context,

\_\_in AllocationPolicyManager::MemoryAllocateEvent allocationEvent,

\_\_in size\_t allocationSize);

private:

size\_t memoryLimit;

size\_t currentMemory;

bool supportConcurrency;

CriticalSection cs;

void \* context;

PageAllocatorMemoryAllocationCallback memoryAllocationCallback;

public:

AllocationPolicyManager(bool needConcurrencySupport) :

memoryLimit((size\_t)-1),

currentMemory(0),

supportConcurrency(needConcurrencySupport),

context(NULL),

memoryAllocationCallback(NULL)

{

}

~AllocationPolicyManager()

{

Assert(currentMemory == 0);

}

size\_t GetUsage()

{

return currentMemory;

}

size\_t GetLimit()

{

return memoryLimit;

}

void SetLimit(size\_t newLimit)

{

memoryLimit = newLimit;

}

bool RequestAlloc(size\_t byteCount)

{

if (supportConcurrency)

{

AutoCriticalSection auto\_cs(&cs);

return RequestAllocImpl(byteCount);

}

else

{

return RequestAllocImpl(byteCount);

}

}

void ReportFailure(size\_t byteCount)

{

if (supportConcurrency)

{

AutoCriticalSection auto\_cs(&cs);

ReportFreeImpl(MemoryAllocateEvent::MemoryFailure, byteCount);

}

else

{

ReportFreeImpl(MemoryAllocateEvent::MemoryFailure, byteCount);

}

}

void ReportFree(size\_t byteCount)

{

if (supportConcurrency)

{

AutoCriticalSection auto\_cs(&cs);

ReportFreeImpl(MemoryAllocateEvent::MemoryFree, byteCount);

}

else

{

ReportFreeImpl(MemoryAllocateEvent::MemoryFree, byteCount);

}

}

void SetMemoryAllocationCallback(LPVOID newContext, PageAllocatorMemoryAllocationCallback callback)

{

this->memoryAllocationCallback = callback;

if (callback == NULL)

{

// doesn't make sense to have non-null context when the callback is NULL.

this->context = NULL;

}

else

{

this->context = newContext;

}

}

private:

\_\_inline bool RequestAllocImpl(size\_t byteCount)

{

size\_t newCurrentMemory = currentMemory + byteCount;

if (newCurrentMemory < currentMemory ||

newCurrentMemory > memoryLimit ||

memoryAllocationCallback != NULL && !memoryAllocationCallback(context, MemoryAllocateEvent::MemoryAllocate, byteCount))

{

if (memoryAllocationCallback != NULL)

{

memoryAllocationCallback(context, MemoryAllocateEvent::MemoryFailure, byteCount);

}

return false;

}

else

{

currentMemory = newCurrentMemory;

return true;

}

}

\_\_inline void ReportFreeImpl(MemoryAllocateEvent allocationEvent, size\_t byteCount)

{

Assert(currentMemory >= byteCount);

currentMemory = currentMemory - byteCount;

if (memoryAllocationCallback != NULL)

{

// The callback should be minimal, with no possibility of calling back to us.

// Note that this can be called both in script or out of script.

memoryAllocationCallback(context, allocationEvent, byteCount);

}

}

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// Disable the warning about no matching operator delete found, we don't need those for the Arena and Recycler

#pragma warning(disable:4291)

// Page heap mode is supported currently only in the Recycler

// Defining here so that other allocators can take advantage of this

// in the future

enum PageHeapMode

{

PageHeapModeOff = 0, // No Page heap

PageHeapModeBlockStart = 1, // Allocate the object at the beginning of the page

PageHeapModeBlockEnd = 2 // Allocate the object at the end of the page

};

#if PROFILE\_DICTIONARY

#include "DictionaryStats.h"

#endif

#if DBG || defined(RECYCLER\_FREE\_MEM\_FILL)

#define DbgMemFill 0XFE

#endif

namespace Memory

{

#ifdef TRACK\_ALLOC

struct TrackAllocData

{

void Clear() { typeinfo = nullptr; plusSize = 0; count = 0; }

bool IsEmpty() { return typeinfo == nullptr && plusSize == 0 && count == 0; }

type\_info const \* GetTypeInfo() const { return typeinfo; }

size\_t GetPlusSize() const { return plusSize; }

size\_t GetCount() const { return count; }

static TrackAllocData CreateTrackAllocData(type\_info const& typeinfo, size\_t size, size\_t count, char const \* const filename, DWORD line)

{

TrackAllocData data;

data.typeinfo = &typeinfo;

data.plusSize = size;

data.count = count;

data.filename = filename;

data.line = line;

return data;

};

type\_info const \* typeinfo;

size\_t plusSize;

size\_t count;

char const \* filename;

DWORD line;

};

#define TRACK\_ALLOC\_INFO(alloc, T, AllocatorType, size, count) static\_cast<AllocatorType \*>((alloc)->TrackAllocInfo(TrackAllocData::CreateTrackAllocData(typeid(T), size, count, \_\_FILE\_\_, \_\_LINE\_\_)))

#else

#define TRACK\_ALLOC\_INFO(alloc, T, AllocatorType, size, count) static\_cast<AllocatorType \*>(alloc)

#endif

#ifdef HEAP\_ENUMERATION\_VALIDATION

namespace Js {

class DynamicObject;

};

extern void PostAllocationCallbackForHeapEnumValidation(const type\_info&, Js::DynamicObject\*);

template <typename T>

inline T\* PostAllocationCallback(const type\_info& objType, T \*obj)

{

if (\_\_is\_base\_of(Js::DynamicObject, T))

{

PostAllocationCallbackForHeapEnumValidation(objType, (Js::DynamicObject\*)obj);

}

return obj;

}

#define VALIDATE\_OBJECT(T, obj) (PostAllocationCallback(typeid(T), obj))

#else

#define VALIDATE\_OBJECT(T, obj) obj

#endif

// Any allocator

#define AllocatorNewBase(AllocatorType, alloc, AllocFunc, T, ...) VALIDATE\_OBJECT(T, new (TRACK\_ALLOC\_INFO(static\_cast<AllocatorType \*>(alloc), T, AllocatorType, 0, (size\_t)-1), &AllocatorType::AllocFunc) T(\_\_VA\_ARGS\_\_))

#define AllocatorNewPlusBase(AllocatorType, alloc, AllocFunc, size, T, ...) VALIDATE\_OBJECT(T, new (TRACK\_ALLOC\_INFO(static\_cast<AllocatorType \*>(alloc), T, AllocatorType, size, (size\_t)-1), &AllocatorType::AllocFunc, size) T(\_\_VA\_ARGS\_\_))

#define AllocatorNewArrayBase(AllocatorType, alloc, AllocFunc, T, count) AllocateArray<AllocatorType, T, false>(TRACK\_ALLOC\_INFO(alloc, T, AllocatorType, 0, count), &AllocatorType::AllocFunc, count)

#define AllocatorNewStructBase(AllocatorType, alloc, AllocFunc, T) new (TRACK\_ALLOC\_INFO(static\_cast<AllocatorType \*>(alloc), T, AllocatorType, 0, (size\_t)-1), &AllocatorType::AllocFunc) T

#define AllocatorNewStructPlusBase(AllocatorType, alloc, AllocFunc, size, T) new (TRACK\_ALLOC\_INFO(static\_cast<AllocatorType \*>(alloc), T, AllocatorType, size, (size\_t)-1), &AllocatorType::AllocFunc, size) T

#define AllocatorNew(AllocatorType, alloc, T, ...) AllocatorNewBase(AllocatorType, alloc, Alloc, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewLeaf(AllocatorType, alloc, T, ...) AllocatorNewBase(AllocatorType, alloc, AllocLeaf, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewZ(AllocatorType, alloc, T, ...) AllocatorNewBase(AllocatorType, alloc, AllocZero, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewLeafZ(AllocatorType, alloc, T, ...) AllocatorNewBase(AllocatorType, alloc, AllocLeafZero, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewPlus(AllocatorType, alloc, size, T, ...) AllocatorNewPlusBase(AllocatorType, alloc, Alloc, size, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewPlusLeaf(AllocatorType, alloc, size, T, ...) AllocatorNewPlusBase(AllocatorType, alloc, AllocLeaf, size, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewPlusLeafZ(AllocatorType, alloc, size, T, ...) AllocatorNewPlusBase(AllocatorType, alloc, AllocLeafZero, size, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewPlusZ(AllocatorType, alloc, size, T, ...) AllocatorNewPlusBase(AllocatorType, alloc, AllocZero, size, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewStruct(AllocatorType, alloc, T) AllocatorNewStructBase(AllocatorType, alloc, Alloc, T)

#define AllocatorNewStructZ(AllocatorType, alloc, T) AllocatorNewStructBase(AllocatorType, alloc, AllocZero, T)

#define AllocatorNewStructLeaf(AllocatorType, alloc, T) AllocatorNewStructBase(AllocatorType, alloc, AllocLeaf, T)

#define AllocatorNewStructLeafZ(AllocatorType, alloc, T) AllocatorNewStructBase(AllocatorType, alloc, AllocLeafZero, T)

#define AllocatorNewStructPlus(AllocatorType, alloc, size, T) AllocatorNewStructPlusBase(AllocatorType, alloc, Alloc, size, T)

#define AllocatorNewStructPlusZ(AllocatorType, alloc, size, T) AllocatorNewStructPlusBase(AllocatorType, alloc, AllocZero, size, T)

#define AllocatorNewStructPlusLeaf(AllocatorType, alloc, size, T) AllocatorNewStructPlusBase(AllocatorType, alloc, AllocLeaf, size, T)

#define AllocatorNewStructPlusLeafZ(AllocatorType, alloc, size, T) AllocatorNewStructPlusBase(AllocatorType, alloc, AllocLeafZero, size, T);

#define AllocatorNewArray(AllocatorType, alloc, T, count) AllocatorNewArrayBase(AllocatorType, alloc, Alloc, T, count)

#define AllocatorNewArrayLeaf(AllocatorType, alloc, T, count) AllocatorNewArrayBase(AllocatorType, alloc, AllocLeaf, T, count)

#define AllocatorNewArrayZ(AllocatorType, alloc, T, count) AllocatorNewArrayBase(AllocatorType, alloc, AllocZero, T, count)

#define AllocatorNewArrayLeafZ(AllocatorType, alloc, T, count) AllocatorNewArrayBase(AllocatorType, alloc, AllocLeafZero, T, count)

#define AllocatorNewNoThrowBase(AllocatorType, alloc, AllocFunc, T, ...) new (TRACK\_ALLOC\_INFO(static\_cast<AllocatorType \*>(alloc), T, AllocatorType, 0, (size\_t)-1), true, &AllocatorType::NoThrow ## AllocFunc) T(\_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrowPlusBase(AllocatorType, alloc, AllocFunc, size, T, ...) new (TRACK\_ALLOC\_INFO(static\_cast<AllocatorType \*>(alloc), T, AllocatorType, size, (size\_t)-1), true, &AllocatorType::NoThrow ## AllocFunc, size) T(\_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrowArrayBase(AllocatorType, alloc, AllocFunc, T, count) AllocateArray<AllocatorType, T, true>(TRACK\_ALLOC\_INFO(alloc, T, AllocatorType, 0, count), &AllocatorType::NoThrow ## AllocFunc, count)

#define AllocatorNewNoThrowStructBase(AllocatorType, alloc, AllocFunc, T) new (TRACK\_ALLOC\_INFO(static\_cast<AllocatorType \*>(alloc), T, AllocatorType, 0, (size\_t)-1), true, &AllocatorType::NoThrow ## AllocFunc) T

#define AllocatorNewNoThrowStructPlusBase(AllocatorType, alloc, AllocFunc, size, T) new (TRACK\_ALLOC\_INFO(static\_cast<AllocatorType \*>(alloc), T, AllocatorType, size, (size\_t)-1), true, &AllocatorType::NoThrow ## AllocFunc, size) T

#define AllocatorNewNoThrowPlusPrefixBase(AllocatorType, alloc, AllocFunc, size, T, ...) new (TRACK\_ALLOC\_INFO(static\_cast<AllocatorType \*>(alloc), T, AllocatorType, size, (size\_t)-1), true, &AllocatorType::NoThrow ## AllocFunc, size, true) T(\_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrow(AllocatorType, alloc, T, ...) AllocatorNewNoThrowBase(AllocatorType, alloc, Alloc, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrowLeaf(AllocatorType, alloc, T, ...) AllocatorNewNoThrowBase(AllocatorType, alloc, AllocLeaf, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrowZ(AllocatorType, alloc, T, ...) AllocatorNewNoThrowBase(AllocatorType, alloc, AllocZero, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrowLeafZ(AllocatorType, alloc, T, ...) AllocatorNewNoThrowBase(AllocatorType, alloc, AllocLeafZero, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrowPlus(AllocatorType, alloc, size, T, ...) AllocatorNewNoThrowPlusBase(AllocatorType, alloc, Alloc, size, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrowPlusZ(AllocatorType, alloc, size, T, ...) AllocatorNewNoThrowPlusBase(AllocatorType, alloc, AllocZero, size, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrowPlusPrefixZ(AllocatorType, alloc, size, T, ...) AllocatorNewNoThrowPlusPrefixBase(AllocatorType, alloc, AllocZero, size, T, \_\_VA\_ARGS\_\_)

#define AllocatorNewNoThrowStruct(AllocatorType, alloc, T) AllocatorNewNoThrowStructBase(AllocatorType, alloc, Alloc, T)

#define AllocatorNewNoThrowStructZ(AllocatorType, alloc, T) AllocatorNewNoThrowStructBase(AllocatorType, alloc, AllocZero, T)

#define AllocatorNewNoThrowStructPlus(AllocatorType, alloc, size, T) AllocatorNewNoThrowStructPlusBase(AllocatorType, alloc, Alloc, size, T)

#define AllocatorNewNoThrowStructPlusZ(AllocatorType, alloc, size, T) AllocatorNewNoThrowStructPlusBase(AllocatorType, alloc, AllocZero, size, T)

#define AllocatorNewNoThrowStructPlusLeaf(AllocatorType, alloc, size, T) AllocatorNewNoThrowStructPlusBase(AllocatorType, alloc, AllocLeaf, size, T)

#define AllocatorNewNoThrowStructPlusLeafZ(AllocatorType, alloc, size, T) AllocatorNewNoThrowStructPlusBase(AllocatorType, alloc, AllocLeafZero, size, T);

#define AllocatorNewNoThrowArray(AllocatorType, alloc, T, count) AllocatorNewNoThrowArrayBase(AllocatorType, alloc, Alloc, T, count)

#define AllocatorNewNoThrowArrayLeaf(AllocatorType, alloc, T, count) AllocatorNewNoThrowArrayBase(AllocatorType, alloc, AllocLeaf, T, count)

#define AllocatorNewNoThrowArrayZ(AllocatorType, alloc, T, count) AllocatorNewNoThrowArrayBase(AllocatorType, alloc, AllocZero, T, count)

#define AllocatorNewNoThrowNoRecoveryArrayBase(AllocatorType, alloc, AllocFunc, T, count) AllocateArray<AllocatorType, T, true>(TRACK\_ALLOC\_INFO(alloc, T, AllocatorType, 0, count), &AllocatorType::NoThrowNoRecovery ## AllocFunc, count)

#define AllocatorNewNoThrowNoRecoveryArrayZ(AllocatorType, alloc, T, count) AllocatorNewNoThrowNoRecoveryArrayBase(AllocatorType, alloc, AllocZero, T, count)

#define AllocatorDelete(AllocatorType, alloc, obj) DeleteObject<AllocatorType>(alloc, obj)

#define AllocatorDeleteInline(AllocatorType, alloc, obj) DeleteObjectInline<AllocatorType>(alloc, obj)

#define AllocatorDeleteLeaf(TAllocator, alloc, obj) DeleteObject<ForceLeafAllocator<TAllocator>::AllocatorType>(alloc, obj)

#define AllocatorDeletePlus(AllocatorType, alloc, size, obj) DeleteObject<AllocatorType>(alloc, obj, size);

#define AllocatorDeletePlusLeaf(TAllocator, alloc, size, obj) DeleteObject<ForceLeafAllocator<TAllocator>::AllocatorType>(alloc, obj, size);

#define AllocatorDeletePlusPrefix(AllocatorType, alloc, size, obj) DeleteObject<AllocatorType>(alloc, obj, size, true);

#define AllocatorDeletePlusPrefixLeaf(TAllocator, alloc, size, obj) DeleteObject<ForceLeafAllocator<TAllocator>::AllocatorType>(alloc, obj, size, true);

#define AllocatorDeleteArray(AllocatorType, alloc, count, obj) DeleteArray<AllocatorType>(alloc, count, obj)

#define AllocatorDeleteArrayLeaf(TAllocator, alloc, count, obj) DeleteArray<ForceLeafAllocator<TAllocator>::AllocatorType>(alloc, count, obj)

// Free routine where we don't care about following C++ semantics (e.g. calling the destructor)

#define AllocatorFree(alloc, freeFunc, obj, size) (alloc->\*freeFunc)(obj, size)

// default type allocator implementation

template <typename TAllocator, typename T>

class TypeAllocatorFunc

{

public:

typedef char \* (TAllocator::\*AllocFuncType)(size\_t);

typedef void(TAllocator::\*FreeFuncType)(void\*, size\_t);

static AllocFuncType GetAllocFunc()

{

return &TAllocator::Alloc;

}

static AllocFuncType GetAllocZeroFunc()

{

return &TAllocator::AllocZero;

}

static FreeFuncType GetFreeFunc()

{

return &TAllocator::Free;

}

};

// List specific allocator info

template <typename TAllocator, bool isLeaf>

class ListTypeAllocatorFunc

{

public:

typedef void(TAllocator::\*FreeFuncType)(void\*, size\_t);

static FreeFuncType GetFreeFunc()

{

return &TAllocator::Free;

}

};

// Default allocation policy

template <typename TAllocator, typename TAllocType>

struct AllocatorInfo

{

typedef TAllocator AllocatorType;

typedef TAllocator TemplateAllocatorType;

typedef TypeAllocatorFunc<TAllocator, TAllocType> AllocatorFunc; // allocate/free an array of given type

typedef TypeAllocatorFunc<TAllocator, TAllocType> InstAllocatorFunc;// allocate/free an object instance itself of given type

};

// Allocator doesn't change for by default for forcing non leaf

template <typename TAllocator>

struct ForceNonLeafAllocator

{

typedef TAllocator AllocatorType;

};

// Allocator doesn't change for by default for forcing leaf

template <typename TAllocator>

struct ForceLeafAllocator

{

typedef TAllocator AllocatorType;

};

template <typename TAllocator, typename T>

void DeleteObject(typename AllocatorInfo<TAllocator, T>::AllocatorType \* allocator, T \* obj)

{

obj->~T();

auto freeFunc = AllocatorInfo<TAllocator, T>::InstAllocatorFunc::GetFreeFunc(); // Use InstAllocatorFunc

(allocator->\*freeFunc)(obj, sizeof(T));

}

template <typename TAllocator, typename T>

void DeleteObjectInline(typename TAllocator \* allocator, T \* obj)

{

obj->~T();

allocator->FreeInline(obj, sizeof(T));

}

template <typename TAllocator, typename T>

void DeleteObject(typename AllocatorInfo<TAllocator, T>::AllocatorType \* allocator, T \* obj, size\_t plusSize)

{

obj->~T();

// DeleteObject can only be called when an object is allocated successfully.

// So the add should never overflow

Assert(sizeof(T) + plusSize >= sizeof(T));

auto freeFunc = AllocatorInfo<TAllocator, T>::InstAllocatorFunc::GetFreeFunc(); // Use InstAllocatorFunc

(allocator->\*freeFunc)(obj, sizeof(T) + plusSize);

}

template <typename TAllocator, typename T>

void DeleteObject(typename AllocatorInfo<TAllocator, T>::AllocatorType \* allocator, T \* obj, size\_t plusSize, bool prefix)

{

Assert(prefix);

obj->~T();

// DeleteObject can only be called when an object is allocated successfully.

// So the add should never overflow

Assert(sizeof(T) + plusSize >= sizeof(T));

// NOTE: This may cause the object not be double aligned

Assert(plusSize == Math::Align<size\_t>(plusSize, sizeof(size\_t)));

auto freeFunc = AllocatorInfo<TAllocator, T>::InstAllocatorFunc::GetFreeFunc(); // Use InstAllocatorFunc

(allocator->\*freeFunc)(((char \*)obj) - plusSize, sizeof(T) + plusSize);

}

#define ZERO\_LENGTH\_ARRAY (void \*)sizeof(void \*)

template <typename TAllocator, typename T, bool nothrow>

\_When\_(nothrow, \_Ret\_writes\_to\_maybenull\_(count, 0)) \_When\_(!nothrow, \_Ret\_writes\_to\_(count, 0))

\_\_inline T \* AllocateArray(TAllocator \* allocator, char \* (TAllocator::\*AllocFunc)(size\_t), size\_t count)

{

if (count == 0 && TAllocator::FakeZeroLengthArray)

{

#ifdef TRACK\_ALLOC

allocator->ClearTrackAllocInfo();

#endif

// C++ standard requires allocator to return non-null if it isn't out of memory

// Just return some small number so we will still AV if someone try to use the memory

return (T \*)ZERO\_LENGTH\_ARRAY;

}

if (nothrow)

{

return new (allocator, nothrow, AllocFunc) T[count];

}

return new (allocator, AllocFunc) T[count];

}

template <typename TAllocator, typename T>

void DeleteArray(typename AllocatorInfo<TAllocator, T>::AllocatorType \* allocator, size\_t count, T \* obj)

{

if(count == 0)

{

return;

}

for (size\_t i = 0; i < count; i++)

{

obj[i].~T();

}

// DeleteArray can only be called when an array is allocated successfully.

// So the add should never overflow

Assert(count \* sizeof(T) / count == sizeof(T));

auto freeFunc = AllocatorInfo<TAllocator, T>::AllocatorFunc::GetFreeFunc();

(allocator->\*freeFunc)((void \*)obj, sizeof(T) \* count);

}

#define AllocatorFieldMove(dest, src, field) \

Assert(dest->field == 0); \

dest->field = src->field; \

src->field = 0;

class Allocator

{

public:

Allocator(void (\*outOfMemoryFunc)(), void (\*recoverMemoryFunc)() = JsUtil::ExternalApi::RecoverUnusedMemory) : outOfMemoryFunc(outOfMemoryFunc), recoverMemoryFunc(recoverMemoryFunc) {}

void Move(Allocator \*srcAllocator)

{

Assert(srcAllocator != nullptr);

AllocatorFieldMove(this, srcAllocator, outOfMemoryFunc);

}

void (\*outOfMemoryFunc)();

void (\*recoverMemoryFunc)();

};

template <typename T>

void AssertValue(void \* mem, T value, uint byteCount)

{

#if DBG

Assert(byteCount % sizeof(T) == 0);

for (uint i = 0; i < byteCount; i += sizeof(T))

{

Assert(\*(T \*)(((char \*)mem) + i) == value);

}

#endif

}

}

// For the debugger extension, we don't need the placement news

#ifndef \_\_PLACEMENT\_NEW\_INLINE

#define \_\_PLACEMENT\_NEW\_INLINE

\_Ret\_notnull\_

\_\_inline void \* \_\_cdecl

operator new(

size\_t byteSize,

\_In\_ void \* previousAllocation) throw()

{

return previousAllocation;

}

\_\_inline void \_\_cdecl

operator delete(

void \* allocationToFree, // Allocation to free

void \* previousAllocation // Previously allocated memory

) throw()

{

}

#endif

//----------------------------------------

// throwing operator new overrides

//----------------------------------------

template <typename TAllocator>

\_Ret\_notnull\_ void \* \_\_cdecl

operator new(size\_t byteSize, TAllocator \* alloc, char \* (TAllocator::\*AllocFunc)(size\_t))

{

AssertCanHandleOutOfMemory();

Assert(byteSize != 0);

void \* buffer = (alloc->\*AllocFunc)(byteSize);

Assume(buffer != nullptr);

return buffer;

}

template <typename TAllocator>

\_Ret\_notnull\_ \_\_inline void \* \_\_cdecl

operator new[](size\_t byteSize, TAllocator \* alloc, char \* (TAllocator::\*AllocFunc)(size\_t))

{

AssertCanHandleOutOfMemory();

Assert(byteSize != 0 || !TAllocator::FakeZeroLengthArray);

void \* buffer = (alloc->\*AllocFunc)(byteSize);

Assume(buffer != nullptr);

return buffer;

}

template <typename TAllocator>

\_Ret\_notnull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, TAllocator \* alloc, char \* (TAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

AssertCanHandleOutOfMemory();

Assert(byteSize != 0);

//Assert(plusSize != 0);

// byteSize is usually a compile-time constant, so put it on the RHS of the add for

// slightly better (smaller and faster) code.

void \* buffer = (alloc->\*AllocFunc)(AllocSizeMath::Add(plusSize, byteSize));

Assume(buffer != nullptr);

return buffer;

}

//----------------------------------------

// nothrow operator new overrides

//----------------------------------------

template <typename TAllocator>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, TAllocator \* alloc, bool nothrow, char \* (TAllocator::\*AllocFunc)(size\_t))

{

Assert(nothrow);

Assert(byteSize != 0);

void \* buffer = (alloc->\*AllocFunc)(byteSize);

return buffer;

}

template <typename TAllocator>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new[](size\_t byteSize, TAllocator \* alloc, bool nothrow, char \* (TAllocator::\*AllocFunc)(size\_t))

{

Assert(nothrow);

Assert(byteSize != 0 || !TAllocator::FakeZeroLengthArray);

void \* buffer = (alloc->\*AllocFunc)(byteSize);

return buffer;

}

template <typename TAllocator>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, TAllocator \* alloc, bool nothrow, char \* (TAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

Assert(nothrow);

Assert(byteSize != 0);

//Assert(plusSize != 0);

// byteSize is usually a compile-time constant, so put it on the RHS of the add for

// slightly better (smaller and faster) code.

void \* buffer = (alloc->\*AllocFunc)(AllocSizeMath::Add(plusSize, byteSize));

return buffer;

}

template <typename TAllocator>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, TAllocator \* alloc, bool nothrow, char \* (TAllocator::\*AllocFunc)(size\_t), size\_t plusSize, bool prefix)

{

Assert(nothrow);

Assert(prefix);

Assert(byteSize != 0);

Assert(plusSize != 0);

// NOTE: This may cause the object not be double aligned

Assert(plusSize == Math::Align<size\_t>(plusSize, sizeof(size\_t)));

// byteSize is usually a compile-time constant, so put it on the RHS of the add for

// slightly better (smaller and faster) code.

char \* buffer = (alloc->\*AllocFunc)(AllocSizeMath::Add(plusSize, byteSize));

// This seems to generate the most compact code

return buffer + (buffer > 0 ? plusSize : (size\_t)buffer);

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#define ASSERT\_THREAD() AssertMsg(this->pageAllocator->ValidThreadAccess(), "Arena allocation should only be used by a single thread")

template \_\_forceinline BVSparseNode \* BVSparse<JitArenaAllocator>::NodeFromIndex(BVIndex i, BVSparseNode \*\*\* prevNextFieldOut, bool create);

ArenaData::ArenaData(PageAllocator \* pageAllocator) :

pageAllocator(pageAllocator),

bigBlocks(nullptr),

mallocBlocks(nullptr),

fullBlocks(nullptr),

cacheBlockCurrent(nullptr),

lockBlockList(false)

{

}

void ArenaData::UpdateCacheBlock() const

{

if (bigBlocks != nullptr)

{

size\_t currentByte = (cacheBlockCurrent - bigBlocks->GetBytes());

// Avoid writing to the page unnecessary, it might be write watched

if (currentByte != bigBlocks->currentByte)

{

bigBlocks->currentByte = currentByte;

}

}

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

ArenaAllocatorBase(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void(\*outOfMemoryFunc)(), void(\*recoverMemoryFunc)()) :

Allocator(outOfMemoryFunc, recoverMemoryFunc),

ArenaData(pageAllocator),

#ifdef ARENA\_ALLOCATOR\_FREE\_LIST\_SIZE

freeListSize(0),

#endif

freeList(nullptr),

largestHole(0),

cacheBlockEnd(nullptr),

blockState(0)

{

#ifdef PROFILE\_MEM

this->name = name;

LogBegin();

#endif

ArenaMemoryTracking::ArenaCreated(this, name);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

~ArenaAllocatorBase()

{

Assert(!lockBlockList);

ArenaMemoryTracking::ReportFreeAll(this);

ArenaMemoryTracking::ArenaDestroyed(this);

if (!pageAllocator->IsClosed())

{

ReleasePageMemory();

}

ReleaseHeapMemory();

TFreeListPolicy::Release(this->freeList);

#ifdef PROFILE\_MEM

LogEnd();

#endif

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

Move(ArenaAllocatorBase \*srcAllocator)

{

Assert(!lockBlockList);

Assert(srcAllocator != nullptr);

Allocator::Move(srcAllocator);

Assert(this->pageAllocator == srcAllocator->pageAllocator);

AllocatorFieldMove(this, srcAllocator, bigBlocks);

AllocatorFieldMove(this, srcAllocator, largestHole);

AllocatorFieldMove(this, srcAllocator, cacheBlockCurrent);

AllocatorFieldMove(this, srcAllocator, cacheBlockEnd);

AllocatorFieldMove(this, srcAllocator, mallocBlocks);

AllocatorFieldMove(this, srcAllocator, fullBlocks);

AllocatorFieldMove(this, srcAllocator, blockState);

AllocatorFieldMove(this, srcAllocator, freeList);

#ifdef PROFILE\_MEM

this->name = srcAllocator->name;

srcAllocator->name = nullptr;

AllocatorFieldMove(this, srcAllocator, memoryData);

#endif

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

size\_t

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

AllocatedSize(ArenaMemoryBlock \* blockList)

{

ArenaMemoryBlock \* memoryBlock = blockList;

size\_t totalBytes = 0;

while (memoryBlock != NULL)

{

totalBytes += memoryBlock->nbytes;

memoryBlock = memoryBlock->next;

}

return totalBytes;

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

size\_t

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

AllocatedSize()

{

UpdateCacheBlock();

return AllocatedSize(this->fullBlocks) + AllocatedSize(this->bigBlocks) + AllocatedSize(this->mallocBlocks);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

size\_t

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

Size()

{

UpdateCacheBlock();

return Size(this->fullBlocks) + Size(this->bigBlocks) + AllocatedSize(this->mallocBlocks);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

size\_t

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

Size(BigBlock \* blockList)

{

BigBlock \* memoryBlock = blockList;

size\_t totalBytes = 0;

while (memoryBlock != NULL)

{

totalBytes += memoryBlock->currentByte;

memoryBlock = (BigBlock \*)memoryBlock->next;

}

return totalBytes;

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

char \*

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

RealAlloc(size\_t nbytes)

{

return RealAllocInlined(nbytes);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

char \*

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

RealAllocInlined(size\_t nbytes)

{

Assert(nbytes != 0);

Assert((nbytes & (ObjectAlignment - 1)) == 0);

#ifdef ARENA\_MEMORY\_VERIFY

if (Js::Configuration::Global.flags.ArenaUseHeapAlloc)

{

return AllocFromHeap<true>(nbytes);

}

#endif

Assert(cacheBlockEnd >= cacheBlockCurrent);

char \* p = cacheBlockCurrent;

if ((size\_t)(cacheBlockEnd - p) >= nbytes)

{

Assert(cacheBlockEnd == bigBlocks->GetBytes() + bigBlocks->nbytes);

Assert(bigBlocks->GetBytes() <= cacheBlockCurrent && cacheBlockCurrent <= cacheBlockEnd);

cacheBlockCurrent = p + nbytes;

ArenaMemoryTracking::ReportAllocation(this, p, nbytes);

return(p);

}

return SnailAlloc(nbytes);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

SetCacheBlock(BigBlock \* newCacheBlock)

{

if (bigBlocks != nullptr)

{

Assert(cacheBlockEnd == bigBlocks->GetBytes() + bigBlocks->nbytes);

Assert(bigBlocks->GetBytes() <= cacheBlockCurrent && cacheBlockCurrent <= cacheBlockEnd);

bigBlocks->currentByte = (cacheBlockCurrent - bigBlocks->GetBytes());

uint cacheBlockRemainBytes = (uint)(cacheBlockEnd - cacheBlockCurrent);

if (cacheBlockRemainBytes < ObjectAlignment && !lockBlockList)

{

BigBlock \* cacheBlock = bigBlocks;

bigBlocks = bigBlocks->nextBigBlock;

cacheBlock->next = fullBlocks;

fullBlocks = cacheBlock;

}

else

{

largestHole = max(largestHole, static\_cast<size\_t>(cacheBlockRemainBytes));

}

}

cacheBlockCurrent = newCacheBlock->GetBytes() + newCacheBlock->currentByte;

cacheBlockEnd = newCacheBlock->GetBytes() + newCacheBlock->nbytes;

newCacheBlock->nextBigBlock = bigBlocks;

bigBlocks = newCacheBlock;

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

char \*

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

SnailAlloc(size\_t nbytes)

{

BigBlock\* blockp = NULL;

size\_t currentLargestHole = 0;

if (nbytes <= largestHole)

{

Assert(bigBlocks != nullptr);

Assert(cacheBlockEnd == bigBlocks->GetBytes() + bigBlocks->nbytes);

Assert(bigBlocks->GetBytes() <= cacheBlockCurrent && cacheBlockCurrent <= cacheBlockEnd);

BigBlock \* cacheBlock = bigBlocks;

BigBlock\*\* pPrev= &(bigBlocks->nextBigBlock);

blockp = bigBlocks->nextBigBlock;

int giveUpAfter = 10;

do

{

size\_t remainingBytes = blockp->nbytes - blockp->currentByte;

if (remainingBytes >= nbytes)

{

char \*p = blockp->GetBytes() + blockp->currentByte;

blockp->currentByte += nbytes;

if (remainingBytes == largestHole || currentLargestHole > largestHole)

{

largestHole = currentLargestHole;

}

remainingBytes -= nbytes;

if (remainingBytes > cacheBlock->nbytes - cacheBlock->currentByte)

{

\*pPrev = blockp->nextBigBlock;

SetCacheBlock(blockp);

}

else if (remainingBytes < ObjectAlignment && !lockBlockList)

{

\*pPrev = blockp->nextBigBlock;

blockp->nextBigBlock = fullBlocks;

fullBlocks = blockp;

}

ArenaMemoryTracking::ReportAllocation(this, p, nbytes);

return(p);

}

currentLargestHole = max(currentLargestHole, remainingBytes);

if (--giveUpAfter == 0)

{

break;

}

pPrev = &(blockp->nextBigBlock);

blockp = blockp->nextBigBlock;

}

while (blockp != nullptr);

}

blockp = AddBigBlock(nbytes);

if (blockp == nullptr)

{

return AllocFromHeap<false>(nbytes); // Passing DoRecoverMemory=false as we already tried recovering memory in AddBigBlock, and it is costly.

}

this->blockState++;

SetCacheBlock(blockp);

char \*p = cacheBlockCurrent;

Assert(p + nbytes <= cacheBlockEnd);

cacheBlockCurrent += nbytes;

ArenaMemoryTracking::ReportAllocation(this, p, nbytes);

return(p);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

template <bool DoRecoverMemory>

char \*

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

AllocFromHeap(size\_t requestBytes)

{

size\_t allocBytes = AllocSizeMath::Add(requestBytes, sizeof(ArenaMemoryBlock));

ARENA\_FAULTINJECT\_MEMORY(this->name, requestBytes);

char \* buffer = HeapNewNoThrowArray(char, allocBytes);

if (buffer == nullptr)

{

if (DoRecoverMemory && recoverMemoryFunc)

{

// Try to recover some memory and see if after that we can allocate.

recoverMemoryFunc();

buffer = HeapNewNoThrowArray(char, allocBytes);

}

if (buffer == nullptr)

{

if (outOfMemoryFunc)

{

outOfMemoryFunc();

}

return nullptr;

}

}

ArenaMemoryBlock \* memoryBlock = (ArenaMemoryBlock \*)buffer;

memoryBlock->nbytes = requestBytes;

memoryBlock->next = this->mallocBlocks;

this->mallocBlocks = memoryBlock;

this->blockState = 2; // set the block state to 2 to disable the reset fast path.

ArenaMemoryTracking::ReportAllocation(this, buffer + sizeof(ArenaMemoryBlock), requestBytes);

return buffer + sizeof(ArenaMemoryBlock);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

BigBlock \*

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

AddBigBlock(size\_t requestBytes)

{

FAULTINJECT\_MEMORY\_NOTHROW(this->name, requestBytes);

size\_t allocBytes = AllocSizeMath::Add(requestBytes, sizeof(BigBlock));

PageAllocation \* allocation = this->GetPageAllocator()->AllocPagesForBytes(allocBytes);

if (allocation == nullptr)

{

// Try to recover some memory and see if after that we can allocate.

if (recoverMemoryFunc)

{

recoverMemoryFunc();

allocation = this->GetPageAllocator()->AllocPagesForBytes(allocBytes);

}

if (allocation == nullptr)

{

return nullptr;

}

}

BigBlock \* blockp = (BigBlock \*)allocation->GetAddress();

blockp->allocation = allocation;

blockp->nbytes = allocation->GetSize() - sizeof(BigBlock);

blockp->currentByte = 0;

#ifdef PROFILE\_MEM

LogRealAlloc(allocation->GetSize() + sizeof(PageAllocation));

#endif

return(blockp);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

FullReset()

{

BigBlock \* initBlock = this->bigBlocks;

if (initBlock != nullptr)

{

this->bigBlocks = initBlock->nextBigBlock;

}

Clear();

if (initBlock != nullptr)

{

this->blockState = 1;

initBlock->currentByte = 0;

SetCacheBlock(initBlock);

}

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

ReleaseMemory()

{

ReleasePageMemory();

ReleaseHeapMemory();

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

ReleasePageMemory()

{

pageAllocator->SuspendIdleDecommit();

#ifdef ARENA\_MEMORY\_VERIFY

bool reenableDisablePageReuse = false;

if (Js::Configuration::Global.flags.ArenaNoPageReuse)

{

reenableDisablePageReuse = !pageAllocator->DisablePageReuse();

}

#endif

BigBlock \*blockp = bigBlocks;

while (blockp != NULL)

{

PageAllocation \* allocation = blockp->allocation;

blockp = blockp->nextBigBlock;

GetPageAllocator()->ReleaseAllocationNoSuspend(allocation);

}

blockp = fullBlocks;

while (blockp != NULL)

{

PageAllocation \* allocation = blockp->allocation;

blockp = blockp->nextBigBlock;

GetPageAllocator()->ReleaseAllocationNoSuspend(allocation);

}

#ifdef ARENA\_MEMORY\_VERIFY

if (reenableDisablePageReuse)

{

pageAllocator->ReenablePageReuse();

}

#endif

pageAllocator->ResumeIdleDecommit();

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

ReleaseHeapMemory()

{

ArenaMemoryBlock \* memoryBlock = this->mallocBlocks;

while (memoryBlock != nullptr)

{

ArenaMemoryBlock \* next = memoryBlock->next;

HeapDeleteArray(memoryBlock->nbytes + sizeof(ArenaMemoryBlock), (char \*)memoryBlock);

memoryBlock = next;

}

}

template \_\_forceinline char \*ArenaAllocatorBase<InPlaceFreeListPolicy, 0, 0, 0>::AllocInternal(size\_t requestedBytes);

template \_\_forceinline char \*ArenaAllocatorBase<InPlaceFreeListPolicy, 3, 0, 0>::AllocInternal(size\_t requestedBytes);

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

char \*

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

AllocInternal(size\_t requestedBytes)

{

Assert(requestedBytes != 0);

if (MaxObjectSize > 0)

{

Assert(requestedBytes <= MaxObjectSize);

}

if (RequireObjectAlignment)

{

Assert(requestedBytes % ObjectAlignment == 0);

}

// If out of memory function is set, that means that the caller is a throwing allocation

// routine, so we can throw from here. Otherwise, we shouldn't throw.

ARENA\_FAULTINJECT\_MEMORY(this->name, requestedBytes);

ASSERT\_THREAD();

size\_t nbytes;

if (freeList != nullptr && requestedBytes > 0 && requestedBytes <= ArenaAllocatorBase::MaxSmallObjectSize)

{

// We have checked the size requested, so no integer overflow check

nbytes = Math::Align(requestedBytes, ArenaAllocator::ObjectAlignment);

Assert(nbytes <= ArenaAllocator::MaxSmallObjectSize);

#ifdef PROFILE\_MEM

LogAlloc(requestedBytes, nbytes);

#endif

void \* freeObject = TFreeListPolicy::Allocate(this->freeList, nbytes);

if (freeObject != nullptr)

{

#ifdef ARENA\_MEMORY\_VERIFY

TFreeListPolicy::VerifyFreeObjectIsFreeMemFilled(freeObject, nbytes);

#endif

#ifdef ARENA\_ALLOCATOR\_FREE\_LIST\_SIZE

this->freeListSize -= nbytes;

#endif

#ifdef PROFILE\_MEM

LogReuse(nbytes);

#endif

ArenaMemoryTracking::ReportAllocation(this, freeObject, nbytes);

return (char \*)freeObject;

}

}

else

{

nbytes = AllocSizeMath::Align(requestedBytes, ArenaAllocator::ObjectAlignment);

#ifdef PROFILE\_MEM

LogAlloc(requestedBytes, nbytes);

#endif

}

// TODO: Support large object free listing

return ArenaAllocatorBase::RealAllocInlined(nbytes);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

Free(void \* buffer, size\_t byteSize)

{

ASSERT\_THREAD();

Assert(byteSize != 0);

if (MaxObjectSize > 0)

{

Assert(byteSize <= MaxObjectSize);

}

if (RequireObjectAlignment)

{

Assert(byteSize % ObjectAlignment == 0);

}

// Since we successfully allocated, we shouldn't have integer overflow here

size\_t size = Math::Align(byteSize, ArenaAllocator::ObjectAlignment);

Assert(size >= byteSize);

ArenaMemoryTracking::ReportFree(this, buffer, byteSize);

#ifdef ARENA\_MEMORY\_VERIFY

if (Js::Configuration::Global.flags.ArenaNoFreeList)

{

return;

}

#endif

if (buffer == cacheBlockCurrent - byteSize)

{

#ifdef PROFILE\_MEM

LogFree(byteSize);

#endif

cacheBlockCurrent = (char \*)buffer;

return;

}

else if (this->pageAllocator->IsClosed())

{

return;

}

else if (size <= ArenaAllocator::MaxSmallObjectSize)

{

// If we plan to free-list this object, we must prepare (typically, debug pattern fill) its memory here, in case we fail to allocate the free list because we're out of memory (see below),

// and we never get to call TFreeListPolicy::Free.

TFreeListPolicy::PrepareFreeObject(buffer, size);

if (freeList == nullptr)

{

// Caution: TFreeListPolicy::New may fail silently if we're out of memory.

freeList = TFreeListPolicy::New(this);

if (freeList == nullptr)

{

return;

}

}

this->freeList = TFreeListPolicy::Free(this->freeList, buffer, size);

#ifdef ARENA\_ALLOCATOR\_FREE\_LIST\_SIZE

this->freeListSize += size;

#endif

#ifdef PROFILE\_MEM

LogFree(byteSize);

#endif

return;

}

// TODO: Free list bigger objects

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

char \*

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

Realloc(void\* buffer, size\_t existingBytes, size\_t requestedBytes)

{

ASSERT\_THREAD();

if (existingBytes == 0)

{

Assert(buffer == nullptr);

return AllocInternal(requestedBytes);

}

if (MaxObjectSize > 0)

{

Assert(requestedBytes <= MaxObjectSize);

}

if (RequireObjectAlignment)

{

Assert(requestedBytes % ObjectAlignment == 0);

}

size\_t nbytes = AllocSizeMath::Align(requestedBytes, ArenaAllocator::ObjectAlignment);

// Since we successfully allocated, we shouldn't have integer overflow here

size\_t nbytesExisting = Math::Align(existingBytes, ArenaAllocator::ObjectAlignment);

Assert(nbytesExisting >= existingBytes);

if (nbytes == nbytesExisting)

{

return (char \*)buffer;

}

if (nbytes < nbytesExisting)

{

ArenaMemoryTracking::ReportReallocation(this, buffer, nbytesExisting, nbytes);

Free(((char \*)buffer) + nbytes, nbytesExisting - nbytes);

return (char \*)buffer;

}

char\* replacementBuf = nullptr;

if (requestedBytes > 0)

{

replacementBuf = AllocInternal(requestedBytes);

if (replacementBuf != nullptr)

{

js\_memcpy\_s(replacementBuf, requestedBytes, buffer, existingBytes);

}

}

if (nbytesExisting > 0)

{

Free(buffer, nbytesExisting);

}

return replacementBuf;

}

#ifdef PROFILE\_MEM

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

LogBegin()

{

memoryData = MemoryProfiler::Begin(this->name);

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

LogReset()

{

if (memoryData)

{

MemoryProfiler::Reset(this->name, memoryData);

}

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

LogEnd()

{

if (memoryData)

{

MemoryProfiler::End(this->name, memoryData);

}

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

LogAlloc(size\_t requestedBytes, size\_t allocateBytes)

{

if (memoryData)

{

memoryData->requestCount++;

memoryData->requestBytes += requestedBytes;

memoryData->alignmentBytes += allocateBytes - requestedBytes;

}

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

LogRealAlloc(size\_t size)

{

if (memoryData)

{

memoryData->allocatedBytes += size;

}

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

LogFree(size\_t size)

{

if (memoryData)

{

memoryData->freelistBytes += size;

memoryData->freelistCount++;

}

}

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg, bool RequireObjectAlignment, size\_t MaxObjectSize>

void

ArenaAllocatorBase<TFreeListPolicy, ObjectAlignmentBitShiftArg, RequireObjectAlignment, MaxObjectSize>::

LogReuse(size\_t size)

{

if (memoryData)

{

memoryData->reuseCount++;

memoryData->reuseBytes += size;

}

}

#endif

void \* InPlaceFreeListPolicy::New(ArenaAllocatorBase<InPlaceFreeListPolicy> \* allocator)

{

return AllocatorNewNoThrowNoRecoveryArrayZ(ArenaAllocator, allocator, FreeObject \*, buckets);

}

void \* InPlaceFreeListPolicy::Allocate(void \* policy, size\_t size)

{

Assert(policy);

FreeObject \*\* freeObjectLists = reinterpret\_cast<FreeObject \*\*>(policy);

size\_t index = (size >> ArenaAllocator::ObjectAlignmentBitShift) - 1;

FreeObject \* freeObject = freeObjectLists[index];

if (NULL != freeObject)

{

freeObjectLists[index] = freeObject->next;

#ifdef ARENA\_MEMORY\_VERIFY

// Make sure the next pointer bytes are also DbgFreeMemFill-ed.

memset(freeObject, DbgFreeMemFill, sizeof(freeObject->next));

#endif

}

return freeObject;

}

void \* InPlaceFreeListPolicy::Free(void \* policy, void \* object, size\_t size)

{

Assert(policy);

FreeObject \*\* freeObjectLists = reinterpret\_cast<FreeObject \*\*>(policy);

FreeObject \* freeObject = reinterpret\_cast<FreeObject \*>(object);

size\_t index = (size >> ArenaAllocator::ObjectAlignmentBitShift) - 1;

freeObject->next = freeObjectLists[index];

freeObjectLists[index] = freeObject;

return policy;

}

void \* InPlaceFreeListPolicy::Reset(void \* policy)

{

return NULL;

}

#ifdef ARENA\_MEMORY\_VERIFY

void InPlaceFreeListPolicy::VerifyFreeObjectIsFreeMemFilled(void \* object, size\_t size)

{

unsigned char \* bytes = reinterpret\_cast<unsigned char\*>(object);

for (size\_t i = 0; i < size; i++)

{

Assert(bytes[i] == InPlaceFreeListPolicy::DbgFreeMemFill);

}

}

#endif

template class ArenaAllocatorBase<InPlaceFreeListPolicy>;

void \* StandAloneFreeListPolicy::New(ArenaAllocatorBase<StandAloneFreeListPolicy> \* /\*allocator\*/)

{

return NewInternal(InitialEntries);

}

void \* StandAloneFreeListPolicy::Allocate(void \* policy, size\_t size)

{

Assert(policy);

StandAloneFreeListPolicy \* \_this = reinterpret\_cast<StandAloneFreeListPolicy \*>(policy);

size\_t index = (size >> ArenaAllocator::ObjectAlignmentBitShift) - 1;

void \* object = NULL;

uint \* freeObjectList = &\_this->freeObjectLists[index];

if (0 != \*freeObjectList)

{

FreeObjectListEntry \* entry = &\_this->entries[\*freeObjectList - 1];

uint oldFreeList = \_this->freeList;

\_this->freeList = \*freeObjectList;

\*freeObjectList = entry->next;

object = entry->object;

Assert(object != NULL);

entry->next = oldFreeList;

entry->object = NULL;

}

return object;

}

void \* StandAloneFreeListPolicy::Free(void \* policy, void \* object, size\_t size)

{

Assert(policy);

StandAloneFreeListPolicy \* \_this = reinterpret\_cast<StandAloneFreeListPolicy \*>(policy);

size\_t index = (size >> ArenaAllocator::ObjectAlignmentBitShift) - 1;

if (TryEnsureFreeListEntry(\_this))

{

Assert(\_this->freeList != 0);

uint \* freeObjectList = &\_this->freeObjectLists[index];

FreeObjectListEntry \* entry = &\_this->entries[\_this->freeList - 1];

uint oldFreeObjectList = \*freeObjectList;

\*freeObjectList = \_this->freeList;

\_this->freeList = entry->next;

entry->object = object;

entry->next = oldFreeObjectList;

}

return \_this;

}

void \* StandAloneFreeListPolicy::Reset(void \* policy)

{

Assert(policy);

StandAloneFreeListPolicy \* \_this = reinterpret\_cast<StandAloneFreeListPolicy \*>(policy);

HeapDeletePlus(GetPlusSize(\_this), \_this);

return NULL;

}

#ifdef ARENA\_MEMORY\_VERIFY

void StandAloneFreeListPolicy::VerifyFreeObjectIsFreeMemFilled(void \* object, size\_t size)

{

char \* bytes = reinterpret\_cast<char\*>(object);

for (size\_t i = 0; i < size; i++)

{

Assert(bytes[i] == StandAloneFreeListPolicy::DbgFreeMemFill);

}

}

#endif

void StandAloneFreeListPolicy::Release(void \* policy)

{

if (NULL != policy)

{

Reset(policy);

}

}

StandAloneFreeListPolicy \* StandAloneFreeListPolicy::NewInternal(uint entries)

{

size\_t plusSize = buckets \* sizeof(uint) + entries \* sizeof(FreeObjectListEntry);

StandAloneFreeListPolicy \* \_this = HeapNewNoThrowPlusZ(plusSize, StandAloneFreeListPolicy);

if (NULL != \_this)

{

\_this->allocated = entries;

\_this->freeObjectLists = (uint \*)(\_this + 1);

\_this->entries = (FreeObjectListEntry \*)(\_this->freeObjectLists + buckets);

}

return \_this;

}

bool StandAloneFreeListPolicy::TryEnsureFreeListEntry(StandAloneFreeListPolicy \*& \_this)

{

if (0 == \_this->freeList)

{

if (\_this->used < \_this->allocated)

{

\_this->used++;

\_this->freeList = \_this->used;

}

else

{

Assert(\_this->used == \_this->allocated);

StandAloneFreeListPolicy \* oldThis = \_this;

uint entries = oldThis->allocated + min(oldThis->allocated, MaxEntriesGrowth);

StandAloneFreeListPolicy \* newThis = NewInternal(entries);

if (NULL != newThis)

{

uint sizeInBytes = buckets \* sizeof(uint);

js\_memcpy\_s(newThis->freeObjectLists, sizeInBytes, oldThis->freeObjectLists, sizeInBytes);

js\_memcpy\_s(newThis->entries, newThis->allocated \* sizeof(FreeObjectListEntry), oldThis->entries, oldThis->used \* sizeof(FreeObjectListEntry));

newThis->used = oldThis->used + 1;

newThis->freeList = newThis->used;

\_this = newThis;

HeapDeletePlus(GetPlusSize(oldThis), oldThis);

}

else

{

return false;

}

}

}

return true;

}

template class ArenaAllocatorBase<StandAloneFreeListPolicy>;

#ifdef PERSISTENT\_INLINE\_CACHES

void \* InlineCacheFreeListPolicy::New(ArenaAllocatorBase<InlineCacheAllocatorTraits> \* allocator)

{

return NewInternal();

}

InlineCacheFreeListPolicy \* InlineCacheFreeListPolicy::NewInternal()

{

InlineCacheFreeListPolicy \* \_this = HeapNewNoThrowZ(InlineCacheFreeListPolicy);

return \_this;

}

InlineCacheFreeListPolicy::InlineCacheFreeListPolicy()

{

Assert(AreFreeListBucketsEmpty());

}

bool InlineCacheFreeListPolicy::AreFreeListBucketsEmpty()

{

for (int b = 0; b < bucketCount; b++)

{

if (this->freeListBuckets[b] != 0) return false;

}

return true;

}

void \* InlineCacheFreeListPolicy::Allocate(void \* policy, size\_t size)

{

Assert(policy);

FreeObject \*\* freeObjectLists = reinterpret\_cast<FreeObject \*\*>(policy);

size\_t index = (size >> InlineCacheAllocatorInfo::ObjectAlignmentBitShift) - 1;

FreeObject \* freeObject = freeObjectLists[index];

if (NULL != freeObject)

{

freeObjectLists[index] = reinterpret\_cast<FreeObject \*>(reinterpret\_cast<intptr>(freeObject->next) & ~InlineCacheFreeListTag);

#ifdef ARENA\_MEMORY\_VERIFY

// Make sure the next pointer bytes are also DbgFreeMemFill-ed, before we give them out.

memset(&freeObject->next, DbgFreeMemFill, sizeof(freeObject->next));

#endif

}

return freeObject;

}

void \* InlineCacheFreeListPolicy::Free(void \* policy, void \* object, size\_t size)

{

Assert(policy);

FreeObject \*\* freeObjectLists = reinterpret\_cast<FreeObject \*\*>(policy);

FreeObject \* freeObject = reinterpret\_cast<FreeObject \*>(object);

size\_t index = (size >> InlineCacheAllocatorInfo::ObjectAlignmentBitShift) - 1;

freeObject->next = reinterpret\_cast<FreeObject \*>(reinterpret\_cast<intptr>(freeObjectLists[index]) | InlineCacheFreeListTag);

freeObjectLists[index] = freeObject;

return policy;

}

void \* InlineCacheFreeListPolicy::Reset(void \* policy)

{

Assert(policy);

InlineCacheFreeListPolicy \* \_this = reinterpret\_cast<InlineCacheFreeListPolicy \*>(policy);

HeapDelete(\_this);

return NULL;

}

#ifdef ARENA\_MEMORY\_VERIFY

void InlineCacheFreeListPolicy::VerifyFreeObjectIsFreeMemFilled(void \* object, size\_t size)

{

unsigned char \* bytes = reinterpret\_cast<unsigned char\*>(object);

for (size\_t i = 0; i < size; i++)

{

// We must allow for zero-filled free listed objects (at least their weakRefs/blankSlots bytes), because during garbage collection, we may zero out

// some of the weakRefs (those that have become unreachable), and this is NOT a sign of "use after free" problem. It would be nice if during collection

// we could reliably distinguish free-listed objects from live caches, but that's not possible because caches can be allocated and freed in batches

// (see more on that in comments inside InlineCacheFreeListPolicy::PrepareFreeObject).

Assert(bytes[i] == NULL || bytes[i] == InlineCacheFreeListPolicy::DbgFreeMemFill);

}

}

#endif

void InlineCacheFreeListPolicy::Release(void \* policy)

{

if (NULL != policy)

{

Reset(policy);

}

}

template class ArenaAllocatorBase<InlineCacheAllocatorTraits>;

#if DBG

bool InlineCacheAllocator::IsAllZero()

{

UpdateCacheBlock();

// See InlineCacheAllocator::ZeroAll for why we ignore the strongRef slot of the CacheLayout.

BigBlock \*bigBlock = this->bigBlocks;

while (bigBlock != NULL)

{

Assert(bigBlock->currentByte % sizeof(CacheLayout) == 0);

CacheLayout\* endPtr = (CacheLayout\*)(bigBlock->GetBytes() + bigBlock->currentByte);

for (CacheLayout\* cache = (CacheLayout\*)bigBlock->GetBytes(); cache < endPtr; cache++)

{

unsigned char\* weakRefBytes = (unsigned char \*)cache->weakRefs;

for (size\_t i = 0; i < sizeof(cache->weakRefs); i++)

{

// If we're verifying arena memory (in debug builds) caches on the free list

// will be debug pattern filled (specifically, at least their weak reference slots).

// All other caches must be zeroed out (again, at least their weak reference slots).

#ifdef ARENA\_MEMORY\_VERIFY

if (weakRefBytes[i] != NULL && weakRefBytes[i] != InlineCacheFreeListPolicy::DbgFreeMemFill)

{

AssertMsg(false, "Inline cache arena is not zeroed!");

return false;

}

#else

if (weakRefBytes[i] != NULL)

{

AssertMsg(false, "Inline cache arena is not zeroed!");

return false;

}

#endif

}

}

bigBlock = bigBlock->nextBigBlock;

}

bigBlock = this->fullBlocks;

while (bigBlock != NULL)

{

Assert(bigBlock->currentByte % sizeof(CacheLayout) == 0);

CacheLayout\* endPtr = (CacheLayout\*)(bigBlock->GetBytes() + bigBlock->currentByte);

for (CacheLayout\* cache = (CacheLayout\*)bigBlock->GetBytes(); cache < endPtr; cache++)

{

char\* weakRefBytes = (char \*)cache->weakRefs;

for (size\_t i = 0; i < sizeof(cache->weakRefs); i++)

{

// If we're verifying arena memory (in debug builds) caches on the free list

// will be debug pattern filled (specifically, their weak reference slots).

// All other caches must be zeroed out (again, their weak reference slots).

#ifdef ARENA\_MEMORY\_VERIFY

if (weakRefBytes[i] != NULL && weakRefBytes[i] != InlineCacheFreeListPolicy::DbgFreeMemFill)

{

AssertMsg(false, "Inline cache arena is not zeroed!");

return false;

}

#else

if (weakRefBytes[i] != NULL)

{

AssertMsg(false, "Inline cache arena is not zeroed!");

return false;

}

#endif

}

}

bigBlock = bigBlock->nextBigBlock;

}

ArenaMemoryBlock \* memoryBlock = this->mallocBlocks;

while (memoryBlock != nullptr)

{

Assert(memoryBlock->nbytes % sizeof(CacheLayout) == 0);

ArenaMemoryBlock \* next = memoryBlock->next;

CacheLayout\* endPtr = (CacheLayout\*)(memoryBlock->GetBytes() + memoryBlock->nbytes);

for (CacheLayout\* cache = (CacheLayout\*)memoryBlock->GetBytes(); cache < endPtr; cache++)

{

unsigned char\* weakRefBytes = (unsigned char \*)cache->weakRefs;

for (size\_t i = 0; i < sizeof(cache->weakRefs); i++)

{

#ifdef ARENA\_MEMORY\_VERIFY

if (weakRefBytes[i] != NULL && weakRefBytes[i] != InlineCacheFreeListPolicy::DbgFreeMemFill)

{

AssertMsg(false, "Inline cache arena is not zeroed!");

return false;

}

#else

if (weakRefBytes[i] != NULL)

{

AssertMsg(false, "Inline cache arena is not zeroed!");

return false;

}

#endif

}

}

memoryBlock = next;

}

return true;

}

#endif

void InlineCacheAllocator::ZeroAll()

{

UpdateCacheBlock();

// We zero the weakRefs part of each cache in the arena unconditionally. The strongRef slot is zeroed only

// if it isn't tagged with InlineCacheFreeListTag. That's so we don't lose our free list, which is

// formed by caches linked via their strongRef slot tagged with InlineCacheFreeListTag. On the other hand,

// inline caches that require invalidation use the same slot as a pointer (untagged) to the cache's address

// in the invalidation list. Hence, we must zero the strongRef slot when untagged to ensure the cache

// doesn't appear registered for invalidation when it's actually blank (which would trigger asserts in InlineCache::VerifyRegistrationForInvalidation).

BigBlock \*bigBlock = this->bigBlocks;

while (bigBlock != NULL)

{

Assert(bigBlock->currentByte % sizeof(CacheLayout) == 0);

CacheLayout\* endPtr = (CacheLayout\*)(bigBlock->GetBytes() + bigBlock->currentByte);

for (CacheLayout\* cache = (CacheLayout\*)bigBlock->GetBytes(); cache < endPtr; cache++)

{

memset(cache->weakRefs, 0, sizeof(cache->weakRefs));

// We want to preserve the free list, whose next pointers are tagged with InlineCacheFreeListTag.

if ((cache->strongRef & InlineCacheFreeListTag) == 0) cache->strongRef = 0;

if (cache->weakRefs[0] != NULL || cache->weakRefs[1] != NULL || cache->weakRefs[2] != NULL)

{

AssertMsg(false, "Inline cache arena is not zeroed!");

}

}

bigBlock = bigBlock->nextBigBlock;

}

bigBlock = this->fullBlocks;

while (bigBlock != NULL)

{

Assert(bigBlock->currentByte % sizeof(CacheLayout) == 0);

CacheLayout\* endPtr = (CacheLayout\*)(bigBlock->GetBytes() + bigBlock->currentByte);

for (CacheLayout\* cache = (CacheLayout\*)bigBlock->GetBytes(); cache < endPtr; cache++)

{

memset(cache->weakRefs, 0, sizeof(cache->weakRefs));

// We want to preserve the free list, whose next pointers are tagged with InlineCacheFreeListTag.

if ((cache->strongRef & InlineCacheFreeListTag) == 0) cache->strongRef = 0;

if (cache->weakRefs[0] != NULL || cache->weakRefs[1] != NULL || cache->weakRefs[2] != NULL)

{

AssertMsg(false, "Inline cache arena is not zeroed!");

}

}

bigBlock = bigBlock->nextBigBlock;

}

ArenaMemoryBlock \* memoryBlock = this->mallocBlocks;

while (memoryBlock != nullptr)

{

Assert(memoryBlock->nbytes % sizeof(CacheLayout) == 0);

ArenaMemoryBlock \* next = memoryBlock->next;

CacheLayout\* endPtr = (CacheLayout\*)(memoryBlock->GetBytes() + memoryBlock->nbytes);

for (CacheLayout\* cache = (CacheLayout\*)memoryBlock->GetBytes(); cache < endPtr; cache++)

{

memset(cache->weakRefs, 0, sizeof(cache->weakRefs));

// We want to preserve the free list, whose next pointers are tagged with InlineCacheFreeListTag.

if ((cache->strongRef & InlineCacheFreeListTag) == 0) cache->strongRef = 0;

if (cache->weakRefs[0] != NULL || cache->weakRefs[1] != NULL || cache->weakRefs[2] != NULL)

{

AssertMsg(false, "Inline cache arena is not zeroed!");

}

}

memoryBlock = next;

}

}

bool InlineCacheAllocator::IsDeadWeakRef(Recycler\* recycler, void\* ptr)

{

return recycler->IsObjectMarked(ptr);

}

bool InlineCacheAllocator::CacheHasDeadWeakRefs(Recycler\* recycler, CacheLayout\* cache)

{

for (intptr\* curWeakRefPtr = cache->weakRefs; curWeakRefPtr < &cache->strongRef; curWeakRefPtr++)

{

intptr curWeakRef = \*curWeakRefPtr;

if (curWeakRef == 0)

{

continue;

}

curWeakRef &= ~(intptr)InlineCacheAuxSlotTypeTag;

if ((curWeakRef & (HeapConstants::ObjectGranularity - 1)) != 0)

{

continue;

}

if (!recycler->IsObjectMarked((void\*)curWeakRef))

{

return true;

}

}

return false;

}

bool InlineCacheAllocator::HasNoDeadWeakRefs(Recycler\* recycler)

{

UpdateCacheBlock();

BigBlock \*bigBlock = this->bigBlocks;

while (bigBlock != NULL)

{

Assert(bigBlock->currentByte % sizeof(CacheLayout) == 0);

CacheLayout\* endPtr = (CacheLayout\*)(bigBlock->GetBytes() + bigBlock->currentByte);

for (CacheLayout\* cache = (CacheLayout\*)bigBlock->GetBytes(); cache < endPtr; cache++)

{

if (CacheHasDeadWeakRefs(recycler, cache))

{

return false;

}

}

bigBlock = bigBlock->nextBigBlock;

}

bigBlock = this->fullBlocks;

while (bigBlock != NULL)

{

Assert(bigBlock->currentByte % sizeof(CacheLayout) == 0);

CacheLayout\* endPtr = (CacheLayout\*)(bigBlock->GetBytes() + bigBlock->currentByte);

for (CacheLayout\* cache = (CacheLayout\*)bigBlock->GetBytes(); cache < endPtr; cache++)

{

if (CacheHasDeadWeakRefs(recycler, cache))

{

return false;

}

}

bigBlock = bigBlock->nextBigBlock;

}

ArenaMemoryBlock \* memoryBlock = this->mallocBlocks;

while (memoryBlock != nullptr)

{

Assert(memoryBlock->nbytes % sizeof(CacheLayout) == 0);

ArenaMemoryBlock \* next = memoryBlock->next;

CacheLayout\* endPtr = (CacheLayout\*)(memoryBlock->GetBytes() + memoryBlock->nbytes);

for (CacheLayout\* cache = (CacheLayout\*)memoryBlock->GetBytes(); cache < endPtr; cache++)

{

if (CacheHasDeadWeakRefs(recycler, cache))

{

return false;

}

}

memoryBlock = next;

}

return true;

}

void InlineCacheAllocator::ClearCacheIfHasDeadWeakRefs(Recycler\* recycler, CacheLayout\* cache)

{

for (intptr\* curWeakRefPtr = cache->weakRefs; curWeakRefPtr < &cache->strongRef; curWeakRefPtr++)

{

intptr curWeakRef = \*curWeakRefPtr;

if (curWeakRef == 0)

{

continue;

}

curWeakRef &= ~(intptr)InlineCacheAuxSlotTypeTag;

if ((curWeakRef & (HeapConstants::ObjectGranularity - 1)) != 0)

{

continue;

}

if (!recycler->IsObjectMarked((void\*)curWeakRef))

{

cache->weakRefs[0] = 0;

cache->weakRefs[1] = 0;

cache->weakRefs[2] = 0;

break;

}

}

}

void InlineCacheAllocator::ClearCachesWithDeadWeakRefs(Recycler\* recycler)

{

UpdateCacheBlock();

BigBlock \*bigBlock = this->bigBlocks;

while (bigBlock != NULL)

{

Assert(bigBlock->currentByte % sizeof(CacheLayout) == 0);

CacheLayout\* endPtr = (CacheLayout\*)(bigBlock->GetBytes() + bigBlock->currentByte);

for (CacheLayout\* cache = (CacheLayout\*)bigBlock->GetBytes(); cache < endPtr; cache++)

{

ClearCacheIfHasDeadWeakRefs(recycler, cache);

}

bigBlock = bigBlock->nextBigBlock;

}

bigBlock = this->fullBlocks;

while (bigBlock != NULL)

{

Assert(bigBlock->currentByte % sizeof(CacheLayout) == 0);

CacheLayout\* endPtr = (CacheLayout\*)(bigBlock->GetBytes() + bigBlock->currentByte);

for (CacheLayout\* cache = (CacheLayout\*)bigBlock->GetBytes(); cache < endPtr; cache++)

{

ClearCacheIfHasDeadWeakRefs(recycler, cache);

}

bigBlock = bigBlock->nextBigBlock;

}

ArenaMemoryBlock \* memoryBlock = this->mallocBlocks;

while (memoryBlock != nullptr)

{

Assert(memoryBlock->nbytes % sizeof(CacheLayout) == 0);

ArenaMemoryBlock \* next = memoryBlock->next;

CacheLayout\* endPtr = (CacheLayout\*)(memoryBlock->GetBytes() + memoryBlock->nbytes);

for (CacheLayout\* cache = (CacheLayout\*)memoryBlock->GetBytes(); cache < endPtr; cache++)

{

ClearCacheIfHasDeadWeakRefs(recycler, cache);

}

memoryBlock = next;

}

}

#else

template class ArenaAllocatorBase<InlineCacheAllocatorTraits>;

#if DBG

bool InlineCacheAllocator::IsAllZero()

{

UpdateCacheBlock();

BigBlock \*blockp = this->bigBlocks;

while (blockp != NULL)

{

for (size\_t i = 0; i < blockp->currentByte; i++)

{

if (blockp->GetBytes()[i] != 0)

{

return false;

}

}

blockp = blockp->nextBigBlock;

}

blockp = this->fullBlocks;

while (blockp != NULL)

{

for (size\_t i = 0; i < blockp->currentByte; i++)

{

if (blockp->GetBytes()[i] != 0)

{

return false;

}

}

blockp = blockp->nextBigBlock;

}

ArenaMemoryBlock \* memoryBlock = this->mallocBlocks;

while (memoryBlock != nullptr)

{

ArenaMemoryBlock \* next = memoryBlock->next;

for (size\_t i = 0; i < memoryBlock->nbytes; i++)

{

if (memoryBlock->GetBytes()[i] != 0)

{

return false;

}

}

memoryBlock = next;

}

return true;

}

#endif

void InlineCacheAllocator::ZeroAll()

{

UpdateCacheBlock();

BigBlock \*blockp = this->bigBlocks;

while (blockp != NULL)

{

memset(blockp->GetBytes(), 0, blockp->currentByte);

blockp = blockp->nextBigBlock;

}

blockp = this->fullBlocks;

while (blockp != NULL)

{

memset(blockp->GetBytes(), 0, blockp->currentByte);

blockp = blockp->nextBigBlock;

}

ArenaMemoryBlock \* memoryBlock = this->mallocBlocks;

while (memoryBlock != nullptr)

{

ArenaMemoryBlock \* next = memoryBlock->next;

memset(memoryBlock->GetBytes(), 0, memoryBlock->nbytes);

memoryBlock = next;

}

}

#endif

template class ArenaAllocatorBase<IsInstInlineCacheAllocatorTraits>;

#if DBG

bool IsInstInlineCacheAllocator::IsAllZero()

{

UpdateCacheBlock();

BigBlock \*blockp = this->bigBlocks;

while (blockp != NULL)

{

for (size\_t i = 0; i < blockp->currentByte; i++)

{

if (blockp->GetBytes()[i] != 0)

{

return false;

}

}

blockp = blockp->nextBigBlock;

}

blockp = this->fullBlocks;

while (blockp != NULL)

{

for (size\_t i = 0; i < blockp->currentByte; i++)

{

if (blockp->GetBytes()[i] != 0)

{

return false;

}

}

blockp = blockp->nextBigBlock;

}

ArenaMemoryBlock \* memoryBlock = this->mallocBlocks;

while (memoryBlock != nullptr)

{

ArenaMemoryBlock \* next = memoryBlock->next;

for (size\_t i = 0; i < memoryBlock->nbytes; i++)

{

if (memoryBlock->GetBytes()[i] != 0)

{

return false;

}

}

memoryBlock = next;

}

return true;

}

#endif

void IsInstInlineCacheAllocator::ZeroAll()

{

UpdateCacheBlock();

BigBlock \*blockp = this->bigBlocks;

while (blockp != NULL)

{

memset(blockp->GetBytes(), 0, blockp->currentByte);

blockp = blockp->nextBigBlock;

}

blockp = this->fullBlocks;

while (blockp != NULL)

{

memset(blockp->GetBytes(), 0, blockp->currentByte);

blockp = blockp->nextBigBlock;

}

ArenaMemoryBlock \* memoryBlock = this->mallocBlocks;

while (memoryBlock != nullptr)

{

ArenaMemoryBlock \* next = memoryBlock->next;

memset(memoryBlock->GetBytes(), 0, memoryBlock->nbytes);

memoryBlock = next;

}

}

#undef ASSERT\_TRHEAD

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#ifdef PROFILE\_MEM

struct ArenaMemoryData;

#endif

namespace Memory

{

// Arena allocator

#define Anew(alloc,T,...) AllocatorNew(ArenaAllocator, alloc, T, \_\_VA\_ARGS\_\_)

#define AnewZ(alloc,T,...) AllocatorNewZ(ArenaAllocator, alloc, T, \_\_VA\_ARGS\_\_)

#define AnewPlus(alloc, size, T, ...) AllocatorNewPlus(ArenaAllocator, alloc, size, T, \_\_VA\_ARGS\_\_)

#define AnewPlusZ(alloc, size, T, ...) AllocatorNewPlusZ(ArenaAllocator, alloc, size, T, \_\_VA\_ARGS\_\_)

#define AnewStruct(alloc,T) AllocatorNewStruct(ArenaAllocator, alloc, T)

#define AnewStructZ(alloc,T) AllocatorNewStructZ(ArenaAllocator, alloc, T)

#define AnewStructPlus(alloc, size, T) AllocatorNewStructPlus(ArenaAllocator, alloc, size, T)

#define AnewArray(alloc, T, count) AllocatorNewArray(ArenaAllocator, alloc, T, count)

#define AnewArrayZ(alloc, T, count) AllocatorNewArrayZ(ArenaAllocator, alloc, T, count)

#define Adelete(alloc, obj) AllocatorDelete(ArenaAllocator, alloc, obj)

#define AdeletePlus(alloc, size, obj) AllocatorDeletePlus(ArenaAllocator, alloc, size, obj)

#define AdeleteArray(alloc, count, obj) AllocatorDeleteArray(ArenaAllocator, alloc, count, obj)

#define AnewNoThrow(alloc,T,...) AllocatorNewNoThrow(ArenaAllocator, alloc, T, \_\_VA\_ARGS\_\_)

#define AnewNoThrowZ(alloc,T,...) AllocatorNewNoThrowZ(ArenaAllocator, alloc, T, \_\_VA\_ARGS\_\_)

#define AnewNoThrowPlus(alloc, size, T, ...) AllocatorNewNoThrowPlus(ArenaAllocator, alloc, size, T, \_\_VA\_ARGS\_\_)

#define AnewNoThrowPlusZ(alloc, size, T, ...) AllocatorNewNoThrowPlusZ(ArenaAllocator, alloc, size, T, \_\_VA\_ARGS\_\_)

#define AnewNoThrowStruct(alloc,T) AllocatorNewNoThrowStruct(ArenaAllocator, alloc, T)

#define AnewNoThrowStructZ(alloc,T) AllocatorNewNoThrowStructZ(ArenaAllocator, alloc, T)

#define AnewNoThrowArray(alloc, T, count) AllocatorNewNoThrowArray(ArenaAllocator, alloc, T, count)

#define AnewNoThrowArrayZ(alloc, T, count) AllocatorNewNoThrowArrayZ(ArenaAllocator, alloc, T, count)

#define JitAnew(alloc,T,...) AllocatorNew(JitArenaAllocator, alloc, T, \_\_VA\_ARGS\_\_)

#define JitAnewZ(alloc,T,...) AllocatorNewZ(JitArenaAllocator, alloc, T, \_\_VA\_ARGS\_\_)

#define JitAnewPlus(alloc, size, T, ...) AllocatorNewPlus(JitArenaAllocator, alloc, size, T, \_\_VA\_ARGS\_\_)

#define JitAnewPlusZ(alloc, size, T, ...) AllocatorNewPlusZ(JitArenaAllocator, alloc, size, T, \_\_VA\_ARGS\_\_)

#define JitAnewStruct(alloc,T) AllocatorNewStruct(JitArenaAllocator, alloc, T)

#define JitAnewStructZ(alloc,T) AllocatorNewStructZ(JitArenaAllocator, alloc, T)

#define JitAnewStructPlus(alloc, size, T) AllocatorNewStructPlus(JitArenaAllocator, alloc, size, T)

#define JitAnewArray(alloc, T, count) AllocatorNewArray(JitArenaAllocator, alloc, T, count)

#define JitAnewArrayZ(alloc, T, count) AllocatorNewArrayZ(JitArenaAllocator, alloc, T, count)

#define JitAdelete(alloc, obj) AllocatorDelete(JitArenaAllocator, alloc, obj)

#define JitAdeletePlus(alloc, size, obj) AllocatorDeletePlus(JitArenaAllocator, alloc, size, obj)

#define JitAdeleteArray(alloc, count, obj) AllocatorDeleteArray(JitArenaAllocator, alloc, count, obj)

#define JitAnewNoThrow(alloc,T,...) AllocatorNewNoThrow(JitArenaAllocator, alloc, T, \_\_VA\_ARGS\_\_)

#define JitAnewNoThrowZ(alloc,T,...) AllocatorNewNoThrowZ(JitArenaAllocator, alloc, T, \_\_VA\_ARGS\_\_)

#define JitAnewNoThrowPlus(alloc, size, T, ...) AllocatorNewNoThrowPlus(JitArenaAllocator, alloc, size, T, \_\_VA\_ARGS\_\_)

#define JitAnewNoThrowPlusZ(alloc, size, T, ...) AllocatorNewNoThrowPlusZ(JitArenaAllocator, alloc, size, T, \_\_VA\_ARGS\_\_)

#define JitAnewNoThrowStruct(alloc,T) AllocatorNewNoThrowStruct(JitArenaAllocator, alloc, T)

#define JitAnewNoThrowArray(alloc, T, count) AllocatorNewNoThrowArray(JitArenaAllocator, alloc, T, count)

#define JitAnewNoThrowArrayZ(alloc, T, count) AllocatorNewNoThrowArrayZ(JitArenaAllocator, alloc, T, count)

struct BigBlock;

struct ArenaMemoryBlock

{

union

{

ArenaMemoryBlock \* next;

BigBlock \* nextBigBlock;

};

size\_t nbytes;

char \* GetBytes() const

{

return ((char \*)this) + sizeof(ArenaMemoryBlock);

}

};

struct BigBlock : public ArenaMemoryBlock

{

public:

PageAllocation \* allocation;

size\_t currentByte;

char \* GetBytes() const

{

return ((char \*)this) + sizeof(BigBlock);

}

};

#define ASSERT\_THREAD() AssertMsg(this->pageAllocator->ValidThreadAccess(), "Arena allocation should only be used by a single thread")

// Basic data layout of arena allocators. This data should be all that is needed

// to perform operations that traverse all allocated memory in the arena:

// the recycler used this to mark through registered arenas and inline cache

// allocator uses this to zero out allocated memory.

class ArenaData

{

protected:

ArenaData(PageAllocator \* pageAllocator);

protected:

BigBlock \* bigBlocks;

BigBlock \* fullBlocks;

ArenaMemoryBlock \* mallocBlocks;

PageAllocator \* pageAllocator;

char \* cacheBlockCurrent;

bool lockBlockList;

public:

BigBlock\* GetBigBlocks(bool background)

{

if (!background)

{

UpdateCacheBlock();

}

return bigBlocks;

}

BigBlock\* GetFullBlocks() { return fullBlocks; }

ArenaMemoryBlock \* GetMemoryBlocks() { return mallocBlocks; }

PageAllocator \* GetPageAllocator() const

{

return pageAllocator;

}

bool IsBlockListLocked() { return lockBlockList; }

void SetLockBlockList(bool lock) { lockBlockList = lock; }

protected:

void UpdateCacheBlock() const;

};

// Implements most of memory management operations over ArenaData.

// The TFreeListPolicy handles free-listing for "small objects". There

// is no support for free-listing for "large objects".

#if defined(\_M\_X64\_OR\_ARM64)

// Some data structures such as jmp\_buf expect to be 16 byte aligned on AMD64.

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg = 4, bool RequireObjectAlignment = false, size\_t MaxObjectSize = 0>

#else

template <class TFreeListPolicy, size\_t ObjectAlignmentBitShiftArg = 3, bool RequireObjectAlignment = false, size\_t MaxObjectSize = 0>

#endif

class ArenaAllocatorBase : public Allocator, public ArenaData

{

private:

char \* cacheBlockEnd;

size\_t largestHole;

uint blockState; // 0 = no block, 1 = one big block, other more then one big block or have malloc blocks

#ifdef PROFILE\_MEM

LPCWSTR name;

#endif

#ifdef PROFILE\_MEM

struct ArenaMemoryData \* memoryData;

#endif

public:

static const size\_t ObjectAlignmentBitShift = ObjectAlignmentBitShiftArg;

static const size\_t ObjectAlignment = 1 << ObjectAlignmentBitShift;

static const size\_t ObjectAlignmentMask = ObjectAlignment - 1;

static const bool FakeZeroLengthArray = true;

static const size\_t MaxSmallObjectSize = 1024;

ArenaAllocatorBase(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void (\*outOfMemoryFunc)(), void (\*recoverMemoryFunc)() = JsUtil::ExternalApi::RecoverUnusedMemory);

~ArenaAllocatorBase();

void Reset()

{

ASSERT\_THREAD();

Assert(!lockBlockList);

freeList = TFreeListPolicy::Reset(freeList);

#ifdef PROFILE\_MEM

LogReset();

#endif

ArenaMemoryTracking::ReportFreeAll(this);

if (this->blockState == 1)

{

Assert(this->bigBlocks != nullptr && this->fullBlocks == nullptr && this->mallocBlocks == nullptr && this->bigBlocks->nextBigBlock == nullptr);

Assert(this->largestHole == 0);

Assert(cacheBlockEnd == bigBlocks->GetBytes() + bigBlocks->nbytes);

Assert(bigBlocks->GetBytes() <= cacheBlockCurrent && cacheBlockCurrent <= cacheBlockEnd);

cacheBlockCurrent = bigBlocks->GetBytes();

#ifdef PROFILE\_MEM

LogRealAlloc(bigBlocks->allocation->GetSize() + sizeof(PageAllocation));

#endif

return;

}

FullReset();

}

void Move(ArenaAllocatorBase \*srcAllocator);

void Clear()

{

ASSERT\_THREAD();

Assert(!lockBlockList);

ArenaMemoryTracking::ReportFreeAll(this);

freeList = TFreeListPolicy::Reset(freeList);

#ifdef ARENA\_ALLOCATOR\_FREE\_LIST\_SIZE

this->freeListSize = 0;

#endif

ReleaseMemory();

this->cacheBlockCurrent = nullptr;

this->cacheBlockEnd = nullptr;

this->bigBlocks = nullptr;

this->fullBlocks = nullptr;

this->largestHole = 0;

this->mallocBlocks = nullptr;

this->blockState = 0;

}

size\_t AllocatedSize(); // amount of memory allocated

size\_t Size(); // amount of allocated memory is used.

size\_t FreeListSize()

{

#ifdef ARENA\_ALLOCATOR\_FREE\_LIST\_SIZE

return this->freeListSize;

#else

return 0;

#endif

}

static size\_t GetAlignedSize(size\_t size) { return AllocSizeMath::Align(size, ArenaAllocatorBase::ObjectAlignment); }

char \* AllocInternal(size\_t requestedBytes);

char\* Realloc(void\* buffer, size\_t existingBytes, size\_t requestedBytes);

void Free(void \* buffer, size\_t byteSize);

#ifdef TRACK\_ALLOC

// Doesn't support tracking information, dummy implementation

ArenaAllocatorBase \* TrackAllocInfo(TrackAllocData const& data) { return this; }

void ClearTrackAllocInfo(TrackAllocData\* data = nullptr) {}

#endif

protected:

char \* RealAlloc(size\_t nbytes);

\_\_forceinline char \* RealAllocInlined(size\_t nbytes);

private:

#ifdef PROFILE\_MEM

void LogBegin();

void LogReset();

void LogEnd();

void LogAlloc(size\_t requestedBytes, size\_t allocateBytes);

void LogRealAlloc(size\_t size);

#endif

static size\_t AllocatedSize(ArenaMemoryBlock \* blockList);

static size\_t Size(BigBlock \* blockList);

void FullReset();

void SetCacheBlock(BigBlock \* cacheBlock);

template <bool DoRecoverMemory> char \* AllocFromHeap(size\_t nbytes);

void ReleaseMemory();

void ReleasePageMemory();

void ReleaseHeapMemory();

char \* SnailAlloc(size\_t nbytes);

BigBlock \* AddBigBlock(size\_t pages);

#ifdef ARENA\_ALLOCATOR\_FREE\_LIST\_SIZE

size\_t freeListSize;

#endif

void \* freeList;

#ifdef PROFILE\_MEM

void LogFree(size\_t size);

void LogReuse(size\_t size);

#endif

};

// Implements free-listing in-place. Bucketizes allocation sizes, there is a free list

// per bucket. The freed memory is treated as nodes in the lists.

class InPlaceFreeListPolicy

{

private:

// Free list support;

struct FreeObject

{

FreeObject \* next;

};

private:

static const uint buckets =

ArenaAllocatorBase<InPlaceFreeListPolicy>::MaxSmallObjectSize >> ArenaAllocatorBase<InPlaceFreeListPolicy>::ObjectAlignmentBitShift;

public:

#ifdef DBG

static const unsigned char DbgFreeMemFill = DbgMemFill;

#endif

static void \* New(ArenaAllocatorBase<InPlaceFreeListPolicy> \* allocator);

static void \* Allocate(void \* policy, size\_t size);

static void \* Free(void \* policy, void \* object, size\_t size);

static void \* Reset(void \* policy);

static void PrepareFreeObject(\_\_out\_bcount(size) void \* object, \_In\_ size\_t size)

{

#ifdef ARENA\_MEMORY\_VERIFY

memset(object, InPlaceFreeListPolicy::DbgFreeMemFill, size);

#endif

}

#ifdef ARENA\_MEMORY\_VERIFY

static void VerifyFreeObjectIsFreeMemFilled(void \* object, size\_t size);

#endif

static void Release(void \* policy) {}

};

// Implements free-listing in separate memory. Bucketizes allocation sizes, there is a free list

// per bucket. Space for the free lists is allocated from the heap. This is used by

// InlineCacheAllocator to quickly zero out the entire arena w/o loosing the free lists.

class StandAloneFreeListPolicy

{

private:

struct FreeObjectListEntry

{

void \* object;

uint next;

};

uint allocated;

uint used;

uint freeList;

uint\* freeObjectLists;

FreeObjectListEntry\* entries;

static const uint buckets =

ArenaAllocatorBase<StandAloneFreeListPolicy>::MaxSmallObjectSize >> ArenaAllocatorBase<StandAloneFreeListPolicy>::ObjectAlignmentBitShift;

static const uint InitialEntries = 64;

static const uint MaxEntriesGrowth = 1024;

static StandAloneFreeListPolicy \* NewInternal(uint entriesPerBucket);

static bool TryEnsureFreeListEntry(StandAloneFreeListPolicy \*& \_this);

static uint GetPlusSize(const StandAloneFreeListPolicy \* policy)

{

return buckets \* sizeof(uint) + policy->allocated \* sizeof(FreeObjectListEntry);

}

public:

#ifdef DBG

// TODO: Consider making DbgFreeMemFill == DbgFill, now that we have ArenaAllocatorBase properly handling filling when free-listing and asserting debug fill at re-allocation.

static const char DbgFreeMemFill = 0x0;

#endif

static void \* New(ArenaAllocatorBase<StandAloneFreeListPolicy> \* allocator);

static void \* Allocate(void \* policy, size\_t size);

static void \* Free(void \* policy, void \* object, size\_t size);

static void \* Reset(void \* policy);

static void PrepareFreeObject(\_Out\_writes\_bytes\_all\_(size) void \* object, \_In\_ size\_t size)

{

#ifdef ARENA\_MEMORY\_VERIFY

memset(object, StandAloneFreeListPolicy::DbgFreeMemFill, size);

#endif

}

#ifdef ARENA\_MEMORY\_VERIFY

static void VerifyFreeObjectIsFreeMemFilled(void \* object, size\_t size);

#endif

static void Release(void \* policy);

};

#define ARENA\_FAULTINJECT\_MEMORY(name, size) { \

if (outOfMemoryFunc) \

{ \

FAULTINJECT\_MEMORY\_THROW(name, size); \

} \

else \

{ \

FAULTINJECT\_MEMORY\_NOTHROW(name, size); \

} \

}

// This allocator by default on OOM makes an attempt to recover memory from Recycler and further throws if that doesn't help the allocation.

class ArenaAllocator : public ArenaAllocatorBase<InPlaceFreeListPolicy>

{

public:

ArenaAllocator(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void (\*outOfMemoryFunc)(), void (\*recoverMemoryFunc)() = JsUtil::ExternalApi::RecoverUnusedMemory) :

ArenaAllocatorBase<InPlaceFreeListPolicy>(name, pageAllocator, outOfMemoryFunc, recoverMemoryFunc)

{

}

\_\_forceinline

char \* Alloc(size\_t requestedBytes)

{

return AllocInternal(requestedBytes);

}

char \* AllocZero(size\_t nbytes)

{

char \* buffer = Alloc(nbytes);

memset(buffer, 0, nbytes);

#if DBG

// Since we successfully allocated, we shouldn't have integer overflow here

memset(buffer + nbytes, 0, Math::Align(nbytes, ArenaAllocatorBase::ObjectAlignment) - nbytes);

#endif

return buffer;

}

char \* AllocLeaf(size\_t requestedBytes)

{

// Leaf allocation is not meaningful here, but needed by Allocator-templatized classes that may call one of the Leaf versions of AllocatorNew

return Alloc(requestedBytes);

}

char \* NoThrowAlloc(size\_t requestedBytes)

{

void (\*tempOutOfMemoryFunc)() = outOfMemoryFunc;

outOfMemoryFunc = nullptr;

char \* buffer = AllocInternal(requestedBytes);

outOfMemoryFunc = tempOutOfMemoryFunc;

return buffer;

}

char \* NoThrowAllocZero(size\_t requestedBytes)

{

char \* buffer = NoThrowAlloc(requestedBytes);

if (buffer != nullptr)

{

memset(buffer, 0, requestedBytes);

}

return buffer;

}

char \* NoThrowNoRecoveryAlloc(size\_t requestedBytes)

{

void (\*tempRecoverMemoryFunc)() = recoverMemoryFunc;

recoverMemoryFunc = nullptr;

char \* buffer = NoThrowAlloc(requestedBytes);

recoverMemoryFunc = tempRecoverMemoryFunc;

return buffer;

}

char \* NoThrowNoRecoveryAllocZero(size\_t requestedBytes)

{

char \* buffer = NoThrowNoRecoveryAlloc(requestedBytes);

if (buffer != nullptr)

{

memset(buffer, 0, requestedBytes);

}

return buffer;

}

};

class JitArenaAllocator : public ArenaAllocator

{

// The only difference between ArenaAllocator and the JitArenaAllocator is it has fast path of anything of size BVSparseNode (16 bytes)

// Throughput improvement in the backend is substantial with this freeList.

private:

BVSparseNode \*bvFreeList;

public:

JitArenaAllocator(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void(\*outOfMemoryFunc)(), void(\*recoverMemoryFunc)() = JsUtil::ExternalApi::RecoverUnusedMemory) :

bvFreeList(nullptr), ArenaAllocator(name, pageAllocator, outOfMemoryFunc, recoverMemoryFunc)

{

}

char \* Alloc(size\_t requestedBytes)

{

// Fast path

if (sizeof(BVSparseNode) == requestedBytes)

{

AssertMsg(Math::Align(requestedBytes, ArenaAllocatorBase::ObjectAlignment) == requestedBytes, "Assert for Perf, T should always be aligned");

// Fast path for BVSparseNode allocation

if (bvFreeList)

{

BVSparseNode \*node = bvFreeList;

bvFreeList = bvFreeList->next;

return (char\*)node;

}

// If the free list is empty, then do the allocation right away for the BVSparseNode size.

// You could call ArenaAllocator::Alloc here, but direct RealAlloc avoids unnecessary checks.

return ArenaAllocatorBase::RealAllocInlined(requestedBytes);

}

return ArenaAllocator::Alloc(requestedBytes);

}

void Free(void \* buffer, size\_t byteSize)

{

return FreeInline(buffer, byteSize);

}

\_\_forceinline void FreeInline(void \* buffer, size\_t byteSize)

{

if (sizeof(BVSparseNode) == byteSize)

{

//FastPath

((BVSparseNode\*)buffer)->next = bvFreeList;

bvFreeList = (BVSparseNode\*)buffer;

return;

}

return ArenaAllocator::Free(buffer, byteSize);

}

char \* AllocZero(size\_t nbytes)

{

return ArenaAllocator::AllocZero(nbytes);

}

char \* AllocLeaf(size\_t requestedBytes)

{

return ArenaAllocator::AllocLeaf(requestedBytes);

}

char \* NoThrowAlloc(size\_t requestedBytes)

{

return ArenaAllocator::NoThrowAlloc(requestedBytes);

}

char \* NoThrowAllocZero(size\_t requestedBytes)

{

return ArenaAllocator::NoThrowAllocZero(requestedBytes);

}

void Reset()

{

bvFreeList = nullptr;

ArenaAllocator::Reset();

}

void Clear()

{

bvFreeList = nullptr;

ArenaAllocator::Clear();

}

};

// This allocator by default on OOM does not attempt to recover memory from Recycler, just throws OOM.

class NoRecoverMemoryJitArenaAllocator : public JitArenaAllocator

{

public:

NoRecoverMemoryJitArenaAllocator(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void(\*outOfMemoryFunc)()) :

JitArenaAllocator(name, pageAllocator, outOfMemoryFunc, NULL)

{

}

};

// This allocator by default on OOM does not attempt to recover memory from Recycler, just throws OOM.

class NoRecoverMemoryArenaAllocator : public ArenaAllocator

{

public:

NoRecoverMemoryArenaAllocator(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void (\*outOfMemoryFunc)()) :

ArenaAllocator(name, pageAllocator, outOfMemoryFunc, NULL)

{

}

};

#define InlineCacheAuxSlotTypeTag 4

#define MinPolymorphicInlineCacheSize 4

#define MaxPolymorphicInlineCacheSize 32

#ifdef PERSISTENT\_INLINE\_CACHES

class InlineCacheAllocatorInfo

{

public:

struct CacheLayout

{

intptr weakRefs[3];

intptr strongRef;

};

struct FreeObject

{

intptr blankSlots[3];

FreeObject \* next;

};

CompileAssert(sizeof(CacheLayout) == sizeof(FreeObject));

CompileAssert(offsetof(CacheLayout, strongRef) == offsetof(FreeObject, next));

#if defined(\_M\_X64\_OR\_ARM64)

CompileAssert(sizeof(CacheLayout) == 32);

static const size\_t ObjectAlignmentBitShift = 5;

#else

CompileAssert(sizeof(CacheLayout) == 16);

static const size\_t ObjectAlignmentBitShift = 4;

#endif

static const size\_t ObjectAlignment = 1 << ObjectAlignmentBitShift;

static const size\_t MaxObjectSize = MaxPolymorphicInlineCacheSize \* sizeof(CacheLayout);

};

#define InlineCacheFreeListTag 0x01

#define InlineCacheAllocatorTraits InlineCacheFreeListPolicy, InlineCacheAllocatorInfo::ObjectAlignmentBitShift, true, InlineCacheAllocatorInfo::MaxObjectSize

class InlineCacheFreeListPolicy : public InlineCacheAllocatorInfo

{

public:

#ifdef DBG

static const unsigned char DbgFreeMemFill = DbgMemFill;

#endif

static void \* New(ArenaAllocatorBase<InlineCacheAllocatorTraits> \* allocator);

static void \* Allocate(void \* policy, size\_t size);

static void \* Free(void \* policy, void \* object, size\_t size);

static void \* Reset(void \* policy);

static void Release(void \* policy);

static void PrepareFreeObject(\_Out\_writes\_bytes\_all\_(size) void \* object, \_In\_ size\_t size)

{

#ifdef ARENA\_MEMORY\_VERIFY

// In debug builds if we're verifying arena memory to avoid "use after free" problems, we want to fill the whole object with the debug pattern here.

// There is a very subtle point here. Inline caches can be allocated and freed in batches. This happens commonly when a PolymorphicInlineCache grows,

// frees up its old array of inline caches, and allocates a bigger one. ArenaAllocatorBase::AllocInternal when allocating an object from the free list

// will verify that the entire object - not just its first sizeof(InlineCache) worth of bytes - is filled with the debug pattern.

memset(object, StandAloneFreeListPolicy::DbgFreeMemFill, size);

#else

// On the other hand, in retail builds when we don't do arena memory validation we want to zero out the whole object, so that during every subsequent garbage collection

// we don't try to trace pointers from freed objects (inside ClearCacheIfHasDeadWeakRefs) and check if they are still reachable. Note that in ClearCacheIfHasDeadWeakRefs

// we cannot distinguish between live inline caches and portions of free objects. That's again because inline caches may be allocated and freed in batches, in which case

// only the first cache in the batch gets the free object's next pointer tag. The rest of the batch is indistinguishable from a batch of live caches. Hence, we scan them

// all for pointers to unreachable objects, and it makes good sense to zero these bytes out, to avoid unnecessary Recycler::IsObjectMarked calls.

memset(object, NULL, size);

#endif

}

#ifdef ARENA\_MEMORY\_VERIFY

static void VerifyFreeObjectIsFreeMemFilled(void \* object, size\_t size);

#endif

private:

static const uint bucketCount = MaxObjectSize >> ObjectAlignmentBitShift;

FreeObject\* freeListBuckets[bucketCount];

static InlineCacheFreeListPolicy \* NewInternal();

InlineCacheFreeListPolicy();

bool AreFreeListBucketsEmpty();

};

class InlineCacheAllocator : public InlineCacheAllocatorInfo, public ArenaAllocatorBase<InlineCacheAllocatorTraits>

{

#ifdef POLY\_INLINE\_CACHE\_SIZE\_STATS

private:

size\_t polyCacheAllocSize;

#endif

public:

// Zeroing and freeing w/o leaking is not implemented for large objects

CompileAssert(MaxObjectSize <= MaxSmallObjectSize);

InlineCacheAllocator(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void(\*outOfMemoryFunc)(), void(\*recoverMemoryFunc)() = JsUtil::ExternalApi::RecoverUnusedMemory) :

ArenaAllocatorBase<InlineCacheAllocatorTraits>(name, pageAllocator, outOfMemoryFunc, recoverMemoryFunc)

#ifdef POLY\_INLINE\_CACHE\_SIZE\_STATS

, polyCacheAllocSize(0)

#endif

{}

char \* Alloc(size\_t requestedBytes)

{

return AllocInternal(requestedBytes);

}

char \* AllocZero(size\_t nbytes)

{

char \* buffer = Alloc(nbytes);

memset(buffer, 0, nbytes);

#if DBG

// Since we successfully allocated, we shouldn't have integer overflow here

memset(buffer + nbytes, 0, Math::Align(nbytes, ArenaAllocatorBase::ObjectAlignment) - nbytes);

#endif

return buffer;

}

#if DBG

bool IsAllZero();

#endif

void ZeroAll();

bool IsDeadWeakRef(Recycler\* recycler, void\* ptr);

bool CacheHasDeadWeakRefs(Recycler\* recycler, CacheLayout\* cache);

bool HasNoDeadWeakRefs(Recycler\* recycler);

void ClearCacheIfHasDeadWeakRefs(Recycler\* recycler, CacheLayout\* cache);

void ClearCachesWithDeadWeakRefs(Recycler\* recycler);

#ifdef POLY\_INLINE\_CACHE\_SIZE\_STATS

size\_t GetPolyInlineCacheSize() { return this->polyCacheAllocSize; }

void LogPolyCacheAlloc(size\_t size) { this->polyCacheAllocSize += size; }

void LogPolyCacheFree(size\_t size) { this->polyCacheAllocSize -= size; }

#endif

};

#else

#define InlineCacheAllocatorTraits StandAloneFreeListPolicy, InlineCacheAllocatorInfo::ObjectAlignmentBitShift, true, InlineCacheAllocatorInfo::MaxObjectSize

class InlineCacheAllocator : public ArenaAllocatorBase<InlineCacheAllocatorTraits>

{

public:

struct CacheLayout

{

intptr weakRefs[3];

intptr strongRef;

};

#ifdef POLY\_INLINE\_CACHE\_SIZE\_STATS

private:

size\_t polyCacheAllocSize;

#endif

public:

InlineCacheAllocator(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void(\*outOfMemoryFunc)()) :

ArenaAllocatorBase<InlineCacheAllocatorTraits>(name, pageAllocator, outOfMemoryFunc) {}

char \* Alloc(size\_t requestedBytes)

{

return AllocInternal(requestedBytes);

}

char \* AllocZero(size\_t nbytes)

{

char \* buffer = Alloc(nbytes);

memset(buffer, 0, nbytes);

#if DBG

// Since we successfully allocated, we shouldn't have integer overflow here

memset(buffer + nbytes, 0, Math::Align(nbytes, ArenaAllocatorBase::ObjectAlignment) - nbytes);

#endif

return buffer;

}

#if DBG

bool IsAllZero();

#endif

void ZeroAll();

#ifdef POLY\_INLINE\_CACHE\_SIZE\_STATS

size\_t GetPolyInlineCacheSize() { return this->polyCacheAllocSize; }

void LogPolyCacheAlloc(size\_t size) { this->polyCacheAllocSize += size; }

void LogPolyCacheFree(size\_t size) { this->polyCacheAllocSize -= size; }

#endif

};

#endif

class IsInstInlineCacheAllocatorInfo

{

public:

struct CacheLayout

{

char bytes[4 \* sizeof(intptr)];

};

#if \_M\_X64 || \_M\_ARM64

CompileAssert(sizeof(CacheLayout) == 32);

static const size\_t ObjectAlignmentBitShift = 5;

#else

CompileAssert(sizeof(CacheLayout) == 16);

static const size\_t ObjectAlignmentBitShift = 4;

#endif

static const size\_t ObjectAlignment = 1 << ObjectAlignmentBitShift;

static const size\_t MaxObjectSize = sizeof(CacheLayout);

};

#define IsInstInlineCacheAllocatorTraits StandAloneFreeListPolicy

class IsInstInlineCacheAllocator : public IsInstInlineCacheAllocatorInfo, public ArenaAllocatorBase<IsInstInlineCacheAllocatorTraits>

{

#ifdef POLY\_INLINE\_CACHE\_SIZE\_STATS

private:

size\_t polyCacheAllocSize;

#endif

public:

IsInstInlineCacheAllocator(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void(\*outOfMemoryFunc)()) :

ArenaAllocatorBase<IsInstInlineCacheAllocatorTraits>(name, pageAllocator, outOfMemoryFunc) {}

char \* Alloc(size\_t requestedBytes)

{

return AllocInternal(requestedBytes);

}

char \* AllocZero(size\_t nbytes)

{

char \* buffer = Alloc(nbytes);

memset(buffer, 0, nbytes);

#if DBG

// Since we successfully allocated, we shouldn't have integer overflow here

memset(buffer + nbytes, 0, Math::Align(nbytes, ArenaAllocatorBase::ObjectAlignment) - nbytes);

#endif

return buffer;

}

#if DBG

bool IsAllZero();

#endif

void ZeroAll();

#ifdef POLY\_INLINE\_CACHE\_SIZE\_STATS

size\_t GetPolyInlineCacheSize() { return this->polyCacheAllocSize; }

void LogPolyCacheAlloc(size\_t size) { this->polyCacheAllocSize += size; }

void LogPolyCacheFree(size\_t size) { this->polyCacheAllocSize -= size; }

#endif

};

#undef ASSERT\_THREAD

class RefCounted

{

volatile long refCount;

protected:

virtual ~RefCounted()

{

}

public:

RefCounted()

: refCount(1)

{

}

ulong AddRef(void)

{

return (ulong)InterlockedIncrement(&refCount);

}

ulong Release(void)

{

ulong refs = (ulong)InterlockedDecrement(&refCount);

if (0 == refs)

{

delete this;

}

return refs;

}

};

class ReferencedArenaAdapter;

template<class T>

class WeakArenaReference

{

ReferencedArenaAdapter\* adapter;

T\* p;

public:

WeakArenaReference(ReferencedArenaAdapter\* \_adapter,T\* \_p)

: adapter(\_adapter),

p(\_p)

{

adapter->AddRef();

}

~WeakArenaReference()

{

adapter->Release();

adapter = NULL;

}

T\* GetStrongReference()

{

if(adapter->AddStrongReference())

{

return p;

}

else

{

return NULL;

}

}

void ReleaseStrongReference()

{

adapter->ReleaseStrongReference();

}

};

// This class enables WeakArenaReferences to track whether

// the arena has been deleted, and allows for extending

// the lifetime of the arena for StrongReferences

// Strong references should be short lived

class ReferencedArenaAdapter : public RefCounted

{

CRITICAL\_SECTION adapterLock;

ulong strongRefCount;

ArenaAllocator\* arena;

bool deleteFlag;

public:

~ReferencedArenaAdapter()

{

if (this->arena)

{

HeapDelete(this->arena);

}

DeleteCriticalSection(&adapterLock);

}

ReferencedArenaAdapter(ArenaAllocator\* \_arena)

: RefCounted(),

strongRefCount(0),

arena(\_arena),

deleteFlag(false)

{

InitializeCriticalSection(&adapterLock);

}

bool AddStrongReference()

{

EnterCriticalSection(&adapterLock);

if (deleteFlag)

{

// Arena exists and is marked deleted, we must fail to acquire a new reference

if (arena && 0 == strongRefCount)

{

// All strong references are gone, delete the arena

HeapDelete(this->arena);

this->arena = nullptr;

}

LeaveCriticalSection(&adapterLock);

return false;

}

else

{

// Succeed at acquiring a Strong Reference into the Arena

strongRefCount++;

LeaveCriticalSection(&adapterLock);

return true;

}

}

void ReleaseStrongReference()

{

EnterCriticalSection(&adapterLock);

strongRefCount--;

if (deleteFlag && this->arena && 0 == strongRefCount)

{

// All strong references are gone, delete the arena

HeapDelete(this->arena);

this->arena = NULL;

}

LeaveCriticalSection(&adapterLock);

}

void DeleteArena()

{

deleteFlag = true;

if (TryEnterCriticalSection(&adapterLock))

{

if (0 == strongRefCount)

{

// All strong references are gone, delete the arena

HeapDelete(this->arena);

this->arena = NULL;

}

LeaveCriticalSection(&adapterLock);

}

}

ArenaAllocator\* Arena()

{

if (!deleteFlag)

{

return this->arena;

}

return NULL;

}

};

}

//we don't need these for the ArenaAllocator

#if 0

inline void \_\_cdecl

operator delete(void \* obj, ArenaAllocator \* alloc, char \* (ArenaAllocator::\*AllocFunc)(size\_t))

{

alloc->Free(obj, (size\_t)-1);

}

inline void \_\_cdecl

operator delete(void \* obj, ArenaAllocator \* alloc, char \* (ArenaAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

alloc->Free(obj, (size\_t)-1);

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

// This object ensures DeleteObject (AllocatorDelete) for an allocator allocated object.

template <typename T, typename TAllocator>

class AutoAllocatorObjectPtr : public BasePtr<T>

{

private:

typedef typename AllocatorInfo<TAllocator, T>::AllocatorType AllocatorType;

AllocatorType\* m\_allocator;

public:

AutoAllocatorObjectPtr(T\* ptr, AllocatorType\* allocator) : BasePtr(ptr), m\_allocator(allocator)

{

Assert(allocator);

}

~AutoAllocatorObjectPtr()

{

Clear();

}

private:

void Clear()

{

if (ptr != nullptr)

{

DeleteObject<TAllocator>(m\_allocator, ptr);

ptr = nullptr;

}

}

};

// The version of AutoArrayPtr that uses allocator to release the memory.

template <typename T, typename TAllocator>

class AutoAllocatorArrayPtr : public BasePtr<T>

{

protected:

typedef typename AllocatorInfo<TAllocator, T>::AllocatorType AllocatorType;

size\_t m\_elementCount;

AllocatorType\* m\_allocator;

public:

AutoAllocatorArrayPtr(T \* ptr, size\_t elementCount, AllocatorType\* allocator) : BasePtr(ptr), m\_elementCount(elementCount), m\_allocator(allocator)

{

Assert(allocator);

}

~AutoAllocatorArrayPtr()

{

Clear();

}

// Do not support "operator=(T\* ptr)". The new ptr may have a different elementCount.

private:

void Clear()

{

if (ptr != nullptr)

{

DeleteArray<TAllocator>(m\_allocator, m\_elementCount, ptr);

ptr = nullptr;

}

}

};

// This version of AutoArrayPtr points to an array of AllocatorObject pointers (T\*). It ensures AllocatorDelete

// each AllocatorObject pointer contained in the array, before deleting the array itself.

//

// Template parameter:

// T The object type allocated from allocator. The array contains T\*.

// TAllocator The allocator type used to allocate/free the objects.

// ArrayAllocator The allocator type used to allocate/free the array.

//

template <typename T, typename TAllocator, typename ArrayAllocator = ForceNonLeafAllocator<TAllocator>::AllocatorType>

class AutoAllocatorObjectArrayPtr : public AutoAllocatorArrayPtr<T\*, ArrayAllocator>

{

public:

AutoAllocatorObjectArrayPtr(T\*\* ptr, size\_t elementCount, AllocatorType\* allocator) :

AutoAllocatorArrayPtr(ptr, elementCount, allocator)

{

}

~AutoAllocatorObjectArrayPtr()

{

Clear();

}

// Do not support "operator=(T\* ptr)". The new ptr may have a different elementCount.

private:

void Clear()

{

if (ptr != nullptr)

{

for (size\_t i = 0; i < m\_elementCount; i++)

{

if (ptr[i] != nullptr)

{

DeleteObject<TAllocator>(m\_allocator, ptr[i]);

ptr[i] = nullptr;

}

}

}

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template <typename T, typename HeapAllocatorT = HeapAllocator>

class AutoPtr : public BasePtr<T>

{

public:

AutoPtr(T \* ptr) : BasePtr(ptr) {}

~AutoPtr()

{

Clear();

}

AutoPtr& operator=(T \* ptr)

{

Clear();

this->ptr = ptr;

return \*this;

}

private:

void Clear()

{

if (ptr != nullptr)

{

AllocatorDelete(HeapAllocatorT, &HeapAllocatorT::Instance, ptr);

ptr = nullptr;

}

}

};

template <typename T>

class AutoArrayPtr : public BasePtr<T>

{

protected:

size\_t m\_elementCount;

public:

AutoArrayPtr(T \* ptr, size\_t elementCount) : BasePtr(ptr), m\_elementCount(elementCount) {}

~AutoArrayPtr()

{

Clear();

}

void Set(T\* ptr, int elementCount)

{

Clear();

this->ptr = ptr;

this->m\_elementCount = elementCount;

}

private:

void Clear()

{

if (ptr != nullptr)

{

HeapDeleteArray(m\_elementCount, ptr);

ptr = nullptr;

}

}

};

template <typename T>

class AutoArrayAndItemsPtr : public AutoArrayPtr<T>

{

public:

AutoArrayAndItemsPtr(T \* ptr, size\_t elementCount) : AutoArrayPtr(ptr, elementCount) {}

~AutoArrayAndItemsPtr()

{

Clear();

}

private:

void Clear()

{

if (ptr != nullptr){

for (size\_t i = 0; i < this->m\_elementCount; i++)

{

if (ptr[i] != nullptr)

{

ptr[i]->CleanUp();

ptr[i] = nullptr;

}

}

HeapDeleteArray(m\_elementCount, ptr);

ptr = nullptr;

}

}

};

template <typename T>

class AutoReleasePtr : public BasePtr<T>

{

public:

AutoReleasePtr(T \* ptr = nullptr) : BasePtr(ptr) {}

~AutoReleasePtr()

{

Release();

}

void Release()

{

if (ptr != nullptr)

{

ptr->Release();

this->ptr = nullptr;

}

}

};

template < typename T>

class AutoCOMPtr : public AutoReleasePtr<T>

{

public:

AutoCOMPtr(T \* ptr = nullptr) : AutoReleasePtr(ptr)

{

if (ptr != nullptr)

{

ptr->AddRef();

}

}

};

class AutoBSTR : public BasePtr<OLECHAR>

{

public:

AutoBSTR(BSTR ptr = nullptr) : BasePtr(ptr) {}

~AutoBSTR()

{

Release();

}

void Release()

{

if (ptr != nullptr)

{

::SysFreeString(ptr);

this->ptr = nullptr;

}

}

};

template <typename T>

class AutoDiscardPTR : public BasePtr < T >

{

public:

AutoDiscardPTR(T \* ptr) : BasePtr(ptr) {}

~AutoDiscardPTR()

{

Clear();

}

AutoDiscardPTR& operator=(T \* ptr)

{

Clear();

this->ptr = ptr;

return \*this;

}

private:

void Clear()

{

if (ptr != nullptr)

{

ptr->Discard();

ptr = nullptr;

}

}

};

<?xml version="1.0" encoding="utf-8"?>

<Project DefaultTargets="Build" ToolsVersion="12.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Import Condition="'$(ChakraBuildPathImported)'!='true'" Project="$(SolutionDir)Chakra.Build.Paths.props" />

<Import Project="$(BuildConfigPropsPath)Chakra.Build.ProjectConfiguration.props" />

<PropertyGroup Label="Globals">

<TargetName>Chakra.Common.Memory</TargetName>

<ProjectGuid>{BB4153FF-AC3E-4734-B562-FF23812DF31B}</ProjectGuid>

<RootNamespace>JS</RootNamespace>

<Keyword>Win32Proj</Keyword>

</PropertyGroup>

<PropertyGroup Label="Configuration">

<ConfigurationType>StaticLibrary</ConfigurationType>

</PropertyGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />

<ImportGroup Label="ExtensionSettings">

<Import Project="$(VCTargetsPath)\BuildCustomizations\masm.props" />

<Import Project="$(BuildConfig\_ARMASM\_Path)armasm.props" />

</ImportGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.props" />

<PropertyGroup>

<\_ProjectFileVersion>10.0.30319.1</\_ProjectFileVersion>

</PropertyGroup>

<ItemDefinitionGroup>

<ClCompile>

<AdditionalIncludeDirectories>

$(MSBuildThisFileDirectory)..;

%(AdditionalIncludeDirectories)

</AdditionalIncludeDirectories>

<PrecompiledHeader>Use</PrecompiledHeader>

<PrecompiledHeaderFile>CommonMemoryPch.h</PrecompiledHeaderFile>

</ClCompile>

</ItemDefinitionGroup>

<ItemGroup>

<ClCompile Include="$(MSBuildThisFileDirectory)Allocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)amd64\XDataAllocator.cpp">

<ExcludedFromBuild Condition="'$(Platform)'!='x64'">true</ExcludedFromBuild>

<!-- Since there are more then one XDataAllocator.cpp, we need to set them output into different directory, even when they are ExcludedFromBuild -->

<ObjectFileName Condition="'$(Platform)'!='x64'">$(IntDir)\amd64</ObjectFileName>

</ClCompile>

<ClCompile Include="$(MSBuildThisFileDirectory)arm\XDataAllocator.cpp">

<ExcludedFromBuild Condition="'$(Platform)'!='ARM'">true</ExcludedFromBuild>

<!-- Since there are more then one XDataAllocator.cpp, we need to set them output into different directory, even when they are ExcludedFromBuild -->

<ObjectFileName Condition="'$(Platform)'!='ARM'">$(IntDir)\arm</ObjectFileName>

</ClCompile>

<ClCompile Include="$(MSBuildThisFileDirectory)ArenaAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CustomHeap.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)EtwMemoryTracking.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)ForcedMemoryConstraints.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)HeapAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)HeapAllocatorOperators.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)HeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)HeapBlockMap.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)heapbucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)heapinfo.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)IdleDecommitPageAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)LargeHeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)leakreport.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)LargeHeapBucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)MarkContext.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)MemoryTracking.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)MemoryLogger.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)PageAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Recycler.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerHeuristic.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerObjectDumper.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerObjectGraphDumper.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerPageAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerSweep.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerWriteBarrierManager.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallFinalizableHeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallFinalizableHeapBucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallHeapBlockAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallLeafHeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallLeafHeapBucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallNormalHeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallNormalHeapBucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)StressTest.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)VirtualAllocWrapper.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CommonMemoryPch.cpp">

<PrecompiledHeader>Create</PrecompiledHeader>

</ClCompile>

</ItemGroup>

<ItemGroup>

<ClInclude Include="Allocator.h" />

<ClInclude Include="amd64\XDataAllocator.h">

<ExcludedFromBuild Condition="'$(Platform)'!='x64'">true</ExcludedFromBuild>

</ClInclude>

<ClInclude Include="ArenaAllocator.h" />

<ClInclude Include="arm\XDataAllocator.h">

<ExcludedFromBuild Condition="'$(Platform)'!='ARM'">true</ExcludedFromBuild>

</ClInclude>

<ClInclude Include="AutoAllocatorObjectPtr.h" />

<ClInclude Include="AutoPtr.h" />

<ClInclude Include="collectionstate.h" />

<ClInclude Include="CommonMemoryPch.h" />

<ClInclude Include="CustomHeap.h" />

<ClInclude Include="ForcedMemoryConstraints.h" />

<ClInclude Include="FreeObject.h" />

<ClInclude Include="HeapAllocator.h" />

<ClInclude Include="HeapBlock.h" />

<ClInclude Include="HeapBlockMap.h" />

<ClInclude Include="heapbucket.h" />

<ClInclude Include="HeapConstants.h" />

<ClInclude Include="heapinfo.h" />

<ClInclude Include="IdleDecommitPageAllocator.h" />

<ClInclude Include="LargeHeapBlock.h" />

<ClInclude Include="leakreport.h" />

<ClInclude Include="LargeHeapBucket.h" />

<ClInclude Include="MarkContext.h" />

<ClInclude Include="MemoryTracking.h" />

<ClInclude Include="PageAllocator.h" />

<ClInclude Include="PageAllocatorDefines.h" />

<ClInclude Include="PageHeapBlockTypeFilter.h" />

<ClInclude Include="PagePool.h" />

<ClInclude Include="Recycler.h" />

<ClInclude Include="RecyclerFastAllocator.h" />

<ClInclude Include="RecyclerHeuristic.h" />

<ClInclude Include="RecyclerObjectDumper.h" />

<ClInclude Include="RecyclerObjectGraphDumper.h" />

<ClInclude Include="RecyclerPageAllocator.h" />

<ClInclude Include="RecyclerPointers.h" />

<ClInclude Include="RecyclerRootPtr.h" />

<ClInclude Include="RecyclerSweep.h" />

<ClInclude Include="RecyclerWeakReference.h" />

<ClInclude Include="RecyclerWriteBarrierManager.h" />

<ClInclude Include="SmallFinalizableHeapBlock.h" />

<ClInclude Include="SmallFinalizableHeapBucket.h" />

<ClInclude Include="SmallHeapBlockAllocator.h" />

<ClInclude Include="SmallLeafHeapBlock.h" />

<ClInclude Include="SmallLeafHeapBucket.h" />

<ClInclude Include="SmallNormalHeapBlock.h" />

<ClInclude Include="SmallNormalHeapBucket.h" />

<ClInclude Include="MemoryLogger.h" />

<ClInclude Include="StressTest.h" />

<ClInclude Include="VirtualAllocWrapper.h" />

<ClInclude Include="XDataAllocator.h" />

</ItemGroup>

<ItemGroup>

<None Include="heapblock.inl" />

<None Include="heapbucket.inl" />

<None Include="LargeHeapBucket.inl" />

<None Include="MarkContext.inl" />

<None Include="Recycler.inl" />

</ItemGroup>

<ItemGroup>

<MASM Include="$(MSBuildThisFileDirectory)amd64\amd64\_SAVE\_REGISTERS.asm">

<ExcludedFromBuild Condition="'$(Platform)'!='x64'">true</ExcludedFromBuild>

</MASM>

<ARMASM Include="$(MSBuildThisFileDirectory)arm\arm\_SAVE\_REGISTERS.asm">

<ExcludedFromBuild Condition="'$(Platform)'!='ARM'">true</ExcludedFromBuild>

</ARMASM>

<ARMASM Include="$(MSBuildThisFileDirectory)arm64\arm64\_SAVE\_REGISTERS.asm">

<ExcludedFromBuild Condition="'$(Platform)'!='Arm64'">true</ExcludedFromBuild>

</ARMASM>

</ItemGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.targets" Condition="exists('$(BuildConfigPropsPath)Chakra.Build.targets')" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />

<ImportGroup Label="ExtensionTargets">

<Import Project="$(VCTargetsPath)\BuildCustomizations\masm.targets" />

<Import Project="$(BuildConfig\_ARMASM\_Path)armasm.targets" />

</ImportGroup>

</Project>

<?xml version="1.0" encoding="utf-8"?>

<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<ItemGroup>

<ARMASM Include="$(MSBuildThisFileDirectory)arm\arm\_SAVE\_REGISTERS.asm">

<Filter>arm</Filter>

</ARMASM>

<ARMASM Include="$(MSBuildThisFileDirectory)arm64\arm64\_SAVE\_REGISTERS.asm">

<Filter>arm64</Filter>

</ARMASM>

</ItemGroup>

<ItemGroup>

<ClCompile Include="$(MSBuildThisFileDirectory)Allocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)ArenaAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CustomHeap.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)EtwMemoryTracking.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)ForcedMemoryConstraints.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)HeapAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)HeapAllocatorOperators.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)HeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)HeapBlockMap.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)heapbucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)heapinfo.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)IdleDecommitPageAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)LargeHeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)leakreport.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)LargeHeapBucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)MarkContext.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)MemoryTracking.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)MemoryLogger.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)PageAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)Recycler.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerHeuristic.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerObjectDumper.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerObjectGraphDumper.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerPageAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerSweep.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)RecyclerWriteBarrierManager.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallFinalizableHeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallFinalizableHeapBucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallHeapBlockAllocator.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallLeafHeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallLeafHeapBucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallNormalHeapBlock.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)SmallNormalHeapBucket.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)StressTest.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)VirtualAllocWrapper.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)CommonMemoryPch.cpp" />

<ClCompile Include="$(MSBuildThisFileDirectory)amd64\XDataAllocator.cpp">

<Filter>amd64</Filter>

</ClCompile>

<ClCompile Include="$(MSBuildThisFileDirectory)arm\XDataAllocator.cpp">

<Filter>arm</Filter>

</ClCompile>

</ItemGroup>

<ItemGroup>

<ClInclude Include="Allocator.h" />

<ClInclude Include="ArenaAllocator.h" />

<ClInclude Include="AutoAllocatorObjectPtr.h" />

<ClInclude Include="AutoPtr.h" />

<ClInclude Include="CommonMemoryPch.h" />

<ClInclude Include="CustomHeap.h" />

<ClInclude Include="ForcedMemoryConstraints.h" />

<ClInclude Include="FreeObject.h" />

<ClInclude Include="HeapAllocator.h" />

<ClInclude Include="HeapBlock.h" />

<ClInclude Include="HeapBlockMap.h" />

<ClInclude Include="heapbucket.h" />

<ClInclude Include="HeapConstants.h" />

<ClInclude Include="heapinfo.h" />

<ClInclude Include="IdleDecommitPageAllocator.h" />

<ClInclude Include="LargeHeapBlock.h" />

<ClInclude Include="leakreport.h" />

<ClInclude Include="LargeHeapBucket.h" />

<ClInclude Include="MarkContext.h" />

<ClInclude Include="MemoryTracking.h" />

<ClInclude Include="PageAllocator.h" />

<ClInclude Include="PageAllocatorDefines.h" />

<ClInclude Include="PageHeapBlockTypeFilter.h" />

<ClInclude Include="PagePool.h" />

<ClInclude Include="Recycler.h" />

<ClInclude Include="RecyclerFastAllocator.h" />

<ClInclude Include="RecyclerHeuristic.h" />

<ClInclude Include="RecyclerObjectDumper.h" />

<ClInclude Include="RecyclerObjectGraphDumper.h" />

<ClInclude Include="RecyclerPageAllocator.h" />

<ClInclude Include="RecyclerPointers.h" />

<ClInclude Include="RecyclerRootPtr.h" />

<ClInclude Include="RecyclerSweep.h" />

<ClInclude Include="RecyclerWeakReference.h" />

<ClInclude Include="RecyclerWriteBarrierManager.h" />

<ClInclude Include="SmallFinalizableHeapBlock.h" />

<ClInclude Include="SmallFinalizableHeapBucket.h" />

<ClInclude Include="SmallHeapBlockAllocator.h" />

<ClInclude Include="SmallLeafHeapBlock.h" />

<ClInclude Include="SmallLeafHeapBucket.h" />

<ClInclude Include="SmallNormalHeapBlock.h" />

<ClInclude Include="SmallNormalHeapBucket.h" />

<ClInclude Include="MemoryLogger.h" />

<ClInclude Include="StressTest.h" />

<ClInclude Include="VirtualAllocWrapper.h" />

<ClInclude Include="arm\XDataAllocator.h">

<Filter>arm</Filter>

</ClInclude>

<ClInclude Include="amd64\XDataAllocator.h">

<Filter>amd64</Filter>

</ClInclude>

<ClInclude Include="XDataAllocator.h" />

<ClInclude Include="collectionstate.h" />

</ItemGroup>

<ItemGroup>

<None Include="heapblock.inl" />

<None Include="heapbucket.inl" />

<None Include="LargeHeapBucket.inl" />

<None Include="MarkContext.inl" />

<None Include="Recycler.inl" />

</ItemGroup>

<ItemGroup>

<Filter Include="amd64">

<UniqueIdentifier>{3a29318a-c1c1-476a-93d8-e2710727b4c7}</UniqueIdentifier>

</Filter>

<Filter Include="arm">

<UniqueIdentifier>{8102112a-3a20-43e4-aa19-3c1b36c55e24}</UniqueIdentifier>

</Filter>

<Filter Include="arm64">

<UniqueIdentifier>{2ed36d89-84cb-443b-9400-fbb616d6fdf9}</UniqueIdentifier>

</Filter>

</ItemGroup>

<ItemGroup>

<MASM Include="$(MSBuildThisFileDirectory)amd64\amd64\_SAVE\_REGISTERS.asm">

<Filter>amd64</Filter>

</MASM>

</ItemGroup>

</Project>

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

enum CollectionState

{

Collection\_Mark = 0x00000001,

Collection\_Sweep = 0x00000002,

Collection\_Exit = 0x00000004,

Collection\_PreCollection = 0x00000008,

// Mark related states

Collection\_ResetMarks = 0x00000010,

Collection\_FindRoots = 0x00000020,

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

Collection\_Rescan = 0x00000040,

#endif

Collection\_FinishMark = 0x00000080,

// Sweep related states

#ifdef CONCURRENT\_GC\_ENABLED

Collection\_ConcurrentSweepSetup = 0x00000100,

#endif

Collection\_TransferSwept = 0x00000200,

// State attributes

#ifdef PARTIAL\_GC\_ENABLED

Collection\_Partial = 0x00001000,

#endif

#ifdef CONCURRENT\_GC\_ENABLED

Collection\_Concurrent = 0x00002000,

Collection\_ExecutingConcurrent = 0x00004000,

Collection\_FinishConcurrent = 0x00008000,

Collection\_ConcurrentMark = Collection\_Concurrent | Collection\_Mark,

Collection\_ConcurrentSweep = Collection\_Concurrent | Collection\_Sweep,

#endif

Collection\_Parallel = 0x00010000,

Collection\_PostCollectionCallback = 0x00020000,

Collection\_WrapperCallback = 0x00040000,

// Actual states

CollectionStateNotCollecting = 0, // not collecting

CollectionStateResetMarks = Collection\_Mark | Collection\_ResetMarks, // reset marks

CollectionStateFindRoots = Collection\_Mark | Collection\_FindRoots, // finding roots

CollectionStateMark = Collection\_Mark, // marking (in thread)

CollectionStateSweep = Collection\_Sweep, // sweeping (in thread)

CollectionStateTransferSwept = Collection\_Sweep | Collection\_TransferSwept, // transfer swept objects (after concurrent sweep)

CollectionStateExit = Collection\_Exit, // exiting concurrent thread

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

CollectionStateRescanFindRoots = Collection\_Mark | Collection\_Rescan | Collection\_FindRoots, // rescan (after concurrent mark)

CollectionStateRescanMark = Collection\_Mark | Collection\_Rescan, // rescan (after concurrent mark)

#endif

#ifdef CONCURRENT\_GC\_ENABLED

CollectionStateConcurrentResetMarks = Collection\_ConcurrentMark | Collection\_ResetMarks | Collection\_ExecutingConcurrent, // concurrent reset mark

CollectionStateConcurrentFindRoots = Collection\_ConcurrentMark | Collection\_FindRoots | Collection\_ExecutingConcurrent, // concurrent findroot

CollectionStateConcurrentMark = Collection\_ConcurrentMark | Collection\_ExecutingConcurrent, // concurrent marking

CollectionStateRescanWait = Collection\_ConcurrentMark | Collection\_FinishConcurrent, // rescan (after concurrent mark)

CollectionStateConcurrentFinishMark = Collection\_ConcurrentMark | Collection\_ExecutingConcurrent | Collection\_FinishConcurrent,

CollectionStateSetupConcurrentSweep = Collection\_Sweep | Collection\_ConcurrentSweepSetup, // setting up concurrent sweep

CollectionStateConcurrentSweep = Collection\_ConcurrentSweep | Collection\_ExecutingConcurrent, // concurrent sweep

CollectionStateTransferSweptWait = Collection\_ConcurrentSweep | Collection\_FinishConcurrent, // transfer swept objects (after concurrent sweep)

#endif

CollectionStateParallelMark = Collection\_Mark | Collection\_Parallel,

CollectionStateBackgroundParallelMark = Collection\_ConcurrentMark | Collection\_ExecutingConcurrent | Collection\_Parallel,

CollectionStatePostCollectionCallback = Collection\_PostCollectionCallback,

CollectionStateConcurrentWrapperCallback = Collection\_Concurrent | Collection\_ExecutingConcurrent | Collection\_WrapperCallback,

};

///-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonMinMemory.h"

typedef \_Return\_type\_success\_(return >= 0) LONG NTSTATUS;

#define NT\_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)

// === C Runtime Header Files ===

#include <time.h>

#if defined(\_UCRT)

#include <cmath>

#else

#include <math.h>

#endif

// Exceptions

#include "Exceptions\Exceptionbase.h"

#include "Exceptions\OutOfMemoryException.h"

// Other Memory headers

#include "Memory\leakreport.h"

#include "Memory\AutoPtr.h"

// Other core headers

#include "Core\FinalizableObject.h"

#include "core\EtwTraceCore.h"

#include "core\ProfileInstrument.h"

#include "core\ProfileMemory.h"

#include "core\StackBackTrace.h"

#pragma warning(push)

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(HEAP\_TRACK\_ALLOC) || defined(ENABLE\_DEBUG\_CONFIG\_OPTIONS)

#include <typeinfo.h>

#endif

#pragma warning(pop)

// Inl files

#include "Memory\Recycler.inl"

#include "Memory\MarkContext.inl"

#include "Memory\HeapBucket.inl"

#include "Memory\LargeHeapBucket.inl"

#include "Memory\HeapBlock.inl"

#include "Memory\HeapBlockMap.inl"

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef \_M\_X64

#include "Memory\amd64\XDataAllocator.h"

#elif defined(\_M\_ARM)

#include "Memory\arm\XDataAllocator.h"

#include <wchar.h>

#elif defined(\_M\_ARM64)

#include "Memory\arm64\XDataAllocator.h"

#endif

#include "CustomHeap.h"

namespace Memory

{

namespace CustomHeap

{

#pragma region "Constructor and Destructor"

Heap::Heap(AllocationPolicyManager \* policyManager, ArenaAllocator \* alloc, bool allocXdata):

auxilliaryAllocator(alloc),

allocXdata(allocXdata),

#if DBG\_DUMP

freeObjectSize(0),

totalAllocationSize(0),

allocationsSinceLastCompact(0),

freesSinceLastCompact(0),

#endif

#if DBG

inDtor(false),

#endif

pageAllocator(policyManager, allocXdata, true /\*excludeGuardPages\*/),

preReservedHeapPageAllocator(policyManager, allocXdata, true /\*excludeGuardPages\*/),

cs(4000)

{

for (int i = 0; i < NumBuckets; i++)

{

this->buckets[i].Reset();

}

}

Heap::~Heap()

{

#if DBG

inDtor = true;

#endif

this->FreeAll();

}

#pragma endregion

#pragma region "Public routines"

void Heap::FreeAll()

{

FreeBuckets(false);

FreeLargeObjects();

FreeDecommittedBuckets();

FreeDecommittedLargeObjects();

}

bool Heap::Free(\_\_in Allocation\* object)

{

Assert(object != nullptr);

if (object == nullptr)

{

return false;

}

BucketId bucket = (BucketId) GetBucketForSize(object->size);

if (bucket == BucketId::LargeObjectList)

{

#if PDATA\_ENABLED

if(!object->xdata.IsFreed())

{

FreeXdata(&object->xdata, object->largeObjectAllocation.segment);

}

#endif

if (object->largeObjectAllocation.isDecommitted)

{

return true;

}

return FreeLargeObject<false>(object);

}

#if PDATA\_ENABLED

if(!object->xdata.IsFreed())

{

FreeXdata(&object->xdata, object->page->segment);

}

#endif

if (object->page->isDecommitted)

{

return true;

}

return FreeAllocation(object);

}

bool Heap::Decommit(\_\_in Allocation\* object)

{

// This function doesn't really touch the page allocator data structure.

// DecommitPages is merely a wrapper for VirtualFree

// So no need to take the critical section to synchronize

Assert(object != nullptr);

if (object == nullptr)

{

return false;

}

Assert(object->isAllocationUsed);

BucketId bucket = (BucketId) GetBucketForSize(object->size);

if (bucket == BucketId::LargeObjectList)

{

Assert(!object->largeObjectAllocation.isDecommitted);

if (!object->largeObjectAllocation.isDecommitted)

{

#if PDATA\_ENABLED

if(!object->xdata.IsFreed())

{

FreeXdata(&object->xdata, object->largeObjectAllocation.segment);

}

#endif

this->DecommitPages(object->address, object->GetPageCount(), object->largeObjectAllocation.segment);

this->largeObjectAllocations.MoveElementTo(object, &this->decommittedLargeObjects);

object->largeObjectAllocation.isDecommitted = true;

return true;

}

}

// Skip asserting here- multiple objects could be on the same page

// Review: should we really decommit here or decommit only when all objects

// on the page have been decommitted?

if (!object->page->isDecommitted)

{

#if PDATA\_ENABLED

if(!object->xdata.IsFreed())

{

FreeXdata(&object->xdata, object->page->segment);

}

#endif

bucket = object->page->currentBucket;

this->DecommitPages(object->page->address, 1, object->page->segment);

if (this->ShouldBeInFullList(object->page))

{

this->fullPages[bucket].MoveElementTo(object->page, &this->decommittedPages);

}

else

{

this->buckets[bucket].MoveElementTo(object->page, &this->decommittedPages);

}

object->page->isDecommitted = true;

}

return true;

}

bool Heap::IsInRange(\_\_in void\* address)

{

AutoCriticalSection autocs(&this->cs);

return (this->preReservedHeapPageAllocator.GetVirtualAllocator()->IsInRange(address) || this->pageAllocator.IsAddressFromAllocator(address));

}

/\*

\* Algorithm:

\* - Find bucket

\* - Check bucket pages - if it has enough free space, allocate that chunk

\* - Check pages in bigger buckets - if that has enough space, split that page and allocate from that chunk

\* - Allocate new page

\*/

Allocation\* Heap::Alloc(size\_t bytes, ushort pdataCount, ushort xdataSize, bool canAllocInPreReservedHeapPageSegment, bool isAnyJittedCode, \_Inout\_ bool\* isAllJITCodeInPreReservedRegion)

{

Assert(bytes > 0);

Assert((allocXdata || pdataCount == 0) && (!allocXdata || pdataCount > 0));

Assert(pdataCount > 0 || (pdataCount == 0 && xdataSize == 0));

// Round up to power of two to allocate, and figure out which bucket to allocate in

size\_t bytesToAllocate = PowerOf2Policy::GetSize(bytes);

BucketId bucket = (BucketId) GetBucketForSize(bytesToAllocate);

Allocation\* allocation;

if (bucket == BucketId::LargeObjectList)

{

return AllocLargeObject(bytes, pdataCount, xdataSize, canAllocInPreReservedHeapPageSegment, isAnyJittedCode, isAllJITCodeInPreReservedRegion);

}

VerboseHeapTrace(L"Bucket is %d\n", bucket);

VerboseHeapTrace(L"Requested: %d bytes. Allocated: %d bytes\n", bytes, bytesToAllocate);

Page\* page = nullptr;

if(!this->buckets[bucket].Empty())

{

page = &this->buckets[bucket].Head();

}

else

{

page = FindPageToSplit(bucket, canAllocInPreReservedHeapPageSegment);

}

if(page == nullptr)

{

page = AllocNewPage(bucket, canAllocInPreReservedHeapPageSegment, isAnyJittedCode, isAllJITCodeInPreReservedRegion);

}

// Out of memory

if (page == nullptr)

{

return nullptr;

}

allocation = AllocInPage(page, bytesToAllocate, pdataCount, xdataSize);

return allocation;

}

BOOL Heap::ProtectAllocation(\_\_in Allocation\* allocation, DWORD dwVirtualProtectFlags, \_\_out DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag)

{

Assert(allocation != nullptr);

Assert(allocation->isAllocationUsed);

return ProtectAllocationInternal(allocation, nullptr, dwVirtualProtectFlags, dwOldVirtualProtectFlags, desiredOldProtectFlag);

}

BOOL Heap::ProtectAllocationPage(\_\_in Allocation\* allocation, \_\_in char\* addressInPage, DWORD dwVirtualProtectFlags, \_\_out DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag)

{

Assert(addressInPage != nullptr);

Assert(allocation != nullptr);

Assert(addressInPage >= allocation->address);

Assert(allocation->isAllocationUsed);

return ProtectAllocationInternal(allocation, addressInPage, dwVirtualProtectFlags, dwOldVirtualProtectFlags, desiredOldProtectFlag);

}

BOOL Heap::ProtectAllocationInternal(\_\_in Allocation\* allocation, \_\_in\_opt char\* addressInPage, DWORD dwVirtualProtectFlags, \_\_out DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag)

{

// Allocate at the page level so that our protections don't

// transcend allocation page boundaries. Here, allocation->address is page

// aligned if the object is a large object allocation. If it isn't, in the else

// branch of the following if statement, we set it to the allocation's page's

// address. This ensures that the address being protected is always page aligned

char\* address = allocation->address;

#ifdef \_CONTROL\_FLOW\_GUARD

if (AutoSystemInfo::Data.IsCFGEnabled() &&

(dwVirtualProtectFlags & (PAGE\_EXECUTE | PAGE\_EXECUTE\_READ | PAGE\_EXECUTE\_READWRITE)))

{

AssertMsg(!(dwVirtualProtectFlags & PAGE\_EXECUTE\_WRITECOPY), "PAGE\_EXECUTE\_WRITECOPY is not used today. Remove this precondition \

and add to the if condition above, if this flag is used.");

dwVirtualProtectFlags |= PAGE\_TARGETS\_NO\_UPDATE;

}

#endif

size\_t pageCount;

void \* segment;

if (allocation->IsLargeAllocation())

{

#if DBG\_DUMP || defined(RECYCLER\_TRACE)

if (Js::Configuration::Global.flags.IsEnabled(Js::TraceProtectPagesFlag))

{

Output::Print(L"Protecting large allocation\n");

}

#endif

segment = allocation->largeObjectAllocation.segment;

allocation->largeObjectAllocation.isReadWrite = ((dwVirtualProtectFlags & PAGE\_READWRITE) == PAGE\_READWRITE);

if (addressInPage != nullptr)

{

if (addressInPage >= allocation->address + AutoSystemInfo::PageSize)

{

size\_t page = (addressInPage - allocation->address) / AutoSystemInfo::PageSize;

address = allocation->address + (page \* AutoSystemInfo::PageSize);

}

pageCount = 1;

}

else

{

pageCount = allocation->GetPageCount();

}

VerboseHeapTrace(L"Protecting 0x%p with 0x%x\n", address, dwVirtualProtectFlags);

return this->ProtectPages(address, pageCount, segment, dwVirtualProtectFlags, dwOldVirtualProtectFlags, desiredOldProtectFlag);

}

else

{

#if DBG\_DUMP || defined(RECYCLER\_TRACE)

if (Js::Configuration::Global.flags.IsEnabled(Js::TraceProtectPagesFlag))

{

Output::Print(L"Protecting small allocation\n");

}

#endif

segment = allocation->page->segment;

address = allocation->page->address;

allocation->page->isReadWrite = ((dwVirtualProtectFlags & PAGE\_READWRITE) == PAGE\_READWRITE);

pageCount = 1;

VerboseHeapTrace(L"Protecting 0x%p with 0x%x\n", address, dwVirtualProtectFlags);

return this->ProtectPages(address, pageCount, segment, dwVirtualProtectFlags, dwOldVirtualProtectFlags, desiredOldProtectFlag);

}

}

#pragma endregion

#pragma region "Large object methods"

Allocation\* Heap::AllocLargeObject(size\_t bytes, ushort pdataCount, ushort xdataSize, bool canAllocInPreReservedHeapPageSegment, bool isAnyJittedCode, \_Inout\_ bool\* isAllJITCodeInPreReservedRegion)

{

size\_t pages = GetNumPagesForSize(bytes);

if (pages == 0)

{

return nullptr;

}

void \* segment = nullptr;

char\* address = nullptr;

#if PDATA\_ENABLED

XDataAllocation xdata;

#endif

{

AutoCriticalSection autocs(&this->cs);

if (canAllocInPreReservedHeapPageSegment)

{

address = this->preReservedHeapPageAllocator.Alloc(&pages, (SegmentBase<PreReservedVirtualAllocWrapper>\*\*)(&segment));

}

if (address == nullptr)

{

if (isAnyJittedCode)

{

\*isAllJITCodeInPreReservedRegion = false;

}

address = this->pageAllocator.Alloc(&pages, (Segment\*\*)&segment);

}

// Out of memory

if (address == nullptr)

{

return nullptr;

}

FillDebugBreak((BYTE\*) address, pages\*AutoSystemInfo::PageSize);

#if PDATA\_ENABLED

if(pdataCount > 0)

{

if (!this->AllocSecondary(segment, (ULONG\_PTR) address, bytes, pdataCount, xdataSize, &xdata))

{

AutoCriticalSection autocs(&this->cs);

this->Release(address, pages, segment);

return nullptr;

}

}

#endif

}

Allocation\* allocation = this->largeObjectAllocations.PrependNode(this->auxilliaryAllocator);

if (allocation == nullptr)

{

AutoCriticalSection autocs(&this->cs);

this->Release(address, pages, segment);

#if PDATA\_ENABLED

if(pdataCount > 0)

{

this->ReleaseSecondary(xdata, segment);

}

#endif

return nullptr;

}

allocation->address = address;

allocation->largeObjectAllocation.segment = segment;

allocation->largeObjectAllocation.isDecommitted = false;

allocation->largeObjectAllocation.isReadWrite = true;

allocation->size = pages \* AutoSystemInfo::PageSize;

#if PDATA\_ENABLED

allocation->xdata = xdata;

if (((Segment\*)segment)->GetSecondaryAllocator() != nullptr && !((Segment\*)segment)->CanAllocSecondary())

{

TransferPages(

[&](Page\* currentPage) -> bool

{

bool transfer = currentPage->segment == segment;

if(transfer)

{

VerboseHeapTrace(L"Moving page from bucket %d to full list because no XDATA allocations can be made\n", currentPage->currentBucket);

}

return transfer;

} , this->buckets, this->fullPages);

}

#endif

return allocation;

}

void Heap::FreeDecommittedLargeObjects()

{

// This is only call when the heap is being destroy, so don't need to sync with the background thread.

Assert(inDtor);

FOREACH\_DLISTBASE\_ENTRY\_EDITING(Allocation, allocation, &this->decommittedLargeObjects, largeObjectIter)

{

VerboseHeapTrace(L"Decommitting large object at address 0x%p of size %u\n", allocation.address, allocation.size);

this->ReleaseDecommited(allocation.address, allocation.GetPageCount(), allocation.largeObjectAllocation.segment);

largeObjectIter.RemoveCurrent(this->auxilliaryAllocator);

}

NEXT\_DLISTBASE\_ENTRY\_EDITING;

}

template <bool freeAll>

bool Heap::FreeLargeObject(Allocation\* address)

{

AutoCriticalSection autocs(&this->cs);

FOREACH\_DLISTBASE\_ENTRY\_EDITING(Allocation, allocation, &this->largeObjectAllocations, largeObjectIter)

{

if (address == (&allocation) || freeAll)

{

EnsureAllocationWriteable(&allocation);

#if PDATA\_ENABLED

Assert(allocation.xdata.IsFreed());

#endif

this->Release(allocation.address, allocation.GetPageCount(), allocation.largeObjectAllocation.segment);

largeObjectIter.RemoveCurrent(this->auxilliaryAllocator);

if (!freeAll) return true;

}

}

NEXT\_DLISTBASE\_ENTRY\_EDITING;

// If we're not freeing everything, and we hit this point, that means that

// something that wasn't in the large object list was asked to be free.

// So, assert that we're freeing everything if we get to this point.

Assert(freeAll);

return false;

}

#pragma endregion

#pragma region "Page methods"

Allocation\* Heap::AllocInPage(Page\* page, size\_t bytes, ushort pdataCount, ushort xdataSize)

{

Assert(Math::IsPow2((int32)bytes));

uint length = GetChunkSizeForBytes(bytes);

BVIndex index = GetFreeIndexForPage(page, bytes);

Assert(index != BVInvalidIndex);

char\* address = page->address + Page::Alignment \* index;

#if PDATA\_ENABLED

XDataAllocation xdata;

if(pdataCount > 0)

{

AutoCriticalSection autocs(&this->cs);

{

if(!this->AllocSecondary(page->segment, (ULONG\_PTR)address, bytes, pdataCount, xdataSize, &xdata))

{

return nullptr;

}

}

}

#endif

Allocation\* allocation = AnewNoThrowStruct(this->auxilliaryAllocator, Allocation);

if (allocation == nullptr)

{

#if PDATA\_ENABLED

if(pdataCount > 0)

{

AutoCriticalSection autocs(&this->cs);

this->ReleaseSecondary(xdata, page->segment);

}

#endif

return nullptr;

}

#if DBG

allocation->isAllocationUsed = false;

allocation->isNotExecutableBecauseOOM = false;

#endif

allocation->page = page;

allocation->size = bytes;

allocation->address = address;

#if DBG\_DUMP

this->allocationsSinceLastCompact += bytes;

this->freeObjectSize -= bytes;

#endif

page->freeBitVector.ClearRange(index, length);

VerboseHeapTrace(L"ChunkSize: %d, Index: %d, Free bit vector in page: ", length, index);

#if VERBOSE\_HEAP

page->freeBitVector.DumpWord();

#endif

VerboseHeapTrace(L"\n");

if (this->ShouldBeInFullList(page))

{

BucketId bucket = page->currentBucket;

VerboseHeapTrace(L"Moving page from bucket %d to full list\n", bucket);

this->buckets[bucket].MoveElementTo(page, &this->fullPages[bucket]);

}

#if PDATA\_ENABLED

allocation->xdata = xdata;

if(((Segment\*)page->segment)->GetSecondaryAllocator() != nullptr && !((Segment\*)page->segment)->CanAllocSecondary())

{

TransferPages(

[&](Page\* currentPage) -> bool

{

bool transfer = currentPage->segment == page->segment;

if(transfer)

{

VerboseHeapTrace(L"Moving page from bucket %d to full list because no XDATA allocations can be made\n", page->currentBucket);

}

return transfer;

} , this->buckets, this->fullPages);

}

#endif

return allocation;

}

char \*

Heap::EnsurePreReservedPageAllocation(PreReservedVirtualAllocWrapper \* preReservedVirtualAllocator)

{

AutoCriticalSection autocs(&this->cs);

Assert(preReservedVirtualAllocator != nullptr);

Assert(preReservedHeapPageAllocator.GetVirtualAllocator() == preReservedVirtualAllocator);

char \* preReservedRegionStartAddress = (char\*)preReservedVirtualAllocator->GetPreReservedStartAddress();

if (preReservedRegionStartAddress == nullptr)

{

preReservedRegionStartAddress = preReservedHeapPageAllocator.InitPageSegment();

}

if (preReservedRegionStartAddress == nullptr)

{

VerboseHeapTrace(L"PRE-RESERVE: PreReserved Segment CANNOT be allocated \n");

}

return preReservedRegionStartAddress;

}

Page\* Heap::AllocNewPage(BucketId bucket, bool canAllocInPreReservedHeapPageSegment, bool isAnyJittedCode, \_Inout\_ bool\* isAllJITCodeInPreReservedRegion)

{

void\* pageSegment = nullptr;

char\* address = nullptr;

{

AutoCriticalSection autocs(&this->cs);

if (canAllocInPreReservedHeapPageSegment)

{

address = this->preReservedHeapPageAllocator.AllocPages(1, (PageSegmentBase<PreReservedVirtualAllocWrapper>\*\*)&pageSegment);

if (address == nullptr)

{

VerboseHeapTrace(L"PRE-RESERVE: PreReserved Segment CANNOT be allocated \n");

}

}

if (address == nullptr) // if no space in Pre-reserved Page Segment, then allocate in regular ones.

{

if (isAnyJittedCode)

{

\*isAllJITCodeInPreReservedRegion = false;

}

address = this->pageAllocator.AllocPages(1, (PageSegmentBase<VirtualAllocWrapper>\*\*)&pageSegment);

}

else

{

VerboseHeapTrace(L"PRE-RESERVE: Allocing new page in PreReserved Segment \n");

}

}

if (address == nullptr)

{

return nullptr;

}

FillDebugBreak((BYTE\*) address, AutoSystemInfo::PageSize);

// Switch to allocating on a list of pages so we can do leak tracking later

VerboseHeapTrace(L"Allocing new page in bucket %d\n", bucket);

Page\* page = this->buckets[bucket].PrependNode(this->auxilliaryAllocator, address, pageSegment, bucket);

if (page == nullptr)

{

AutoCriticalSection autocs(&this->cs);

this->ReleasePages(address, 1, pageSegment);

return nullptr;

}

#if DBG\_DUMP

this->totalAllocationSize += AutoSystemInfo::PageSize;

this->freeObjectSize += AutoSystemInfo::PageSize;

#endif

return page;

}

Page\* Heap::AddPageToBucket(Page\* page, BucketId bucket, bool wasFull)

{

Assert(bucket > BucketId::InvalidBucket && bucket < BucketId::NumBuckets);

BucketId oldBucket = page->currentBucket;

page->currentBucket = bucket;

if (wasFull)

{

#pragma prefast(suppress: \_\_WARNING\_UNCHECKED\_LOWER\_BOUND\_FOR\_ENUMINDEX, "targetBucket is always in range >= SmallObjectList, but an \_\_in\_range doesn't fix the warning.");

this->fullPages[oldBucket].MoveElementTo(page, &this->buckets[bucket]);

}

else

{

#pragma prefast(suppress: \_\_WARNING\_UNCHECKED\_LOWER\_BOUND\_FOR\_ENUMINDEX, "targetBucket is always in range >= SmallObjectList, but an \_\_in\_range doesn't fix the warning.");

this->buckets[oldBucket].MoveElementTo(page, &this->buckets[bucket]);

}

return page;

}

/\*

\* This method goes through the buckets greater than the target bucket

\* and if the higher bucket has a page with enough free space to allocate

\* something in the smaller bucket, then we bring the page to the smaller

\* bucket.

\* Note that if we allocate something from a page in the given bucket,

\* and then that page is split into a lower bucket, freeing is still not

\* a problem since the larger allocation is a multiple of the smaller one.

\* This gets more complicated if we can coalesce buckets. In that case,

\* we need to make sure that if a page was coalesced, and an allocation

\* pre-coalescing was freed, the page would need to get split upon free

\* to ensure correctness. For now, we've skipped implementing coalescing.

\* findPreReservedHeapPages - true, if we need to find pages only belonging to PreReservedHeapSegment

\*/

Page\* Heap::FindPageToSplit(BucketId targetBucket, bool findPreReservedHeapPages)

{

for (BucketId b = (BucketId)(targetBucket + 1); b < BucketId::NumBuckets; b = (BucketId) (b + 1))

{

#pragma prefast(suppress: \_\_WARNING\_UNCHECKED\_LOWER\_BOUND\_FOR\_ENUMINDEX, "targetBucket is always in range >= SmallObjectList, but an \_\_in\_range doesn't fix the warning.");

FOREACH\_DLISTBASE\_ENTRY\_EDITING(Page, pageInBucket, &this->buckets[b], bucketIter)

{

if (findPreReservedHeapPages && !IsPreReservedSegment(pageInBucket.segment))

{

//Find only pages that are pre-reserved using preReservedHeapPageAllocator

continue;

}

if (pageInBucket.CanAllocate(targetBucket))

{

Page\* page = &pageInBucket;

if (findPreReservedHeapPages)

{

VerboseHeapTrace(L"PRE-RESERVE: Found page for splitting in Pre Reserved Segment\n");

}

VerboseHeapTrace(L"Found page to split. Moving from bucket %d to %d\n", b, targetBucket);

return AddPageToBucket(page, targetBucket);

}

}

NEXT\_DLISTBASE\_ENTRY\_EDITING;

}

return nullptr;

}

void Heap::RemovePageFromFullList(Page\* pageToRemove)

{

FOREACH\_DLISTBASE\_ENTRY\_EDITING(Page, page, &this->fullPages[pageToRemove->currentBucket], pageIter)

{

if (&page == pageToRemove)

{

pageIter.RemoveCurrent(this->auxilliaryAllocator);

return;

}

}

NEXT\_DLISTBASE\_ENTRY\_EDITING;

// Page not found- why?

Assert(false);

}

BVIndex Heap::GetIndexInPage(\_\_in Page\* page, \_\_in char\* address)

{

Assert(page->address <= address && address < page->address + AutoSystemInfo::PageSize);

return (BVIndex) ((address - page->address) / Page::Alignment);

}

#pragma endregion

/\*\*

\* Free List methods

\*/

#pragma region "Freeing methods"

bool Heap::FreeAllocation(Allocation\* object)

{

Page\* page = object->page;

void\* segment = page->segment;

size\_t pageSize = AutoSystemInfo::PageSize;

unsigned int length = GetChunkSizeForBytes(object->size);

BVIndex index = GetIndexInPage(page, object->address);

#if DBG

// Make sure that it's not already been freed

for (BVIndex i = index; i < length; i++)

{

Assert(!page->freeBitVector.Test(i));

}

#endif

if (this->ShouldBeInFullList(page))

{

VerboseHeapTrace(L"Recycling page 0x%p because address 0x%p of size %d was freed\n", page->address, object->address, object->size);

// If the object being freed is equal to the page size, we're

// going to remove it anyway so don't add it to a bucket

if (object->size != pageSize)

{

AddPageToBucket(page, page->currentBucket, true);

}

else

{

EnsureAllocationWriteable(object);

// Fill the old buffer with debug breaks

CustomHeap::FillDebugBreak((BYTE \*)object->address, object->size);

void\* pageAddress = page->address;

RemovePageFromFullList(page);

// The page is not in any bucket- just update the stats, free the allocation

// and dump the page- we don't need to update free object size since the object

// size is equal to the page size so they cancel each other out

#if DBG\_DUMP

this->totalAllocationSize -= pageSize;

#endif

this->auxilliaryAllocator->Free(object, sizeof(Allocation));

{

AutoCriticalSection autocs(&this->cs);

this->ReleasePages(pageAddress, 1, segment);

}

VerboseHeapTrace(L"FastPath: freeing page-sized object directly\n");

return true;

}

}

// If the page is about to become empty then we should not need

// to set it to executable and we don't expect to restore the

// previous protection settings.

if (page->freeBitVector.Count() == BVUnit::BitsPerWord - length)

{

EnsureAllocationWriteable(object);

}

else

{

EnsureAllocationExecuteWriteable(object);

}

// Fill the old buffer with debug breaks

CustomHeap::FillDebugBreak((BYTE \*)object->address, object->size);

VerboseHeapTrace(L"Setting %d bits starting at bit %d, Free bit vector in page was ", length, index);

#if VERBOSE\_HEAP

page->freeBitVector.DumpWord();

#endif

VerboseHeapTrace(L"\n");

page->freeBitVector.SetRange(index, length);

VerboseHeapTrace(L"Free bit vector in page: ", length, index);

#if VERBOSE\_HEAP

page->freeBitVector.DumpWord();

#endif

VerboseHeapTrace(L"\n");

#if DBG\_DUMP

this->freeObjectSize += object->size;

this->freesSinceLastCompact += object->size;

#endif

this->auxilliaryAllocator->Free(object, sizeof(Allocation));

if (page->IsEmpty())

{

// Find the page and remove it from the buckets- the page is going to be freed anyway

FOREACH\_DLISTBASE\_ENTRY\_EDITING(Page, pageInBucket, &this->buckets[page->currentBucket], pageIter)

{

// Templatize this to remove branches/make code more compact?

if (&pageInBucket == page)

{

VerboseHeapTrace(L"Removing page in bucket %d\n", page->currentBucket);

{

AutoCriticalSection autocs(&this->cs);

this->ReleasePages(page->address, 1, page->segment);

}

pageIter.RemoveCurrent(this->auxilliaryAllocator);

#if DBG\_DUMP

this->freeObjectSize -= pageSize;

this->totalAllocationSize -= pageSize;

#endif

return true;

}

}

NEXT\_DLISTBASE\_ENTRY\_EDITING;

return false;

}

else // after freeing part of the page, the page should be in PAGE\_EXECUTE\_READWRITE protection, and turning to PAGE\_EXECUTE

{

DWORD dwExpectedFlags = 0;

this->ProtectPages(page->address, 1, segment, PAGE\_EXECUTE, &dwExpectedFlags, PAGE\_EXECUTE\_READWRITE);

Assert(!object->isAllocationUsed || dwExpectedFlags == PAGE\_EXECUTE\_READWRITE);

page->isReadWrite = false;

return true;

}

}

void Heap::FreeDecommittedBuckets()

{

// This is only call when the heap is being destroy, so don't need to sync with the background thread.

Assert(inDtor);

FOREACH\_DLISTBASE\_ENTRY\_EDITING(Page, page, &this->decommittedPages, iter)

{

this->TrackDecommitedPages(page.address, 1, page.segment);

iter.RemoveCurrent(this->auxilliaryAllocator);

}

NEXT\_DLISTBASE\_ENTRY\_EDITING;

}

void Heap::FreePage(Page\* page)

{

// This is only call when the heap is being destroy, so don't need to sync with the background thread.

Assert(inDtor);

DWORD pageSize = AutoSystemInfo::PageSize;

EnsurePageWriteable(page);

size\_t freeSpace = page->freeBitVector.Count() \* Page::Alignment;

VerboseHeapTrace(L"Removing page in bucket %d, freeSpace: %d\n", page->currentBucket, freeSpace);

this->ReleasePages(page->address, 1, page->segment);

#if DBG\_DUMP

this->freeObjectSize -= freeSpace;

this->totalAllocationSize -= pageSize;

#endif

}

void Heap::FreeBucket(DListBase<Page>\* bucket, bool freeOnlyEmptyPages)

{

// This is only call when the heap is being destroy, so don't need to sync with the background thread.

Assert(inDtor);

FOREACH\_DLISTBASE\_ENTRY\_EDITING(Page, page, bucket, pageIter)

{

// Templatize this to remove branches/make code more compact?

if (!freeOnlyEmptyPages || page.IsEmpty())

{

FreePage(&page);

pageIter.RemoveCurrent(this->auxilliaryAllocator);

}

}

NEXT\_DLISTBASE\_ENTRY\_EDITING;

}

void Heap::FreeBuckets(bool freeOnlyEmptyPages)

{

// This is only call when the heap is being destroy, so don't need to sync with the background thread.

Assert(inDtor);

for (int i = 0; i < NumBuckets; i++)

{

FreeBucket(&this->buckets[i], freeOnlyEmptyPages);

FreeBucket(&this->fullPages[i], freeOnlyEmptyPages);

}

#if DBG\_DUMP

this->allocationsSinceLastCompact = 0;

this->freesSinceLastCompact = 0;

#endif

}

#if PDATA\_ENABLED

void Heap::FreeXdata(XDataAllocation\* xdata, void\* segment)

{

Assert(!xdata->IsFreed());

if(!((Segment\*)segment)->CanAllocSecondary())

{

this->TransferPages([&](Page\* currentPage) -> bool

{

bool transfer = currentPage->segment == segment && !currentPage->HasNoSpace();

if(transfer)

{

VerboseHeapTrace(L"Recycling page 0x%p because XDATA was freed\n", currentPage->address);

}

return transfer;

}, this->fullPages, this->buckets);

}

{

AutoCriticalSection autocs(&this->cs);

this->ReleaseSecondary(\*xdata, segment);

xdata->Free();

}

}

#endif

#if DBG\_DUMP

void Heap::DumpStats()

{

HeapTrace(L"Total allocation size: %d\n", totalAllocationSize);

HeapTrace(L"Total free size: %d\n", freeObjectSize);

HeapTrace(L"Total allocations since last compact: %d\n", allocationsSinceLastCompact);

HeapTrace(L"Total frees since last compact: %d\n", freesSinceLastCompact);

HeapTrace(L"Large object count: %d\n", this->largeObjectAllocations.Count());

HeapTrace(L"Buckets: \n");

for (int i = 0; i < BucketId::NumBuckets; i++)

{

printf("\t%d => %u [", (1 << (i + 7)), buckets[i].Count());

FOREACH\_DLISTBASE\_ENTRY\_EDITING(Page, page, &this->buckets[i], bucketIter)

{

BVUnit usedBitVector = page.freeBitVector;

usedBitVector.ComplimentAll(); // Get the actual used bit vector

printf(" %u ", usedBitVector.Count() \* Page::Alignment); // Print out the space used in this page

}

NEXT\_DLISTBASE\_ENTRY\_EDITING

printf("] {{%u}}\n", this->fullPages[i].Count());

}

}

#endif

#pragma endregion

/\*\*

\* Helper methods

\*/

#pragma region "Helpers"

inline unsigned int log2(size\_t number)

{

const unsigned int b[] = {0x2, 0xC, 0xF0, 0xFF00, 0xFFFF0000};

const unsigned int S[] = {1, 2, 4, 8, 16};

unsigned int result = 0;

for (int i = 4; i >= 0; i--)

{

if (number & b[i])

{

number >>= S[i];

result |= S[i];

}

}

return result;

}

inline BucketId GetBucketForSize(size\_t bytes)

{

if (bytes > Page::MaxAllocationSize)

{

return BucketId::LargeObjectList;

}

BucketId bucket = (BucketId) (log2(bytes) - 7);

// < 8 => 0

// 8 => 1

// 9 => 2 ...

Assert(bucket < BucketId::LargeObjectList);

if (bucket < BucketId::SmallObjectList)

{

bucket = BucketId::SmallObjectList;

}

return bucket;

}

// Fills the specified buffer with "debug break" instruction encoding.

// If there is any space left after that due to alignment, fill it with 0.

// static

void FillDebugBreak(\_\_out\_bcount\_full(byteCount) BYTE\* buffer, \_\_in size\_t byteCount)

{

#if defined(\_M\_ARM)

// On ARM there is breakpoint instruction (BKPT) which is 0xBEii, where ii (immediate 8) can be any value, 0xBE in particular.

// While it could be easier to put 0xBE (same way as 0xCC on x86), BKPT is not recommended -- it may cause unexpected side effects.

// So, use same sequence are C++ compiler uses (0xDEFE), this is recognized by debugger as \_\_debugbreak.

// This is 2 bytes, and in case there is a gap of 1 byte in the end, fill it with 0 (there is no 1 byte long THUMB instruction).

CompileAssert(sizeof(wchar\_t) == 2);

wchar\_t pattern = 0xDEFE;

wmemset(reinterpret\_cast<wchar\_t\*>(buffer), pattern, byteCount / 2);

if (byteCount % 2)

{

// Note: this is valid scenario: in JIT mode, we may not be 2-byte-aligned in the end of unwind info.

\*(buffer + byteCount - 1) = 0; // Fill last remaining byte.

}

#elif defined(\_M\_ARM64)

CompileAssert(sizeof(DWORD) == 4);

DWORD pattern = 0xd4200000 | (0xf000 << 5);

for (size\_t i = 0; i < byteCount / 4; i++)

{

reinterpret\_cast<DWORD\*>(buffer)[i] = pattern;

}

for (size\_t i = (byteCount / 4) \* 4; i < byteCount; i++)

{

// Note: this is valid scenario: in JIT mode, we may not be 2-byte-aligned in the end of unwind info.

buffer[i] = 0; // Fill last remaining bytes.

}

#else

// On Intel just use "INT 3" instruction which is 0xCC.

memset(buffer, 0xCC, byteCount);

#endif

}

#pragma endregion

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

#define VerboseHeapTrace(...) { \

OUTPUT\_VERBOSE\_TRACE(Js::CustomHeapPhase, \_\_VA\_ARGS\_\_); \

}

#define HeapTrace(...) { \

Output::Print(\_\_VA\_ARGS\_\_); \

Output::Flush(); \

}

namespace CustomHeap

{

enum BucketId

{

InvalidBucket = -1,

SmallObjectList,

Bucket256,

Bucket512,

Bucket1024,

Bucket2048,

Bucket4096,

LargeObjectList,

NumBuckets

};

BucketId GetBucketForSize(size\_t bytes);

struct PageAllocatorAllocation

{

bool isDecommitted;

bool isReadWrite;

};

struct Page: public PageAllocatorAllocation

{

void\* segment;

BVUnit freeBitVector;

char\* address;

BucketId currentBucket;

bool HasNoSpace()

{

return freeBitVector.IsEmpty();

}

bool IsEmpty()

{

return freeBitVector.IsFull();

}

bool CanAllocate(BucketId targetBucket)

{

return freeBitVector.FirstStringOfOnes(targetBucket + 1) != BVInvalidIndex;

}

Page(\_\_in char\* address, void\* segment, BucketId bucket):

address(address),

segment(segment),

currentBucket(bucket),

freeBitVector(0xFFFFFFFF)

{

// Initialize PageAllocatorAllocation fields

this->isDecommitted = false;

this->isReadWrite = true;

}

// Each bit in the bit vector corresponds to 128 bytes of memory

// This implies that 128 bytes is the smallest allocation possible

static const uint Alignment = 128;

static const uint MaxAllocationSize = 4096;

};

struct Allocation

{

union

{

Page\* page;

struct: PageAllocatorAllocation

{

void\* segment;

} largeObjectAllocation;

};

\_\_field\_bcount(size) char\* address;

size\_t size;

bool IsLargeAllocation() const { return size > Page::MaxAllocationSize; }

size\_t GetPageCount() const { Assert(this->IsLargeAllocation()); return size / AutoSystemInfo::PageSize; }

#if DBG

// Initialized to false, this is set to true when the allocation

// is actually used by the emit buffer manager

// This is almost always true- it's there only for assertion purposes

bool isAllocationUsed: 1;

bool isNotExecutableBecauseOOM: 1;

#endif

#if PDATA\_ENABLED

XDataAllocation xdata;

XDataAllocator\* GetXDataAllocator()

{

XDataAllocator\* allocator;

if (!this->IsLargeAllocation())

{

allocator = static\_cast<XDataAllocator\*>(((Segment\*)(this->page->segment))->GetSecondaryAllocator());

}

else

{

allocator = static\_cast<XDataAllocator\*>(((Segment\*) (largeObjectAllocation.segment))->GetSecondaryAllocator());

}

return allocator;

}

#if defined(\_M\_ARM32\_OR\_ARM64)

void RegisterPdata(ULONG\_PTR functionStart, DWORD length)

{

Assert(this->xdata.pdataCount > 0);

XDataAllocator\* xdataAllocator = GetXDataAllocator();

xdataAllocator->Register(this->xdata, functionStart, length);

}

#endif

#endif

};

/\*

\* Simple free-listing based heap allocator

\*

\* Each allocation is tracked using a "HeapAllocation" record

\* Once we alloc, we start assigning chunks sliced from the end of a HeapAllocation

\* If we don't have enough to slice off, we push a new heap allocation record to the record stack, and try and assign from that

\*/

class Heap

{

public:

Heap(AllocationPolicyManager \* policyManager, ArenaAllocator \* alloc, bool allocXdata);

Allocation\* Alloc(size\_t bytes, ushort pdataCount, ushort xdataSize, bool canAllocInPreReservedHeapPageSegment, bool isAnyJittedCode, \_Inout\_ bool\* isAllJITCodeInPreReservedRegion);

bool Free(\_\_in Allocation\* allocation);

bool Decommit(\_\_in Allocation\* allocation);

void FreeAll();

bool IsInRange(\_\_in void\* address);

template<typename T>

HeapPageAllocator<T>\* GetPageAllocator(Page \* page)

{

AssertMsg(page, "Why is page null?");

return GetPageAllocator<T>(page->segment);

}

template<typename T>

HeapPageAllocator<T>\* GetPageAllocator(void \* segmentParam)

{

SegmentBase<T> \* segment = (SegmentBase<T>\*)segmentParam;

AssertMsg(segment, "Why is segment null?");

AssertMsg(segment->GetAllocator(), "Segment doesn't have an allocator?");

Assert((HeapPageAllocator<VirtualAllocWrapper>\*)(segment->GetAllocator()) == &this->pageAllocator ||

(HeapPageAllocator<PreReservedVirtualAllocWrapper>\*)(segment->GetAllocator()) == &this->preReservedHeapPageAllocator);

return (HeapPageAllocator<T> \*)(segment->GetAllocator());

}

bool IsPreReservedSegment(void \* segment)

{

Assert(segment);

return (((Segment\*)(segment))->IsInPreReservedHeapPageAllocator());

}

HeapPageAllocator<PreReservedVirtualAllocWrapper> \* GetPreReservedHeapPageAllocator()

{

return &preReservedHeapPageAllocator;

}

HeapPageAllocator<VirtualAllocWrapper>\* GetHeapPageAllocator()

{

Assert(!pageAllocator.GetVirtualAllocator()->IsPreReservedRegionPresent());

return &pageAllocator;

}

void ReleasePages(void\* pageAddress, uint pageCount, \_\_in void\* segment)

{

Assert(segment);

if (IsPreReservedSegment(segment))

{

this->GetPageAllocator<PreReservedVirtualAllocWrapper>(segment)->ReleasePages(pageAddress, pageCount, segment);

}

else

{

this->GetPageAllocator<VirtualAllocWrapper>(segment)->ReleasePages(pageAddress, pageCount, segment);

}

}

BOOL ProtectPages(\_\_in char\* address, size\_t pageCount, \_\_in void\* segment, DWORD dwVirtualProtectFlags, DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag)

{

Assert(segment);

if (IsPreReservedSegment(segment))

{

return this->GetPageAllocator<PreReservedVirtualAllocWrapper>(segment)->ProtectPages(address, pageCount, segment, dwVirtualProtectFlags, dwOldVirtualProtectFlags, desiredOldProtectFlag);

}

else

{

return this->GetPageAllocator<VirtualAllocWrapper>(segment)->ProtectPages(address, pageCount, segment, dwVirtualProtectFlags, dwOldVirtualProtectFlags, desiredOldProtectFlag);

}

}

void TrackDecommitedPages(void \* address, uint pageCount, \_\_in void\* segment)

{

Assert(segment);

if (IsPreReservedSegment(segment))

{

this->GetPageAllocator<PreReservedVirtualAllocWrapper>(segment)->TrackDecommitedPages(address, pageCount, segment);

}

else

{

this->GetPageAllocator<VirtualAllocWrapper>(segment)->TrackDecommitedPages(address, pageCount, segment);

}

}

void ReleaseSecondary(const SecondaryAllocation& allocation, void\* segment)

{

Assert(segment);

if (IsPreReservedSegment(segment))

{

this->GetPageAllocator<PreReservedVirtualAllocWrapper>(segment)->ReleaseSecondary(allocation, segment);

}

else

{

this->GetPageAllocator<VirtualAllocWrapper>(segment)->ReleaseSecondary(allocation, segment);

}

}

void DecommitPages(\_\_in char\* address, size\_t pageCount, void\* segment)

{

Assert(segment);

if (IsPreReservedSegment(segment))

{

this->GetPageAllocator<PreReservedVirtualAllocWrapper>(segment)->DecommitPages(address, pageCount);

}

else

{

this->GetPageAllocator<VirtualAllocWrapper>(segment)->DecommitPages(address, pageCount);

}

}

bool AllocSecondary(void\* segment, ULONG\_PTR functionStart, size\_t functionSize\_t, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation)

{

Assert(functionSize\_t <= MAXUINT32);

DWORD functionSize = static\_cast<DWORD>(functionSize\_t);

Assert(segment);

if (IsPreReservedSegment(segment))

{

return this->GetPageAllocator<PreReservedVirtualAllocWrapper>(segment)->AllocSecondary(segment, functionStart, functionSize, pdataCount, xdataSize, allocation);

}

else

{

return this->GetPageAllocator<VirtualAllocWrapper>(segment)->AllocSecondary(segment, functionStart, functionSize, pdataCount, xdataSize, allocation);

}

}

void Release(void \* address, size\_t pageCount, void \* segment)

{

Assert(segment);

if (IsPreReservedSegment(segment))

{

this->GetPageAllocator<PreReservedVirtualAllocWrapper>(segment)->Release(address, pageCount, segment);

}

else

{

this->GetPageAllocator<VirtualAllocWrapper>(segment)->Release(address, pageCount, segment);

}

}

void ReleaseDecommited(void \* address, size\_t pageCount, \_\_in void \* segment)

{

Assert(segment);

if (IsPreReservedSegment(segment))

{

this->GetPageAllocator<PreReservedVirtualAllocWrapper>(segment)->ReleaseDecommited(address, pageCount, segment);

}

else

{

this->GetPageAllocator<VirtualAllocWrapper>(segment)->ReleaseDecommited(address, pageCount, segment);

}

}

char \* EnsurePreReservedPageAllocation(PreReservedVirtualAllocWrapper \* preReservedVirtualAllocator);

// A page should be in full list if:

// 1. It does not have any space

// 2. Parent segment cannot allocate any more XDATA

bool ShouldBeInFullList(Page\* page)

{

return page->HasNoSpace() || (allocXdata && !((Segment\*)(page->segment))->CanAllocSecondary());

}

BOOL ProtectAllocation(\_\_in Allocation\* allocation, DWORD dwVirtualProtectFlags, \_\_out DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag);

BOOL ProtectAllocationPage(\_\_in Allocation\* allocation, \_\_in char\* addressInPage, DWORD dwVirtualProtectFlags, \_\_out DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag);

DWORD EnsureAllocationProtection(Allocation\* allocation, bool readWrite)

{

if (readWrite)

{

// this only call from InterpreterThunkEmitter

return EnsureAllocationReadWrite<true, PAGE\_READWRITE>(allocation);

}

else

{

return EnsureAllocationReadWrite<false, PAGE\_READWRITE>(allocation);

}

}

~Heap();

#if DBG\_DUMP

void DumpStats();

#endif

private:

/\*\*

\* Inline methods

\*/

inline unsigned int GetChunkSizeForBytes(size\_t bytes)

{

return (bytes > Page::Alignment ? static\_cast<unsigned int>(bytes) / Page::Alignment : 1);

}

inline size\_t GetNumPagesForSize(size\_t bytes)

{

size\_t allocSize = AllocSizeMath::Add(bytes, AutoSystemInfo::PageSize);

if (allocSize == (size\_t) -1)

{

return 0;

}

return ((allocSize - 1)/ AutoSystemInfo::PageSize);

}

inline BVIndex GetFreeIndexForPage(Page\* page, size\_t bytes)

{

unsigned int length = GetChunkSizeForBytes(bytes);

BVIndex index = page->freeBitVector.FirstStringOfOnes(length);

return index;

}

/\*\*

\* Large object methods

\*/

Allocation\* AllocLargeObject(size\_t bytes, ushort pdataCount, ushort xdataSize, bool canAllocInPreReservedHeapPageSegment, bool isAnyJittedCode, \_Inout\_ bool\* isAllJITCodeInPreReservedRegion);

template<bool freeAll>

bool FreeLargeObject(Allocation\* header);

void FreeLargeObjects()

{

FreeLargeObject<true>(nullptr);

}

DWORD EnsurePageWriteable(Page\* page)

{

return EnsurePageReadWrite<true, PAGE\_READWRITE>(page);

}

// this get called when freeing the whole page

DWORD EnsureAllocationWriteable(Allocation\* allocation)

{

return EnsureAllocationReadWrite<true, PAGE\_READWRITE>(allocation);

}

// this get called when only freeing a part in the page

DWORD EnsureAllocationExecuteWriteable(Allocation\* allocation)

{

return EnsureAllocationReadWrite<true, PAGE\_EXECUTE\_READWRITE>(allocation);

}

template<bool readWrite, DWORD readWriteFlags>

DWORD EnsurePageReadWrite(Page\* page)

{

if (readWrite)

{

if (!page->isReadWrite && !page->isDecommitted)

{

DWORD dwOldProtectFlags = 0;

BOOL result = this->ProtectPages(page->address, 1, page->segment, readWriteFlags, &dwOldProtectFlags, PAGE\_EXECUTE);

page->isReadWrite = true;

Assert(result && (dwOldProtectFlags & readWriteFlags) == 0);

return dwOldProtectFlags;

}

}

else

{

if (page->isReadWrite && !page->isDecommitted)

{

DWORD dwOldProtectFlags = 0;

BOOL result = this->ProtectPages(page->address, 1, page->segment, PAGE\_EXECUTE, &dwOldProtectFlags, readWriteFlags);

page->isReadWrite = false;

Assert(result && (dwOldProtectFlags & PAGE\_EXECUTE) == 0);

return dwOldProtectFlags;

}

}

return 0;

}

template<bool readWrite, DWORD readWriteFlags>

DWORD EnsureAllocationReadWrite(Allocation\* allocation)

{

if (allocation->IsLargeAllocation())

{

if (readWrite)

{

if (!allocation->largeObjectAllocation.isReadWrite)

{

DWORD dwOldProtectFlags;

BOOL result = this->ProtectAllocation(allocation, readWriteFlags, &dwOldProtectFlags, PAGE\_EXECUTE);

Assert(result && (dwOldProtectFlags & readWriteFlags) == 0);

return dwOldProtectFlags;

}

}

else

{

if (allocation->largeObjectAllocation.isReadWrite)

{

DWORD dwOldProtectFlags;

this->ProtectAllocation(allocation, PAGE\_EXECUTE, &dwOldProtectFlags, readWriteFlags);

Assert((dwOldProtectFlags & PAGE\_EXECUTE) == 0);

return dwOldProtectFlags;

}

}

}

else

{

return EnsurePageReadWrite<readWrite, readWriteFlags>(allocation->page);

}

// 0 is safe to return as its not a memory protection constant

// so it indicates that nothing was changed

return 0;

}

BOOL ProtectAllocationInternal(\_\_in Allocation\* allocation, \_\_in\_opt char\* addressInPage, DWORD dwVirtualProtectFlags, \_\_out DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag);

/\*\*

\* Freeing Methods

\*/

void FreeBuckets(bool freeOnlyEmptyPages);

void FreeBucket(DListBase<Page>\* bucket, bool freeOnlyEmptyPages);

void FreePage(Page\* page);

bool FreeAllocation(Allocation\* allocation);

#if PDATA\_ENABLED

void FreeXdata(XDataAllocation\* xdata, void\* segment);

#endif

void FreeDecommittedBuckets();

void FreeDecommittedLargeObjects();

/\*\*

\* Page methods

\*/

Page\* AddPageToBucket(Page\* page, BucketId bucket, bool wasFull = false);

Allocation\* AllocInPage(Page\* page, size\_t bytes, ushort pdataCount, ushort xdataSize);

Page\* AllocNewPage(BucketId bucket, bool canAllocInPreReservedHeapPageSegment, bool isAnyJittedCode, \_Inout\_ bool\* isAllJITCodeInPreReservedRegion);

Page\* FindPageToSplit(BucketId targetBucket, bool findPreReservedHeapPages = false);

template<class Fn>

void TransferPages(Fn predicate, DListBase<Page>\* fromList, DListBase<Page>\* toList)

{

Assert(fromList != toList);

for(int bucket = 0; bucket < BucketId::NumBuckets; bucket++)

{

FOREACH\_DLISTBASE\_ENTRY\_EDITING(Page, page, &(fromList[bucket]), bucketIter)

{

if(predicate(&page))

{

bucketIter.MoveCurrentTo(&(toList[bucket]));

}

}

NEXT\_DLISTBASE\_ENTRY\_EDITING;

}

}

BVIndex GetIndexInPage(\_\_in Page\* page, \_\_in char\* address);

void RemovePageFromFullList(Page\* page);

/\*\*

\* Stats

\*/

#if DBG\_DUMP

size\_t totalAllocationSize;

size\_t freeObjectSize;

size\_t allocationsSinceLastCompact;

size\_t freesSinceLastCompact;

#endif

/\*\*

\* Allocator stuff

\*/

HeapPageAllocator<VirtualAllocWrapper> pageAllocator;

HeapPageAllocator<PreReservedVirtualAllocWrapper> preReservedHeapPageAllocator;

ArenaAllocator\* auxilliaryAllocator;

/\*

\* Various tracking lists

\*/

DListBase<Page> buckets[NumBuckets];

DListBase<Page> fullPages[NumBuckets];

DListBase<Allocation> largeObjectAllocations;

DListBase<Page> decommittedPages;

DListBase<Allocation> decommittedLargeObjects;

// Critical section synchronize in the BGJIT thread and IsInRange in the main thread

CriticalSection cs;

bool allocXdata;

#if DBG

bool inDtor;

#endif

};

// Helpers

unsigned int log2(size\_t number);

BucketId GetBucketForSize(size\_t bytes);

void FillDebugBreak(\_\_out\_bcount\_full(byteCount) BYTE\* buffer, \_\_in size\_t byteCount);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef ETW\_MEMORY\_TRACKING

#include "microsoft-scripting-jscript9.internalevents.h"

enum ArenaType: unsigned short

{

ArenaTypeRecycler = 1,

ArenaTypeArena = 2

};

class EtwMemoryEvents

{

public:

\_\_declspec(noinline) static void ReportAllocation(void \*arena, void \*address, size\_t size)

{

allocCount++;

if (allocCount == 7500)

{

allocCount = 0;

::Sleep(350);

}

EventWriteJSCRIPT\_INTERNAL\_ALLOCATOR\_ALLOC(arena, address, size);

}

\_\_declspec(noinline) static void ReportFree(void \*arena, void \*address, size\_t size)

{

EventWriteJSCRIPT\_INTERNAL\_ALLOCATOR\_FREE(arena, address, size);

}

\_\_declspec(noinline) static void ReportReallocation(void \*arena, void \*address, size\_t existingSize, size\_t newSize)

{

EventWriteJSCRIPT\_INTERNAL\_ALLOCATOR\_FREE(arena, address, existingSize);

EventWriteJSCRIPT\_INTERNAL\_ALLOCATOR\_ALLOC(arena, address, newSize);

}

\_\_declspec(noinline) static void ReportArenaCreated(void \*arena, ArenaType arenaType)

{

EventWriteJSCRIPT\_INTERNAL\_ALLOCATOR\_CREATE(arena, arenaType);

}

\_\_declspec(noinline) static void ReportArenaDestroyed(void \*arena)

{

EventWriteJSCRIPT\_INTERNAL\_ALLOCATOR\_DESTROY(arena);

}

\_\_declspec(noinline) static void ReportFreeAll(void \*arena, ArenaType arenaType)

{

EventWriteJSCRIPT\_INTERNAL\_ALLOCATOR\_DESTROY(arena);

EventWriteJSCRIPT\_INTERNAL\_ALLOCATOR\_CREATE(arena, arenaType);

}

private:

static int allocCount;

};

int EtwMemoryEvents::allocCount = 0;

// Workaround to stop the linker from collapsing the Arena/recycler methods to

// a single implementation so that we have nicer stacks

ArenaType g\_arena;

#define DISTINGUISH\_FUNCTION(arenaType) g\_arena = ArenaType##arenaType

void ArenaMemoryTracking::Activate()

{

}

\_\_declspec(noinline) void ArenaMemoryTracking::ArenaCreated(Allocator \*arena, \_\_in LPCWSTR name)

{

DISTINGUISH\_FUNCTION(Arena);

EtwMemoryEvents::ReportArenaCreated(arena, ArenaTypeArena);

}

\_\_declspec(noinline) void ArenaMemoryTracking::ArenaDestroyed(Allocator \*arena)

{

DISTINGUISH\_FUNCTION(Arena);

EtwMemoryEvents::ReportArenaDestroyed(arena);

}

\_\_declspec(noinline) void ArenaMemoryTracking::ReportAllocation(Allocator \*arena, void \*address, size\_t size)

{

DISTINGUISH\_FUNCTION(Arena);

EtwMemoryEvents::ReportAllocation(arena, address, size);

}

\_\_declspec(noinline) void ArenaMemoryTracking::ReportReallocation(Allocator \*arena, void \*address, size\_t existingSize, size\_t newSize)

{

DISTINGUISH\_FUNCTION(Arena);

EtwMemoryEvents::ReportReallocation(arena, address, existingSize, newSize);

}

\_\_declspec(noinline) void ArenaMemoryTracking::ReportFree(Allocator \*arena, void \*address, size\_t size)

{

DISTINGUISH\_FUNCTION(Arena);

EtwMemoryEvents::ReportFree(arena, address, size);

}

\_\_declspec(noinline) void ArenaMemoryTracking::ReportFreeAll(Allocator \*arena)

{

DISTINGUISH\_FUNCTION(Arena);

EtwMemoryEvents::ReportFreeAll(arena, ArenaTypeArena);

}

// Recycler tracking

void RecyclerMemoryTracking::Activate()

{

}

bool RecyclerMemoryTracking::IsActive()

{

return true;

}

// The external reporting for the recycler uses the MemspectMemoryTracker

\_\_declspec(noinline) void RecyclerMemoryTracking::ReportRecyclerCreate(Recycler \* recycler)

{

DISTINGUISH\_FUNCTION(Recycler);

EtwMemoryEvents::ReportArenaCreated(recycler, ArenaTypeRecycler);

}

\_\_declspec(noinline) void RecyclerMemoryTracking::ReportRecyclerDestroy(Recycler \* recycler)

{

DISTINGUISH\_FUNCTION(Recycler);

EtwMemoryEvents::ReportArenaDestroyed(recycler);

}

\_\_declspec(noinline) void RecyclerMemoryTracking::ReportAllocation(Recycler \* recycler, \_\_in void \*address, size\_t size)

{

DISTINGUISH\_FUNCTION(Recycler);

EtwMemoryEvents::ReportAllocation(recycler, address, size);

}

\_\_declspec(noinline) void RecyclerMemoryTracking::ReportFree(Recycler \* recycler, \_\_in void \*address, size\_t size)

{

DISTINGUISH\_FUNCTION(Recycler);

EtwMemoryEvents::ReportFree(recycler, address, size);

}

void RecyclerMemoryTracking::ReportUnallocated(Recycler \* recycler, \_\_in void\* address, \_\_in void \*endAddress, size\_t sizeCat)

{

byte \* byteAddress = (byte \*) address;

while (byteAddress + sizeCat <= endAddress)

{

EtwMemoryEvents::ReportAllocation(recycler, byteAddress, sizeCat);

byteAddress += sizeCat;

}

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

#include "Memory\ForcedMemoryConstraints.h"

void

ForcedMemoryConstraint::Apply()

{

if (Js::Configuration::Global.flags.IsEnabled(Js::ForceFragmentAddressSpaceFlag))

{

FragmentAddressSpace(Js::Configuration::Global.flags.ForceFragmentAddressSpace);

}

}

#pragma prefast(suppress:6262, "Where this function is call should have ample of stack space")

void ForcedMemoryConstraint::FragmentAddressSpace(size\_t usableSize)

{

// AMD64 address space is too big

#if !defined(\_M\_X64\_OR\_ARM64)

uint const allocationGranuality = 64 \* 1024; // 64 KB

Assert(allocationGranuality == AutoSystemInfo::Data.dwAllocationGranularity);

uint64 const addressEnd = ((uint64)4) \* 1024 \* 1024 \* 1024;

uint const freeSpaceSize = Math::Align<size\_t>(usableSize, allocationGranuality);

void \* address[addressEnd / allocationGranuality];

// Reserve a contiguous usable space

void \* freeAddress = ::VirtualAlloc(NULL, freeSpaceSize, MEM\_RESERVE, PAGE\_NOACCESS);

// Reserve the reset the address space

for (uint i = 1; i < \_countof(address); i++)

{

address[i] = ::VirtualAlloc((LPVOID)(i \* allocationGranuality), allocationGranuality, MEM\_RESERVE, PAGE\_NOACCESS);

}

// fragment

int j = \_countof(address) - 2;

do

{

if (address[j + 1] == nullptr)

{

j--;

continue;

}

::VirtualFree(address[j + 1], 0, MEM\_RELEASE);

j -= 2;

}

while (j > 0);

::VirtualFree(freeAddress, 0, MEM\_RELEASE);

#endif

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

class ForcedMemoryConstraint

{

public:

static void Apply();

private:

static void FragmentAddressSpace(size\_t usableSize);

};

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

struct FreeObject

{

public:

FreeObject \* GetNext() const

{

AssertMsg((taggedNext & TaggedBit) == TaggedBit, "Free list corrupted");

return (FreeObject \*)(taggedNext & ~TaggedBit);

}

void SetNext(FreeObject \* next)

{

Assert(((INT\_PTR)next & TaggedBit) == 0);

taggedNext = ((INT\_PTR)next) | TaggedBit;

}

void ZeroNext() { taggedNext = 0; }

#ifdef RECYCLER\_MEMORY\_VERIFY

#pragma warning(suppress:4310)

void DebugFillNext() { taggedNext = (INT\_PTR)0xCACACACACACACACA; }

#endif

private:

INT\_PTR taggedNext;

static INT\_PTR const TaggedBit = 0x1;

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

// Not enabled in ChakraCore

#include "MemProtectHeap.h"

#endif

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAT")

#ifdef HEAP\_TRACK\_ALLOC

CriticalSection HeapAllocator::cs;

#endif

#ifdef CHECK\_MEMORY\_LEAK

MemoryLeakCheck MemoryLeakCheck::leakCheck;

#endif

HeapAllocator HeapAllocator::Instance;

NoThrowHeapAllocator NoThrowHeapAllocator::Instance;

NoCheckHeapAllocator NoCheckHeapAllocator::Instance;

HANDLE NoCheckHeapAllocator::processHeap = NULL;

template <bool noThrow>

char \* HeapAllocator::AllocT(size\_t byteSize)

{

#ifdef HEAP\_TRACK\_ALLOC

size\_t requestedBytes = byteSize;

byteSize = AllocSizeMath::Add(requestedBytes, ::Math::Align<size\_t>(sizeof(HeapAllocRecord), MEMORY\_ALLOCATION\_ALIGNMENT));

TrackAllocData allocData;

ClearTrackAllocInfo(&allocData);

#elif defined(HEAP\_PERF\_COUNTERS)

size\_t requestedBytes = byteSize;

byteSize = AllocSizeMath::Add(requestedBytes, ::Math::Align<size\_t>(sizeof(size\_t), MEMORY\_ALLOCATION\_ALIGNMENT));

#endif

if (noThrow)

{

FAULTINJECT\_MEMORY\_NOTHROW(L"Heap", byteSize);

}

else

{

FAULTINJECT\_MEMORY\_THROW(L"Heap", byteSize);

}

char \* buffer;

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

if (DoUseMemProtectHeap())

{

void \* memory = MemProtectHeapRootAlloc(memProtectHeapHandle, byteSize);

if (memory == nullptr)

{

if (noThrow)

{

return nullptr;

}

Js::Throw::OutOfMemory();

}

buffer = (char \*)memory;

}

else

#endif

{

buffer = (char \*)malloc(byteSize);

}

if (!noThrow && buffer == nullptr)

{

Js::Throw::OutOfMemory();

}

#if defined(HEAP\_TRACK\_ALLOC) || defined(HEAP\_PERF\_COUNTERS)

if (!noThrow || buffer != nullptr)

{

#ifdef HEAP\_TRACK\_ALLOC

cs.Enter();

data.LogAlloc((HeapAllocRecord \*)buffer, requestedBytes, allocData);

cs.Leave();

buffer += ::Math::Align<size\_t>(sizeof(HeapAllocRecord), MEMORY\_ALLOCATION\_ALIGNMENT);

#else

\*(size\_t \*)buffer = requestedBytes;

buffer += ::Math::Align<size\_t>(sizeof(size\_t), MEMORY\_ALLOCATION\_ALIGNMENT);

#endif

HEAP\_PERF\_COUNTER\_INC(LiveObject);

HEAP\_PERF\_COUNTER\_ADD(LiveObjectSize, requestedBytes);

}

#endif

return buffer;

}

template char \* HeapAllocator::AllocT<true>(size\_t byteSize);

template char \* HeapAllocator::AllocT<false>(size\_t byteSize);

void HeapAllocator::Free(void \* buffer, size\_t byteSize)

{

#ifdef HEAP\_TRACK\_ALLOC

if (buffer != nullptr)

{

HeapAllocRecord \* record = (HeapAllocRecord \*)(((char \*)buffer) - ::Math::Align<size\_t>(sizeof(HeapAllocRecord), MEMORY\_ALLOCATION\_ALIGNMENT));

Assert(byteSize == (size\_t)-1 || record->size == byteSize);

HEAP\_PERF\_COUNTER\_DEC(LiveObject);

HEAP\_PERF\_COUNTER\_SUB(LiveObjectSize, record->size);

cs.Enter();

data.LogFree(record);

cs.Leave();

buffer = record;

#if DBG

memset(buffer, DbgMemFill, record->size + ::Math::Align<size\_t>(sizeof(HeapAllocRecord), MEMORY\_ALLOCATION\_ALIGNMENT));

#endif

}

#elif defined(HEAP\_PERF\_COUNTERS)

if (buffer != nullptr)

{

HEAP\_PERF\_COUNTER\_DEC(LiveObject);

size\_t \* allocSize = (size\_t \*)(((char \*)buffer) - ::Math::Align<size\_t>(sizeof(size\_t), MEMORY\_ALLOCATION\_ALIGNMENT));

HEAP\_PERF\_COUNTER\_SUB(LiveObjectSize, \*allocSize);

buffer = allocSize;

}

#endif

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

if (DoUseMemProtectHeap())

{

HRESULT hr = MemProtectHeapUnrootAndZero(memProtectHeapHandle, buffer);

Assert(SUCCEEDED(hr));

return;

}

#endif

free(buffer);

}

#ifdef TRACK\_ALLOC

#ifdef HEAP\_TRACK\_ALLOC

\_\_declspec(thread) TrackAllocData HeapAllocator::nextAllocData;

#endif

HeapAllocator \* HeapAllocator::TrackAllocInfo(TrackAllocData const& data)

{

#ifdef HEAP\_TRACK\_ALLOC

Assert(nextAllocData.IsEmpty());

nextAllocData = data;

#endif

return this;

}

void HeapAllocator::ClearTrackAllocInfo(TrackAllocData\* data/\* = NULL\*/)

{

#ifdef HEAP\_TRACK\_ALLOC

Assert(!nextAllocData.IsEmpty());

if (data)

{

\*data = nextAllocData;

}

nextAllocData.Clear();

#endif

}

#endif

#ifdef HEAP\_TRACK\_ALLOC

//static

bool HeapAllocator::CheckLeaks()

{

return Instance.data.CheckLeaks();

}

#endif HEAP\_TRACK\_ALLOC

char \* NoThrowHeapAllocator::AllocZero(size\_t byteSize)

{

return HeapAllocator::Instance.NoThrowAllocZero(byteSize);

}

char \* NoThrowHeapAllocator::Alloc(size\_t byteSize)

{

return HeapAllocator::Instance.NoThrowAlloc(byteSize);

}

void NoThrowHeapAllocator::Free(void \* buffer, size\_t byteSize)

{

HeapAllocator::Instance.Free(buffer, byteSize);

}

#ifdef TRACK\_ALLOC

NoThrowHeapAllocator \* NoThrowHeapAllocator::TrackAllocInfo(TrackAllocData const& data)

{

HeapAllocator::Instance.TrackAllocInfo(data);

return this;

}

#endif TRACK\_ALLOC

#ifdef TRACK\_ALLOC

void NoThrowHeapAllocator::ClearTrackAllocInfo(TrackAllocData\* data /\*= NULL\*/)

{

HeapAllocator::Instance.ClearTrackAllocInfo(data);

}

#endif TRACK\_ALLOC

HeapAllocator \* HeapAllocator::GetNoMemProtectInstance()

{

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

// Used only in Chakra, no need to use CUSTOM\_CONFIG\_FLAG

if (CONFIG\_FLAG(MemProtectHeap))

{

return &NoMemProtectInstance;

}

#endif

return &Instance;

}

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

HeapAllocator HeapAllocator::NoMemProtectInstance(false);

HeapAllocator::HeapAllocator(bool allocMemProtect) : isUsed(false), memProtectHeapHandle(nullptr), allocMemProtect(allocMemProtect)

{

}

bool HeapAllocator::DoUseMemProtectHeap()

{

if (!allocMemProtect)

{

return false;

}

if (memProtectHeapHandle != nullptr)

{

return true;

}

DebugOnly(bool wasUsed = isUsed);

isUsed = true;

// Flag is used only in Chakra, no need to use CUSTOM\_CONFIG\_FLAG

if (CONFIG\_FLAG(MemProtectHeap))

{

Assert(!wasUsed);

if (FAILED(MemProtectHeapCreate(&memProtectHeapHandle, MemProtectHeapCreateFlags\_ProtectCurrentStack)))

{

Assert(false);

}

return true;

}

return false;

}

void HeapAllocator::FinishMemProtectHeapCollect()

{

if (memProtectHeapHandle)

{

MemProtectHeapCollect(memProtectHeapHandle, MemProtectHeap\_ForceFinishCollect);

DebugOnly(MemProtectHeapSetDisableConcurrentThreadExitedCheck(memProtectHeapHandle));

}

}

NoThrowNoMemProtectHeapAllocator NoThrowNoMemProtectHeapAllocator::Instance;

char \* NoThrowNoMemProtectHeapAllocator::AllocZero(size\_t byteSize)

{

return HeapAllocator::GetNoMemProtectInstance()->NoThrowAllocZero(byteSize);

}

char \* NoThrowNoMemProtectHeapAllocator::Alloc(size\_t byteSize)

{

return HeapAllocator::GetNoMemProtectInstance()->NoThrowAlloc(byteSize);

}

void NoThrowNoMemProtectHeapAllocator::Free(void \* buffer, size\_t byteSize)

{

HeapAllocator::GetNoMemProtectInstance()->Free(buffer, byteSize);

}

#ifdef TRACK\_ALLOC

NoThrowNoMemProtectHeapAllocator \* NoThrowNoMemProtectHeapAllocator::TrackAllocInfo(TrackAllocData const& data)

{

HeapAllocator::GetNoMemProtectInstance()->TrackAllocInfo(data);

return this;

}

#endif TRACK\_ALLOC

#ifdef TRACK\_ALLOC

void NoThrowNoMemProtectHeapAllocator::ClearTrackAllocInfo(TrackAllocData\* data /\*= NULL\*/)

{

HeapAllocator::GetNoMemProtectInstance()->ClearTrackAllocInfo(data);

}

#endif TRACK\_ALLOC

#endif

#if defined(HEAP\_TRACK\_ALLOC) || defined(INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC)

HeapAllocator::~HeapAllocator()

{

#ifdef HEAP\_TRACK\_ALLOC

bool hasFakeHeapLeak = false;

auto fakeHeapLeak = [&]()

{

// REVIEW: Okay to use global flags?

if (Js::Configuration::Global.flags.ForceMemoryLeak && !hasFakeHeapLeak)

{

AUTO\_HANDLED\_EXCEPTION\_TYPE(ExceptionType\_DisableCheck);

struct FakeMemory { int f; };

HeapNewStruct(FakeMemory);

hasFakeHeapLeak = true;

}

};

#ifdef LEAK\_REPORT

// REVIEW: Okay to use global flags?

if (Js::Configuration::Global.flags.IsEnabled(Js::LeakReportFlag))

{

fakeHeapLeak();

LeakReport::StartSection(L"Heap Leaks");

LeakReport::StartRedirectOutput();

bool leaked = !HeapAllocator::CheckLeaks();

LeakReport::EndRedirectOutput();

LeakReport::EndSection();

LeakReport::Print(L"--------------------------------------------------------------------------------\n");

if (leaked)

{

LeakReport::Print(L"Heap Leaked Object: %d bytes (%d objects)\n",

data.outstandingBytes, data.allocCount - data.deleteCount);

}

}

#endif // LEAK\_REPORT

#ifdef CHECK\_MEMORY\_LEAK

// REVIEW: Okay to use global flags?

if (Js::Configuration::Global.flags.CheckMemoryLeak)

{

fakeHeapLeak();

Output::CaptureStart();

Output::Print(L"-------------------------------------------------------------------------------------\n");

Output::Print(L"Heap Leaks\n");

Output::Print(L"-------------------------------------------------------------------------------------\n");

if (!HeapAllocator::CheckLeaks())

{

Output::Print(L"-------------------------------------------------------------------------------------\n");

Output::Print(L"Heap Leaked Object: %d bytes (%d objects)\n",

data.outstandingBytes, data.allocCount - data.deleteCount);

wchar\_t \* buffer = Output::CaptureEnd();

MemoryLeakCheck::AddLeakDump(buffer, data.outstandingBytes, data.allocCount - data.deleteCount);

}

else

{

free(Output::CaptureEnd());

}

}

#endif // CHECK\_MEMORY\_LEAK

#endif // HEAP\_TRACK\_ALLOC

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

if (memProtectHeapHandle != nullptr)

{

MemProtectHeapDestroy(memProtectHeapHandle);

}

#endif // INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

}

#endif // defined(HEAP\_TRACK\_ALLOC) || defined(INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC)

#ifdef HEAP\_TRACK\_ALLOC

void

HeapAllocatorData::LogAlloc(HeapAllocRecord \* record, size\_t requestedBytes, TrackAllocData const& data)

{

record->prev = nullptr;

record->size = requestedBytes;

record->data = this;

record->next = head;

record->allocId = allocCount;

record->allocData = data;

if (head != nullptr)

{

head->prev = record;

}

head = record;

outstandingBytes += requestedBytes;

allocCount++;

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

// REVIEW: Okay to use global flags?

if (Js::Configuration::Global.flags.LeakStackTrace)

{

// Allocation done before the flags is parse doesn't get a stack trace

record->stacktrace = StackBackTrace::Capture(&NoCheckHeapAllocator::Instance, 1, StackTraceDepth);

}

else

{

record->stacktrace = nullptr;

}

#endif

}

void

HeapAllocatorData::LogFree(HeapAllocRecord \* record)

{

Assert(record->data == this);

// This is an expensive check for double free

#if 0

HeapAllocRecord \* curr = head;

while (curr != nullptr)

{

if (curr == record)

{

break;

}

curr = curr->next;

}

Assert(curr != nullptr);

#endif

if (record->next != nullptr)

{

record->next->prev = record->prev;

}

if (record->prev == nullptr)

{

head = record->next;

}

else

{

record->prev->next = record->next;

}

deleteCount++;

outstandingBytes -= record->size;

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

if (record->stacktrace != nullptr)

{

record->stacktrace->Delete(&NoCheckHeapAllocator::Instance);

}

#endif

}

bool

HeapAllocatorData::CheckLeaks()

{

bool needPause = false;

if (allocCount != deleteCount)

{

needPause = true;

HeapAllocRecord \* current = head;

while (current != nullptr)

{

Output::Print(L"%S%s", current->allocData.GetTypeInfo()->name(),

current->allocData.GetCount() == (size\_t)-1? L"" : L"[]");

Output::SkipToColumn(50);

Output::Print(L"- %p - %10d bytes\n",

((char\*)current) + ::Math::Align<size\_t>(sizeof(HeapAllocRecord), MEMORY\_ALLOCATION\_ALIGNMENT),

current->size);

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

// REVIEW: Okay to use global flags?

if (Js::Configuration::Global.flags.LeakStackTrace && current->stacktrace)

{

// Allocation done before the flags is parse doesn't get a stack trace

Output::Print(L" Allocation Stack:\n");

current->stacktrace->Print();

}

#endif

current = current->next;

}

}

else if (outstandingBytes != 0)

{

needPause = true;

Output::Print(L"Unbalanced new/delete size: %d\n", outstandingBytes);

}

Output::Flush();

#if defined(ENABLE\_DEBUG\_CONFIG\_OPTIONS) && !DBG

// REVIEW: Okay to use global flags?

if (needPause && Js::Configuration::Global.flags.Console)

{

//This is not defined for WinCE

HANDLE handle = GetStdHandle( STD\_INPUT\_HANDLE );

FlushConsoleInputBuffer(handle);

Output::Print(L"Press any key to continue...\n");

Output::Flush();

WaitForSingleObject(handle, INFINITE);

}

#endif

return allocCount == deleteCount && outstandingBytes == 0;

}

#endif

#ifdef CHECK\_MEMORY\_LEAK

MemoryLeakCheck::~MemoryLeakCheck()

{

if (head != nullptr)

{

if (enableOutput)

{

Output::Print(L"FATAL ERROR: Memory Leak Detected\n");

}

LeakRecord \* current = head;

do

{

if (enableOutput)

{

Output::PrintBuffer(current->dump, wcslen(current->dump));

}

LeakRecord \* prev = current;

current = current->next;

free((void \*)prev->dump);

NoCheckHeapDelete(prev);

}

while (current != nullptr);

if (enableOutput)

{

Output::Print(L"-------------------------------------------------------------------------------------\n");

Output::Print(L"Total leaked: %d bytes (%d objects)\n", leakedBytes, leakedCount);

Output::Flush();

}

if (enableOutput)

{

Js::Throw::GenerateDump(Js::Configuration::Global.flags.DumpOnCrash, true, true);

}

}

}

void

MemoryLeakCheck::AddLeakDump(wchar\_t const \* dump, size\_t bytes, size\_t count)

{

AutoCriticalSection autocs(&leakCheck.cs);

LeakRecord \* record = NoCheckHeapNewStruct(LeakRecord);

record->dump = dump;

record->next = nullptr;

if (leakCheck.tail == nullptr)

{

leakCheck.head = record;

leakCheck.tail = record;

}

else

{

leakCheck.tail->next = record;

leakCheck.tail = record;

}

leakCheck.leakedBytes += bytes;

leakCheck.leakedCount += count;

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#define HeapNew(T, ...) AllocatorNew(HeapAllocator, &HeapAllocator::Instance, T, \_\_VA\_ARGS\_\_)

#define HeapNewZ(T, ...) AllocatorNewZ(HeapAllocator, &HeapAllocator::Instance, T, \_\_VA\_ARGS\_\_)

#define HeapNewPlus(size, T, ...) AllocatorNewPlus(HeapAllocator, &HeapAllocator::Instance, size, T, \_\_VA\_ARGS\_\_)

#define HeapNewPlusZ(size, T, ...) AllocatorNewPlusZ(HeapAllocator, &HeapAllocator::Instance, size, T, \_\_VA\_ARGS\_\_)

#define HeapNewStruct(T) AllocatorNewStruct(HeapAllocator, &HeapAllocator::Instance, T);

#define HeapNewStructZ(T) AllocatorNewStructZ(HeapAllocator, &HeapAllocator::Instance, T);

#define HeapNewStructPlus(size, T, ...) AllocatorNewStructPlus(HeapAllocator, &HeapAllocator::Instance, size, T)

#define HeapNewStructPlusZ(size, T, ...) AllocatorNewStructPlusZ(HeapAllocator, &HeapAllocator::Instance, size, T)

#define HeapNewArray(T, count, ...) AllocatorNewArray(HeapAllocator, &HeapAllocator::Instance, T, count)

#define HeapNewArrayZ(T, count, ...) AllocatorNewArrayZ(HeapAllocator, &HeapAllocator::Instance, T, count)

#define HeapDelete(obj) AllocatorDelete(HeapAllocator, &HeapAllocator::Instance, obj)

#define HeapDeletePlus(size, obj) AllocatorDeletePlus(HeapAllocator, &HeapAllocator::Instance, size, obj)

#define HeapDeletePlusPrefix(size, obj) AllocatorDeletePlusPrefix(HeapAllocator, &HeapAllocator::Instance, size, obj)

#define HeapDeleteArray(count, obj) AllocatorDeleteArray(HeapAllocator, &HeapAllocator::Instance, count, obj)

#define HeapNewNoThrow(T, ...) AllocatorNewNoThrow(HeapAllocator, &HeapAllocator::Instance, T, \_\_VA\_ARGS\_\_)

#define HeapNewNoThrowZ(T, ...) AllocatorNewNoThrowZ(HeapAllocator, &HeapAllocator::Instance, T, \_\_VA\_ARGS\_\_)

#define HeapNewNoThrowPlus(size, T, ...) AllocatorNewNoThrowPlus(HeapAllocator, &HeapAllocator::Instance, size, T, \_\_VA\_ARGS\_\_)

#define HeapNewNoThrowPlusZ(size, T, ...) AllocatorNewNoThrowPlusZ(HeapAllocator, &HeapAllocator::Instance, size, T, \_\_VA\_ARGS\_\_)

#define HeapNewNoThrowPlusPrefixZ(size, T, ...) AllocatorNewNoThrowPlusPrefixZ(HeapAllocator, &HeapAllocator::Instance, size, T, \_\_VA\_ARGS\_\_)

#define HeapNewNoThrowStruct(T) AllocatorNewNoThrowStruct(HeapAllocator, &HeapAllocator::Instance, T)

#define HeapNewNoThrowStructZ(T) AllocatorNewNoThrowStructZ(HeapAllocator, &HeapAllocator::Instance, T)

#define HeapNewNoThrowArray(T, count, ...) AllocatorNewNoThrowArray(HeapAllocator, &HeapAllocator::Instance, T, count)

#define HeapNewNoThrowArrayZ(T, count, ...) AllocatorNewNoThrowArrayZ(HeapAllocator, &HeapAllocator::Instance, T, count)

#define NoMemProtectHeapNewNoThrow(T, ...) AllocatorNewNoThrow(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), T, \_\_VA\_ARGS\_\_)

#define NoMemProtectHeapNewNoThrowZ(T, ...) AllocatorNewNoThrowZ(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), T, \_\_VA\_ARGS\_\_)

#define NoMemProtectHeapNewNoThrowPlus(size, T, ...) AllocatorNewNoThrowPlus(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), size, T, \_\_VA\_ARGS\_\_)

#define NoMemProtectHeapNewNoThrowPlusZ(size, T, ...) AllocatorNewNoThrowPlusZ(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), size, T, \_\_VA\_ARGS\_\_)

#define NoMemProtectHeapNewNoThrowPlusPrefixZ(size, T, ...) AllocatorNewNoThrowPlusPrefixZ(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), size, T, \_\_VA\_ARGS\_\_)

#define NoMemProtectHeapNewNoThrowStruct(T) AllocatorNewNoThrowStruct(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), T)

#define NoMemProtectHeapNewNoThrowStructZ(T) AllocatorNewNoThrowStructZ(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), T)

#define NoMemProtectHeapNewNoThrowArray(T, count, ...) AllocatorNewNoThrowArray(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), T, count)

#define NoMemProtectHeapNewNoThrowArrayZ(T, count, ...) AllocatorNewNoThrowArrayZ(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), T, count)

#define NoMemProtectHeapDelete(obj) AllocatorDelete(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), obj)

#define NoMemProtectHeapDeletePlus(size, obj) AllocatorDeletePlus(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), size, obj)

#define NoMemProtectHeapDeletePlusPrefix(size, obj) AllocatorDeletePlusPrefix(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), size, obj)

#define NoMemProtectHeapDeleteArray(count, obj) AllocatorDeleteArray(HeapAllocator, HeapAllocator::GetNoMemProtectInstance(), count, obj)

#define NoCheckHeapNew(T, ...) AllocatorNew(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, T, \_\_VA\_ARGS\_\_)

#define NoCheckHeapNewZ(T, ...) AllocatorNewZ(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, T, \_\_VA\_ARGS\_\_)

#define NoCheckHeapNewPlus(size, T, ...) AllocatorNewPlus(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, size, T, \_\_VA\_ARGS\_\_)

#define NoCheckHeapNewPlusZ(size, T, ...) AllocatorNewPlusZ(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, size, T, \_\_VA\_ARGS\_\_)

#define NoCheckHeapNewStruct(T) AllocatorNewStruct(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, T)

#define NoCheckHeapNewStructZ(T) AllocatorNewStructZ(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, T)

#define NoCheckHeapNewArray(T, count, ...) AllocatorNewArray(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, T, count)

#define NoCheckHeapNewArrayZ(T, count, ...) AllocatorNewArrayZ(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, T, count)

#define NoCheckHeapDelete(obj) AllocatorDelete(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, obj)

#define NoCheckHeapDeletePlus(size, obj) AllocatorDeletePlus(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, size, obj)

#define NoCheckHeapDeleteArray(count, obj) AllocatorDeleteArray(NoCheckHeapAllocator, &NoCheckHeapAllocator::Instance, count, obj)

namespace Memory

{

#ifdef HEAP\_TRACK\_ALLOC

struct HeapAllocatorData;

struct HeapAllocRecord

{

HeapAllocRecord \* prev;

HeapAllocRecord \* next;

size\_t allocId;

size\_t size;

TrackAllocData allocData;

HeapAllocatorData\* data;

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

StackBackTrace \* stacktrace;

#endif

};

struct HeapAllocatorData

{

void LogAlloc(HeapAllocRecord \* record, size\_t requestedBytes, TrackAllocData const& data);

void LogFree(HeapAllocRecord \* record);

bool CheckLeaks();

HeapAllocRecord \* head;

size\_t allocCount;

size\_t deleteCount;

size\_t outstandingBytes;

static uint const StackTraceDepth = 10;

};

#endif

struct HeapAllocator

{

static const bool FakeZeroLengthArray = false;

char \* Alloc(size\_t byteSize)

{

return AllocT<false>(byteSize);

}

template <bool noThrow>

char \* AllocT(size\_t byteSize);

// This exists solely to make the AllocateXXX macros more polymorphic

char \* AllocLeaf(size\_t byteSize)

{

return Alloc(byteSize);

}

char \* NoThrowAlloc(size\_t byteSize)

{

return AllocT<true>(byteSize);

}

char \* AllocZero(size\_t byteSize)

{

char \* buffer = Alloc(byteSize);

memset(buffer, 0, byteSize);

return buffer;

}

char \* NoThrowAllocZero(size\_t byteSize)

{

char \* buffer = NoThrowAlloc(byteSize);

if (buffer != nullptr)

{

memset(buffer, 0, byteSize);

}

return buffer;

}

void Free(void \* buffer, size\_t byteSize);

static HeapAllocator Instance;

static HeapAllocator \* GetNoMemProtectInstance();

#ifdef TRACK\_ALLOC

// Doesn't support tracking information, dummy implementation

HeapAllocator \* TrackAllocInfo(TrackAllocData const& data);

void ClearTrackAllocInfo(TrackAllocData\* data = NULL);

#ifdef HEAP\_TRACK\_ALLOC

#ifndef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

~HeapAllocator();

#endif

static void InitializeThread()

{

memset(&nextAllocData, 0, sizeof(nextAllocData));

}

static bool CheckLeaks();

\_\_declspec(thread) static TrackAllocData nextAllocData;

HeapAllocatorData data;

static CriticalSection cs;

#endif

#endif

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

HeapAllocator(bool allowMemProtect = true);

~HeapAllocator();

void FinishMemProtectHeapCollect();

private:

bool DoUseMemProtectHeap();

static HeapAllocator NoMemProtectInstance;

#if DBG

bool isUsed;

bool allocMemProtect;

void \* memProtectHeapHandle;

#endif

#endif

}; // HeapAllocator.

class NoThrowHeapAllocator

{

public:

static const bool FakeZeroLengthArray = false;

char \* Alloc(size\_t byteSize);

char \* AllocZero(size\_t byteSize);

void Free(void \* buffer, size\_t byteSize);

static NoThrowHeapAllocator Instance;

#ifdef TRACK\_ALLOC

// Doesn't support tracking information, dummy implementation

NoThrowHeapAllocator \* TrackAllocInfo(TrackAllocData const& data);

void ClearTrackAllocInfo(TrackAllocData\* data = NULL);

#endif

};

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

class NoThrowNoMemProtectHeapAllocator

{

public:

static const bool FakeZeroLengthArray = false;

char \* Alloc(size\_t byteSize);

char \* AllocZero(size\_t byteSize);

void Free(void \* buffer, size\_t byteSize);

static NoThrowNoMemProtectHeapAllocator Instance;

#ifdef TRACK\_ALLOC

// Doesn't support tracking information, dummy implementation

NoThrowNoMemProtectHeapAllocator \* TrackAllocInfo(TrackAllocData const& data);

void ClearTrackAllocInfo(TrackAllocData\* data = NULL);

#endif

};

#endif

class NoCheckHeapAllocator

{

public:

static const bool FakeZeroLengthArray = false;

char \* Alloc(size\_t byteSize)

{

if (processHeap == NULL)

{

processHeap = GetProcessHeap();

}

char \* buffer = (char\*)HeapAlloc(processHeap, 0, byteSize);

if (buffer == nullptr)

{

// NoCheck heap allocator is only used by debug only code, and if we fail to allocate

// memory, we will just raise an exception and kill the process

DebugHeap\_OOM\_fatal\_error();

}

return buffer;

}

char \* AllocZero(size\_t byteSize)

{

if (processHeap == NULL)

{

processHeap = GetProcessHeap();

}

char \* buffer = (char\*)HeapAlloc(processHeap, HEAP\_ZERO\_MEMORY, byteSize);

if (buffer == nullptr)

{

// NoCheck heap allocator is only used by debug only code, and if we fail to allocate

// memory, we will just raise an exception and kill the process

DebugHeap\_OOM\_fatal\_error();

}

return buffer;

}

void Free(void \* buffer, size\_t byteSize)

{

Assert(processHeap != NULL);

HeapFree(processHeap, 0, buffer);

}

#ifdef TRACK\_ALLOC

// Doesn't support tracking information, dummy implementation

NoCheckHeapAllocator \* TrackAllocInfo(TrackAllocData const& data) { return this; }

void ClearTrackAllocInfo(TrackAllocData\* data = NULL) {}

#endif

static NoCheckHeapAllocator Instance;

static HANDLE processHeap;

};

#ifdef CHECK\_MEMORY\_LEAK

class MemoryLeakCheck

{

public:

MemoryLeakCheck() : head(NULL), tail(NULL), leakedBytes(0), leakedCount(0), enableOutput(true) {}

~MemoryLeakCheck();

static void AddLeakDump(wchar\_t const \* dump, size\_t bytes, size\_t count);

static void SetEnableOutput(bool flag) { leakCheck.enableOutput = flag; }

static bool IsEnableOutput() { return leakCheck.enableOutput; }

private:

static MemoryLeakCheck leakCheck;

struct LeakRecord

{

wchar\_t const \* dump;

LeakRecord \* next;

};

CriticalSection cs;

LeakRecord \* head;

LeakRecord \* tail;

size\_t leakedBytes;

size\_t leakedCount;

bool enableOutput;

};

#endif

} // namespace Memory

#ifdef INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC

//----------------------------------------

// NoThrowNoMemProtectHeapAllocator overrides

//----------------------------------------

template <>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, NoThrowNoMemProtectHeapAllocator \* alloc, char \* (NoThrowNoMemProtectHeapAllocator::\*AllocFunc)(size\_t))

{

return ::operator new(byteSize, alloc, true, AllocFunc);

}

template <>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new[](size\_t byteSize, NoThrowNoMemProtectHeapAllocator \* alloc, char \* (NoThrowNoMemProtectHeapAllocator::\*AllocFunc)(size\_t))

{

return ::operator new[](byteSize, alloc, true, AllocFunc);

}

template <>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, NoThrowNoMemProtectHeapAllocator \* alloc, char \* (NoThrowNoMemProtectHeapAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

return ::operator new(byteSize, alloc, true, AllocFunc, plusSize);

}

inline void \_\_cdecl

operator delete(void \* obj, NoThrowNoMemProtectHeapAllocator \* alloc, char \* (NoThrowNoMemProtectHeapAllocator::\*AllocFunc)(size\_t))

{

alloc->Free(obj, (size\_t)-1);

}

inline void \_\_cdecl

operator delete(void \* obj, NoThrowNoMemProtectHeapAllocator \* alloc, char \* (NoThrowNoMemProtectHeapAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

alloc->Free(obj, (size\_t)-1);

}

#else

typedef NoThrowHeapAllocator NoThrowNoMemProtectHeapAllocator;

#endif

//----------------------------------------

// Default operator new/delete overrides

//----------------------------------------

#if !defined(USED\_IN\_STATIC\_LIB)

\_Ret\_maybenull\_ void \* \_\_cdecl operator new(size\_t byteSize);

\_Ret\_maybenull\_ void \* \_\_cdecl operator new[](size\_t byteSize);

void \_\_cdecl operator delete(void \* obj);

void \_\_cdecl operator delete[](void \* obj);

#endif

//----------------------------------------

// HeapAllocator overrides

//----------------------------------------

inline void \_\_cdecl

operator delete(void \* obj, HeapAllocator \* alloc, char \* (HeapAllocator::\*AllocFunc)(size\_t))

{

alloc->Free(obj, (size\_t)-1);

}

inline void \_\_cdecl

operator delete(void \* obj, HeapAllocator \* alloc, char \* (HeapAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

alloc->Free(obj, (size\_t)-1);

}

//----------------------------------------

// NoThrowHeapAllocator overrides

//----------------------------------------

template <>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, NoThrowHeapAllocator \* alloc, char \* (NoThrowHeapAllocator::\*AllocFunc)(size\_t))

{

return ::operator new(byteSize, alloc, true, AllocFunc);

}

template <>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new[](size\_t byteSize, NoThrowHeapAllocator \* alloc, char \* (NoThrowHeapAllocator::\*AllocFunc)(size\_t))

{

return ::operator new[](byteSize, alloc, true, AllocFunc);

}

template <>

\_Ret\_maybenull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, NoThrowHeapAllocator \* alloc, char \* (NoThrowHeapAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

return ::operator new(byteSize, alloc, true, AllocFunc, plusSize);

}

inline void \_\_cdecl

operator delete(void \* obj, NoThrowHeapAllocator \* alloc, char \* (NoThrowHeapAllocator::\*AllocFunc)(size\_t))

{

alloc->Free(obj, (size\_t)-1);

}

inline void \_\_cdecl

operator delete(void \* obj, NoThrowHeapAllocator \* alloc, char \* (NoThrowHeapAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

alloc->Free(obj, (size\_t)-1);

}

template <>

\_Ret\_notnull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, NoCheckHeapAllocator \* alloc, char \* (NoCheckHeapAllocator::\*AllocFunc)(size\_t))

{

Assert(byteSize != 0);

void \* buffer = (alloc->\*AllocFunc)(byteSize);

return buffer;

}

template <>

\_Ret\_notnull\_ \_\_inline void \* \_\_cdecl

operator new(size\_t byteSize, NoCheckHeapAllocator \* alloc, char \* (NoCheckHeapAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

Assert(byteSize != 0);

Assert(plusSize != 0);

void \* buffer = (alloc->\*AllocFunc)(AllocSizeMath::Add(byteSize, plusSize));

return buffer;

}

\_Ret\_notnull\_ \_\_inline void \* \_\_cdecl

operator new[](size\_t byteSize, NoCheckHeapAllocator \* alloc, char \* (NoCheckHeapAllocator::\*AllocFunc)(size\_t))

{

void \* buffer = (alloc->\*AllocFunc)(byteSize);

return buffer;

}

inline void \_\_cdecl

operator delete(void \* obj, NoCheckHeapAllocator \* alloc, char \* (NoCheckHeapAllocator::\*AllocFunc)(size\_t))

{

alloc->Free(obj, (size\_t)-1);

}

inline void \_\_cdecl

operator delete(void \* obj, NoCheckHeapAllocator \* alloc, char \* (NoCheckHeapAllocator::\*AllocFunc)(size\_t), size\_t plusSize)

{

alloc->Free(obj, (size\_t)-1);

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

//----------------------------------------

// Default operator new/delete overrides

//----------------------------------------

\_Ret\_maybenull\_ void \* \_\_cdecl

operator new(size\_t byteSize)

{

return HeapNewNoThrowArray(char, byteSize);

}

\_Ret\_maybenull\_ void \* \_\_cdecl

operator new[](size\_t byteSize)

{

return HeapNewNoThrowArray(char, byteSize);

}

void \_\_cdecl

operator delete(void \* obj)

{

HeapAllocator::Instance.Free(obj, (size\_t)-1);

}

void \_\_cdecl

operator delete[](void \* obj)

{

HeapAllocator::Instance.Free(obj, (size\_t)-1);

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

template <typename TBlockAttributes>

SmallNormalHeapBlockT<TBlockAttributes> \*

HeapBlock::AsNormalBlock()

{

Assert(this == nullptr || IsAnyNormalBlock());

return static\_cast<SmallNormalHeapBlockT<TBlockAttributes> \*>(this);

}

template <typename TBlockAttributes>

SmallLeafHeapBlockT<TBlockAttributes> \*

HeapBlock::AsLeafBlock()

{

Assert(this == nullptr || IsLeafBlock());

return static\_cast<SmallLeafHeapBlockT<TBlockAttributes> \*>(this);

}

template <typename TBlockAttributes>

SmallFinalizableHeapBlockT<TBlockAttributes> \*

HeapBlock::AsFinalizableBlock()

{

Assert(this == nullptr || IsAnyFinalizableBlock());

return static\_cast<SmallFinalizableHeapBlockT<TBlockAttributes> \*>(this);

}

#ifdef RECYCLER\_WRITE\_BARRIER

template <typename TBlockAttributes>

SmallNormalWithBarrierHeapBlockT<TBlockAttributes> \*

HeapBlock::AsNormalWriteBarrierBlock()

{

Assert(this == nullptr || IsNormalWriteBarrierBlock());

return static\_cast<SmallNormalWithBarrierHeapBlockT<TBlockAttributes> \*>(this);

}

template <typename TBlockAttributes>

SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes> \*

HeapBlock::AsFinalizableWriteBarrierBlock()

{

Assert(this == nullptr || IsFinalizableWriteBarrierBlock());

return static\_cast<SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes> \*>(this);

}

#endif

void

HeapBlock::SetNeedOOMRescan(Recycler \* recycler)

{

Assert(!this->IsLeafBlock());

this->needOOMRescan = true;

recycler->SetNeedOOMRescan();

}

#ifdef RECYCLER\_PAGE\_HEAP

void

HeapBlock::CapturePageHeapAllocStack()

{

Assert(this->InPageHeapMode());

// These asserts are true because explicit free is disallowed in

// page heap mode. If they weren't, we'd have to modify the asserts

Assert(this->pageHeapFreeStack == nullptr);

Assert(this->pageHeapAllocStack == nullptr);

// Note: NoCheckHeapAllocator will fail fast if we can't allocate the stack to capture

// REVIEW: Should we have a flag to configure the number of frames captured?

if (pageHeapAllocStack != nullptr)

{

this->pageHeapAllocStack->Capture(Recycler::s\_numFramesToSkipForPageHeapAlloc);

}

else

{

this->pageHeapAllocStack = StackBackTrace::Capture(&NoCheckHeapAllocator::Instance, Recycler::s\_numFramesToSkipForPageHeapAlloc, Recycler::s\_numFramesToCaptureForPageHeap);

}

}

void

HeapBlock::CapturePageHeapFreeStack()

{

Assert(this->InPageHeapMode());

// These asserts are true because explicit free is disallowed in

// page heap mode. If they weren't, we'd have to modify the asserts

Assert(this->pageHeapFreeStack == nullptr);

Assert(this->pageHeapAllocStack != nullptr);

if (this->pageHeapFreeStack != nullptr)

{

this->pageHeapFreeStack->Capture(Recycler::s\_numFramesToSkipForPageHeapFree);

}

else

{

this->pageHeapFreeStack = StackBackTrace::Capture(&NoCheckHeapAllocator::Instance, Recycler::s\_numFramesToSkipForPageHeapFree, Recycler::s\_numFramesToCaptureForPageHeap);

}

}

#endif

//========================================================================================================

// SmallHeapBlock

//========================================================================================================

template <class TBlockAttributes>

size\_t

SmallHeapBlockT<TBlockAttributes>::GetAllocPlusSize(uint objectCount)

{

// Small Heap Block Layout:

// TrackerData \* [objectCount] (Optional)

// ObjectInfo [objectCount] (In reverse index order)

// <Small\*HeapBlock>

size\_t allocPlusSize = Math::Align<size\_t>(sizeof(unsigned char) \* objectCount, sizeof(size\_t));

#ifdef PROFILE\_RECYCLER\_ALLOC

if (Recycler::DoProfileAllocTracker())

{

allocPlusSize += objectCount \* sizeof(void \*);

}

#endif

return allocPlusSize;

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::ConstructorCommon(HeapBucket \* bucket, ushort objectSize, ushort objectCount, HeapBlockType heapBlockType)

{

this->heapBucket = bucket;

this->Init(objectSize, objectCount);

Assert(heapBlockType < HeapBlock::HeapBlockType::SmallAllocBlockTypeCount + HeapBlock::HeapBlockType::MediumAllocBlockTypeCount);

Assert(objectCount > 1 && objectCount == (this->GetPageCount() \* AutoSystemInfo::PageSize) / objectSize);

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

heapBucket->heapInfo->heapBlockCount[heapBlockType]++;

#endif

if (TBlockAttributes::IsSmallBlock)

{

Assert(heapBlockType < HeapBlockType::SmallAllocBlockTypeCount);

}

else

{

Assert(heapBlockType >= HeapBlockType::SmallAllocBlockTypeCount && heapBlockType < HeapBlockType::SmallBlockTypeCount);

}

DebugOnly(lastUncollectedAllocBytes = 0);

}

template <class TBlockAttributes>

SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockT(HeapBucket \* bucket, ushort objectSize, ushort objectCount, HeapBlockType heapBlockType)

: HeapBlock(heapBlockType),

bucketIndex(HeapInfo::GetBucketIndex(objectSize)),

validPointers(HeapInfo::smallAllocValidPointersMap.GetValidPointersForIndex(HeapInfo::GetBucketIndex(objectSize))),

objectSize(objectSize), objectCount(objectCount)

{

ConstructorCommon(bucket, objectSize, objectCount, heapBlockType);

}

template <>

SmallHeapBlockT<MediumAllocationBlockAttributes>::SmallHeapBlockT(HeapBucket \* bucket, ushort objectSize, ushort objectCount, HeapBlockType heapBlockType)

: HeapBlock((HeapBlockType)(heapBlockType)),

bucketIndex(HeapInfo::GetMediumBucketIndex(objectSize)),

validPointers(HeapInfo::mediumAllocValidPointersMap.GetValidPointersForIndex(HeapInfo::GetMediumBucketIndex(objectSize))),

objectSize(objectSize), objectCount(objectCount)

{

ConstructorCommon(bucket, objectSize, objectCount, heapBlockType);

}

template <class TBlockAttributes>

SmallHeapBlockT<TBlockAttributes>::~SmallHeapBlockT()

{

Assert((this->segment == nullptr && this->address == nullptr) ||

(this->IsLeafBlock()) ||

this->GetPageAllocator(heapBucket->heapInfo->recycler)->IsClosed());

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

heapBucket->heapInfo->heapBlockCount[this->GetHeapBlockType()]--;

heapBucket->heapBlockCount--;

#endif

#ifdef RECYCLER\_PAGE\_HEAP

if (this->pageHeapAllocStack != nullptr)

{

this->pageHeapAllocStack->Delete(&NoCheckHeapAllocator::Instance);

this->pageHeapAllocStack = nullptr;

}

// REVIEW: This means that the old free stack is lost when we get free the heap block

// Is this okay? Should we delay freeing heap blocks till process/thread shutdown time?

if (this->pageHeapFreeStack != nullptr)

{

this->pageHeapFreeStack->Delete(&NoCheckHeapAllocator::Instance);

this->pageHeapFreeStack = nullptr;

}

#endif

}

template <class TBlockAttributes>

uint

SmallHeapBlockT<TBlockAttributes>::GetObjectBitDeltaForBucketIndex(uint bucketIndex)

{

return bucketIndex + 1;

}

template <>

uint

SmallHeapBlockT<MediumAllocationBlockAttributes>::GetObjectBitDeltaForBucketIndex(uint bucketIndex)

{

return HeapInfo::GetObjectSizeForBucketIndex<MediumAllocationBlockAttributes>(bucketIndex) / HeapConstants::ObjectGranularity;

}

// TODO: consider remove and merge with GetPageCount

template <class TBlockAttributes>

template<bool pageheap>

const uint

SmallHeapBlockT<TBlockAttributes>::GetPageHeapModePageCount() const

{

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap)

{

if (InPageHeapMode())

{

return TBlockAttributes::PageCount + 1;

}

}

#endif

return TBlockAttributes::PageCount;

}

template <class TBlockAttributes>

uint

SmallHeapBlockT<TBlockAttributes>::GetPageCount() const

{

return TBlockAttributes::PageCount;

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::ClearObjectInfoList()

{

ushort count = this->objectCount;

// the object info list is prefix to the object

memset(((byte \*)this) - count, 0, count);

}

template <class TBlockAttributes>

byte&

SmallHeapBlockT<TBlockAttributes>::ObjectInfo(uint index)

{

// See SmallHeapBlockT<TBlockAttributes>::GetAllocPlusSize for layout description

// the object info list is prefix to the object and in reverse index order

Assert(index < this->objectCount);

return \*(((byte \*)this) - index - 1);

}

template <class TBlockAttributes>

ushort

SmallHeapBlockT<TBlockAttributes>::GetExpectedFreeObjectCount() const

{

Assert(this->GetRecycler()->IsSweeping());

return objectCount - markCount;

}

template <class TBlockAttributes>

uint

SmallHeapBlockT<TBlockAttributes>::GetExpectedFreeBytes() const

{

return GetExpectedFreeObjectCount() \* objectSize;

}

template <class TBlockAttributes>

ushort

SmallHeapBlockT<TBlockAttributes>::GetExpectedSweepObjectCount() const

{

return GetExpectedFreeObjectCount() - freeCount;

}

#ifdef RECYCLER\_PAGE\_HEAP

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::EnablePageHeap()

{

if (this->heapBucket->IsPageHeapEnabled())

{

this->pageHeapMode = this->heapBucket->heapInfo->pageHeapMode;

}

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::ClearPageHeap()

{

this->pageHeapMode = PageHeapMode::PageHeapModeOff;

}

#endif

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::Init(ushort objectSize, ushort objectCount)

{

Assert(objectCount != 0);

Assert(TBlockAttributes::IsAlignedObjectSize(objectSize));

Assert(this->next == nullptr);

Assert(this->freeObjectList == nullptr);

Assert(this->freeCount == 0);

#ifdef PARTIAL\_GC\_ENABLED

this->oldFreeCount = this->lastFreeCount = this->objectCount;

#endif

this->isPendingConcurrentSweep = false;

Assert(!this->isInAllocator);

Assert(!this->isClearedFromAllocator);

Assert(!this->isIntegratedBlock);

#ifdef RECYCLER\_PAGE\_HEAP

if (this->pageHeapAllocStack != nullptr)

{

this->pageHeapAllocStack->Delete(&NoCheckHeapAllocator::Instance);

this->pageHeapAllocStack = nullptr;

}

// REVIEW: This means that the old free stack is lost when we get reuse the heap block

// Is this okay? Should we never reuse heap blocks in page heap mode?

if (this->pageHeapFreeStack != nullptr)

{

this->pageHeapFreeStack->Delete(&NoCheckHeapAllocator::Instance);

this->pageHeapFreeStack = nullptr;

}

#endif

}

template <class TBlockAttributes>

template<bool pageheap>

BOOL

SmallHeapBlockT<TBlockAttributes>::ReassignPages(Recycler \* recycler)

{

Assert(this->address == nullptr);

Assert(this->segment == nullptr);

PageSegment \* segment;

#ifdef RECYCLER\_PAGE\_HEAP

const PageHeapMode pageHeapModeLocal = this->pageHeapMode;

#else

const PageHeapMode pageHeapModeLocal = PageHeapModeOff;

#endif

char \* address = this->GetPageAllocator(recycler)->AllocPagesPageAligned(this->GetPageHeapModePageCount<pageheap>(), &segment, pageHeapModeLocal);

if (address == NULL)

{

return FALSE;

}

#ifdef PARTIAL\_GC\_ENABLED

recycler->autoHeap.uncollectedNewPageCount += this->GetPageCount();

#endif

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

if (!this->IsLeafBlock()

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_THREAD\_PAGE

&& !this->IsWithBarrier()

#endif

)

{

recycler->VerifyZeroFill(address, AutoSystemInfo::PageSize \* this->GetPageHeapModePageCount<pageheap>());

}

#endif

if (!this->SetPage<pageheap>(address, segment, recycler))

{

this->GetPageAllocator(recycler)->SuspendIdleDecommit();

this->ReleasePages<pageheap>(recycler);

this->GetPageAllocator(recycler)->ResumeIdleDecommit();

return FALSE;

}

RECYCLER\_PERF\_COUNTER\_ADD(FreeObjectSize, this->GetPageCount() \* AutoSystemInfo::PageSize);

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockFreeObjectSize, this->GetPageCount() \* AutoSystemInfo::PageSize);

return TRUE;

}

#ifdef RECYCLER\_PAGE\_HEAP

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::ClearPageHeapState()

{

// If this page has a guard page associated with it,

// restore its access protections

if (this->guardPageAddress != nullptr)

{

Assert(this->InPageHeapMode());

DWORD oldProtectFlags = 0;

BOOL ret = ::VirtualProtect(static\_cast<LPVOID>(this->guardPageAddress), AutoSystemInfo::PageSize, this->guardPageOldProtectFlags, &oldProtectFlags);

Assert(ret == TRUE);

Assert(oldProtectFlags == PAGE\_NOACCESS);

}

}

#endif

template <class TBlockAttributes>

template<bool pageheap>

BOOL

SmallHeapBlockT<TBlockAttributes>::SetPage(\_\_in\_ecount\_pagesize char \* baseAddress, PageSegment \* pageSegment, Recycler \* recycler)

{

char\* address = baseAddress;

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap)

{

if (InPageHeapMode())

{

uint pageCount = this->GetPageHeapModePageCount<true>();

Assert(pageCount == TBlockAttributes::PageCount + 1 || (!this->InPageHeapMode() && pageCount == TBlockAttributes::PageCount));

if (this->pageHeapMode == PageHeapMode::PageHeapModeBlockStart)

{

address = baseAddress + AutoSystemInfo::PageSize;

this->guardPageAddress = baseAddress;

}

else if (this->pageHeapMode == PageHeapMode::PageHeapModeBlockEnd)

{

address = baseAddress;

this->guardPageAddress = baseAddress + (TBlockAttributes::PageCount \* AutoSystemInfo::PageSize);

}

if (this->guardPageAddress != nullptr)

{

if (::VirtualProtect(static\_cast<LPVOID>(this->guardPageAddress), AutoSystemInfo::PageSize, PAGE\_NOACCESS, &guardPageOldProtectFlags) == FALSE)

{

return FALSE;

}

}

}

}

#endif

uint l2Id = HeapBlockMap32::GetLevel2Id(address);

Assert(l2Id + (TBlockAttributes::PageCount - 1) < 256);

this->segment = pageSegment;

this->address = address;

// Set up the page to have nothing is free

Assert(this->freeObjectList == nullptr);

Assert(this->IsFreeBitsValid());

Assert(this->freeCount == 0);

Assert(this->freeCount == this->GetFreeBitVector()->Count());

Assert(this->objectCount == this->lastFreeCount);

Assert(this->explicitFreeBits.Count() == 0);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(recycler->IsConcurrentMarkState() || !recycler->IsMarkState() || recycler->IsCollectionDisabled());

#else

Assert(!recycler->IsMarkState() || recycler->IsCollectionDisabled());

#endif

Assert(this->bucketIndex <= 0xFF);

// We use the block type directly here, without the getter so that we can tell on the heap block map,

// whether the block is a medium block or not

if (!recycler->heapBlockMap.SetHeapBlock(this->address, this->GetPageCount(), this, this->heapBlockType, (byte)this->bucketIndex))

{

return FALSE;

}

// Retrieve pointer to mark bits for this block and store it locally.

// Note, mark bits aren't guaranteed to exist until after we register with HBM.

this->markBits = recycler->heapBlockMap.GetMarkBitVectorForPages<TBlockAttributes::BitVectorCount>(this->address);

Assert(this->markBits);

#if defined(\_M\_ARM32\_OR\_ARM64)

// We need to ensure that the above writes to the SmallHeapBlock are visible to the background GC thread.

// In particular, see Threshold 331596 -- we were seeing an old value for SmallHeapBlockT<TBlockAttributes>::markBits in ResetMarks.

// which caused the bit vector Copy operation there to AV.

// See also SmallHeapBlockT<TBlockAttributes>::ResetMarks.

MemoryBarrier();

#endif

return TRUE;

}

template <class TBlockAttributes>

template<bool pageheap>

void

SmallHeapBlockT<TBlockAttributes>::ReleasePages(Recycler \* recycler)

{

Assert(recycler->collectionState != CollectionStateMark);

Assert(segment != nullptr);

Assert(address != nullptr);

#if DBG

if (this->IsLeafBlock())

{

RecyclerVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Releasing leaf block pages at address 0x%p\n", address);

}

#endif

char\* address = this->address;

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap)

{

if (InPageHeapMode())

{

ClearPageHeapState();

if (guardPageAddress != nullptr)

{

if (this->pageHeapMode == PageHeapMode::PageHeapModeBlockStart)

{

address = guardPageAddress;

}

guardPageAddress = nullptr;

}

}

}

#endif

#ifdef RECYCLER\_FREE\_MEM\_FILL

memset(address, DbgMemFill, AutoSystemInfo::PageSize \* this->GetPageHeapModePageCount<pageheap>());

#endif

this->GetPageAllocator(recycler)->ReleasePages(address, this->GetPageHeapModePageCount<pageheap>(), this->GetPageSegment());

this->segment = nullptr;

this->address = nullptr;

}

template <class TBlockAttributes>

template<bool pageheap>

void

SmallHeapBlockT<TBlockAttributes>::BackgroundReleasePagesSweep(Recycler\* recycler)

{

recycler->heapBlockMap.ClearHeapBlock(address, this->GetPageCount());

char\* address = this->address;

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap)

{

if (InPageHeapMode())

{

ClearPageHeapState();

if (guardPageAddress != nullptr)

{

if (this->pageHeapMode == PageHeapMode::PageHeapModeBlockStart)

{

address = guardPageAddress;

}

guardPageAddress = nullptr;

}

}

}

#endif

this->GetPageAllocator(recycler)->BackgroundReleasePages(address, this->GetPageHeapModePageCount<pageheap>(), this->GetPageSegment());

this->address = nullptr;

this->segment = nullptr;

this->Reset();

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::ReleasePagesShutdown(Recycler \* recycler)

{

#if DBG

if (this->IsLeafBlock())

{

RecyclerVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Releasing leaf block pages at address 0x%p\n", address);

}

#ifdef RECYCLER\_PAGE\_HEAP

ClearPageHeapState();

#endif

RemoveFromHeapBlockMap(recycler);

// Don't release the page in shut down, the page allocator will release them faster

// Leaf block's allocator need not be closed

Assert(this->IsLeafBlock() || this->GetPageAllocator(recycler)->IsClosed());

#endif

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::RemoveFromHeapBlockMap(Recycler\* recycler)

{

recycler->heapBlockMap.ClearHeapBlock(address, this->GetPageCount());

}

template <class TBlockAttributes>

template<bool pageheap>

void

SmallHeapBlockT<TBlockAttributes>::ReleasePagesSweep(Recycler \* recycler)

{

RemoveFromHeapBlockMap(recycler);

ReleasePages<pageheap>(recycler);

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::Reset()

{

this->GetFreeBitVector()->ClearAll();

this->freeCount = 0;

this->markCount = 0;

#ifdef PARTIAL\_GC\_ENABLED

this->oldFreeCount = this->lastFreeCount = this->objectCount;

#endif

this->freeObjectList = nullptr;

this->lastFreeObjectHead = nullptr;

this->ClearObjectInfoList();

this->isInAllocator = false;

#if DBG || defined(RECYCLER\_STATS)

this->GetDebugFreeBitVector()->ClearAll();

#endif

#if DBG

this->isClearedFromAllocator = false;

this->isIntegratedBlock = false;

#endif

#ifdef RECYCLER\_PAGE\_HEAP

if (this->pageHeapFreeStack != nullptr)

{

this->pageHeapFreeStack->Delete(&NoCheckHeapAllocator::Instance);

this->pageHeapFreeStack = nullptr;

}

if (this->pageHeapAllocStack != nullptr)

{

this->pageHeapAllocStack->Delete(&NoCheckHeapAllocator::Instance);

this->pageHeapAllocStack = nullptr;

}

#endif

// There is no page associated with this heap block,

// and therefore we should have no mark bits either

this->markBits = nullptr;

Assert(this->explicitFreeBits.Count() == 0);

}

// Map any object address to it's object index within the heap block

template <class TBlockAttributes>

ushort

SmallHeapBlockT<TBlockAttributes>::GetAddressIndex(void \* objectAddress)

{

Assert(objectAddress >= address && objectAddress < this->GetEndAddress());

Assert(HeapInfo::IsAlignedAddress(objectAddress));

Assert(HeapInfo::IsAlignedAddress(address));

unsigned int offset = (unsigned int)((char\*)objectAddress - address);

offset = offset >> HeapConstants::ObjectAllocationShift;

ushort index = validPointers.GetAddressIndex(offset);

Assert(index == SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit ||

index <= TBlockAttributes::MaxAddressBit);

return index;

}

template <class TBlockAttributes>

typename SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector const\*

SmallHeapBlockT<TBlockAttributes>::GetInvalidBitVector()

{

return HeapInfo::GetInvalidBitVector<TBlockAttributes>(objectSize);

}

template <class TBlockAttributes>

typename SmallHeapBlockT<TBlockAttributes>::BlockInfo const\*

SmallHeapBlockT<TBlockAttributes>::GetBlockInfo()

{

return HeapInfo::GetBlockInfo<TBlockAttributes>(objectSize);

}

template <class TBlockAttributes>

ushort

SmallHeapBlockT<TBlockAttributes>::GetInteriorAddressIndex(void \* interiorAddress)

{

Assert(interiorAddress >= address && interiorAddress < this->GetEndAddress());

Assert(HeapInfo::IsAlignedAddress(address));

unsigned int offset = (unsigned int)((char\*)interiorAddress - address);

offset = offset >> HeapConstants::ObjectAllocationShift;

ushort index = validPointers.GetInteriorAddressIndex(offset);

Assert(index == SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit ||

index <= TBlockAttributes::MaxAddressBit);

return index;

}

template <class TBlockAttributes>

BOOL

SmallHeapBlockT<TBlockAttributes>::IsInFreeObjectList(void \* objectAddress)

{

FreeObject \* freeObject = this->freeObjectList;

while (freeObject != nullptr)

{

if (freeObject == objectAddress)

{

return true;

}

freeObject = freeObject->GetNext();

}

return false;

}

template <class TBlockAttributes>

template <typename TBlockType>

bool

SmallHeapBlockT<TBlockAttributes>::FindHeapObjectImpl(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject)

{

if (flags & FindHeapObjectFlags\_AllowInterior)

{

objectAddress = (void\*) this->GetRealAddressFromInterior(objectAddress);

if (objectAddress == nullptr)

{

return false;

}

}

ushort index = GetAddressIndex(objectAddress);

Assert(index != SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit);

if (index == SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit)

{

return false;

}

// If we have pending object, we still need to check the free bit if the caller requested the attribute to be correct

bool const disableCheck = ((flags & FindHeapObjectFlags\_NoFreeBitVerify) != 0) ||

((flags & FindHeapObjectFlags\_VerifyFreeBitForAttribute) != 0 && !this->HasPendingDisposeObjects());

if (!disableCheck)

{

// REVIEW: Checking if an object if free is strictly not necessary

// In all case, we should have a valid object, For memory protect heap, this is just to make sure we don't

// free pointers that are invalid.

if (recycler->IsConcurrentSweepExecutingState())

{

// TODO: unless we know the state of the heap block, we don't know.

// skip the check for now.

}

else

{

if (flags & FindHeapObjectFlags\_ClearedAllocators)

{

// Heap enum has some case where it allocates, so we can't assert

Assert(((HeapBucketT<TBlockType> \*)this->heapBucket)->AllocatorsAreEmpty() || recycler->isHeapEnumInProgress);

}

else if (this->IsInAllocator())

{

((HeapBucketT<TBlockType> \*)this->heapBucket)->UpdateAllocators();

}

// REVIEW allocation heuristics

if (this->EnsureFreeBitVector()->Test(this->GetObjectBitDelta() \* index))

{

return false;

}

}

}

byte& attributes = ObjectInfo(index);

heapObject = RecyclerHeapObjectInfo(objectAddress, recycler, this, &attributes);

return true;

}

template <class TBlockAttributes>

BOOL

SmallHeapBlockT<TBlockAttributes>::IsValidObject(void\* objectAddress)

{

if (objectAddress < this->GetAddress() || objectAddress >= this->GetEndAddress())

{

return false;

}

ushort index = GetAddressIndex(objectAddress);

if (index == SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit)

{

return false;

}

#if DBG

return !this->GetDebugFreeBitVector()->Test(GetAddressBitIndex(objectAddress));

#else

return true;

#endif

}

template <class TBlockAttributes>

bool

SmallHeapBlockT<TBlockAttributes>::IsInAllocator() const

{

return isInAllocator;

}

template <class TBlockAttributes>

bool

SmallHeapBlockT<TBlockAttributes>::HasPendingDisposeObjects()

{

return this->IsAnyFinalizableBlock() && this->AsFinalizableBlock<TBlockAttributes>()->HasPendingDisposeObjects();

}

template <class TBlockAttributes>

bool

SmallHeapBlockT<TBlockAttributes>::HasAnyDisposeObjects()

{

return this->IsAnyFinalizableBlock() && this->AsFinalizableBlock<TBlockAttributes>()->HasAnyDisposeObjects();

}

template <class TBlockAttributes>

Recycler \*

SmallHeapBlockT<TBlockAttributes>::GetRecycler() const

{

#if DBG

return this->heapBucket->heapInfo->recycler;

#else

return nullptr;

#endif

}

#if DBG

template <class TBlockAttributes>

BOOL

SmallHeapBlockT<TBlockAttributes>::IsFreeObject(void \* objectAddress)

{

if (objectAddress < this->GetAddress() || objectAddress >= this->GetEndAddress())

{

return false;

}

ushort index = GetAddressIndex(objectAddress);

if (index == SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit)

{

return false;

}

return this->GetDebugFreeBitVector()->Test(GetAddressBitIndex(objectAddress));

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::VerifyMarkBitVector()

{

this->GetRecycler()->heapBlockMap.VerifyMarkCountForPages<TBlockAttributes::BitVectorCount>(this->address, TBlockAttributes::PageCount);

}

template <class TBlockAttributes>

bool

SmallHeapBlockT<TBlockAttributes>::IsClearedFromAllocator() const

{

return isClearedFromAllocator;

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::SetIsClearedFromAllocator(bool value)

{

isClearedFromAllocator = value;

}

#endif

template <class TBlockAttributes>

byte \*

SmallHeapBlockT<TBlockAttributes>::GetRealAddressFromInterior(void \* interiorAddress)

{

Assert(interiorAddress >= this->address && interiorAddress < this->address + AutoSystemInfo::PageSize \* this->GetPageCount());

ushort index = GetInteriorAddressIndex(interiorAddress);

if (index != SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit)

{

return (byte \*)this->address + index \* this->GetObjectSize();

}

return nullptr;

}

template <class TBlockAttributes>

bool

SmallHeapBlockT<TBlockAttributes>::TestObjectMarkedBit(void\* objectAddress)

{

Assert(this->address != nullptr);

Assert(this->segment != nullptr);

uint bitIndex = GetAddressBitIndex(objectAddress);

Assert(IsValidBitIndex(bitIndex));

return this->GetMarkedBitVector()->Test(bitIndex) != 0;

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::SetObjectMarkedBit(void\* objectAddress)

{

Assert(this->address != nullptr);

Assert(this->segment != nullptr);

uint bitIndex = GetAddressBitIndex(objectAddress);

Assert(IsValidBitIndex(bitIndex));

this->GetMarkedBitVector()->Set(bitIndex);

}

#ifdef RECYCLER\_MEMORY\_VERIFY

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::SetExplicitFreeBitForObject(void\* objectAddress)

{

Assert(this->address != nullptr);

Assert(this->segment != nullptr);

uint bitIndex = GetAddressBitIndex(objectAddress);

Assert(IsValidBitIndex(bitIndex));

BOOLEAN wasSet = this->explicitFreeBits.TestAndSet(bitIndex);

Assert(!wasSet);

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::ClearExplicitFreeBitForObject(void\* objectAddress)

{

Assert(this->address != nullptr);

Assert(this->segment != nullptr);

uint bitIndex = GetAddressBitIndex(objectAddress);

Assert(IsValidBitIndex(bitIndex));

BOOLEAN wasSet = this->explicitFreeBits.TestAndClear(bitIndex);

Assert(wasSet);

}

#endif

#ifdef RECYCLER\_PAGE\_HEAP

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::VerifyPageHeapAllocation(\_In\_ char\* allocation, PageHeapMode mode)

{

if (mode == PageHeapMode::PageHeapModeBlockStart)

{

Assert(allocation == this->address);

}

else

{

Assert(mode == PageHeapMode::PageHeapModeBlockEnd);

char\* lastObjectAddress = this->GetAddress() + ((this->GetObjectCount() - 1) \* this->objectSize);

Assert(lastObjectAddress <= this->GetEndAddress() - this->objectSize);

}

}

#endif

#ifdef RECYCLER\_VERIFY\_MARK

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::VerifyMark()

{

Assert(!this->needOOMRescan);

SmallHeapBlockBitVector \* marked = this->GetMarkedBitVector();

SmallHeapBlockBitVector tempFreeBits;

this->BuildFreeBitVector(&tempFreeBits);

SmallHeapBlockBitVector \* free = &tempFreeBits;

SmallHeapBlockBitVector const \* invalid = this->GetInvalidBitVector();

uint objectWordCount = this->GetObjectWordCount();

Recycler \* recycler = this->heapBucket->heapInfo->recycler;

FOREACH\_BITSET\_IN\_FIXEDBV(bitIndex, marked)

{

if (!free->Test(bitIndex) && !invalid->Test(bitIndex))

{

Assert(IsValidBitIndex(bitIndex));

uint objectIndex = GetObjectIndexFromBitIndex((ushort)bitIndex);

Assert((this->ObjectInfo(objectIndex) & NewTrackBit) == 0);

// NOTE: We can't verify mark for software write barrier blocks, because they may have

// non-pointer updates that don't trigger the write barrier, but still look like a false reference.

// Thus, when we get here, we'll see a false reference that isn't marked.

// Since this situation is hard to detect, just don't verify mark for write barrier blocks.

// We could fix this if we had object layout info.

if (!this->IsLeafBlock()

#ifdef RECYCLER\_WRITE\_BARRIER

&& !this->IsWithBarrier()

#endif

)

{

if ((ObjectInfo(objectIndex) & LeafBit) == 0)

{

char \* objectAddress = this->address + objectIndex \* objectSize;

for (uint i = 0; i < objectWordCount; i++)

{

void\* target = \*(void\*\*) objectAddress;

recycler->VerifyMark(target);

objectAddress += sizeof(void \*);

}

}

}

}

}

NEXT\_BITSET\_IN\_FIXEDBV;

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::VerifyMark(void \* objectAddress)

{

// Because we mark through new object, we might have a false reference

// somewhere that we have scanned before this new block is allocated

// so the object will not be marked even though it looks like a reference

// Can't verify when the block is new

if (this->heapBucket->GetRecycler()->heapBlockMap.IsAddressInNewChunk(objectAddress))

{

return;

}

ushort bitIndex = GetAddressBitIndex(objectAddress);

#if DBG

Assert(this->GetMarkedBitVector()->Test(bitIndex));

#else

if (!this->GetMarkedBitVector()->Test(bitIndex))

{

DebugBreak();

}

#endif

}

#endif

#ifdef RECYCLER\_STRESS

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::InduceFalsePositive(Recycler \* recycler)

{

// Induce a false positive mark by marking the first object on the free list, if any.

// Note that if the block is in the allocator, freeObjectList is not up to date.

// So we may be marking an already-allocated block, but that's okay --

// we call TryMark so that normal processing (including tracked object processing, etc)

// will occur just as if we had a false reference to this object previously.

void \* falsePositive = this->freeObjectList;

if (falsePositive != nullptr)

{

recycler->TryMarkNonInterior(falsePositive, nullptr);

}

}

#endif

#ifdef PARTIAL\_GC\_ENABLED

template <class TBlockAttributes>

bool

SmallHeapBlockT<TBlockAttributes>::DoPartialReusePage(RecyclerSweep const& recyclerSweep, uint& expectFreeByteCount)

{

// Partial GC page reuse heuristic

#ifdef RECYCLER\_PAGE\_HEAP

// we should not get here in page heap mode.

Assert(!this->InPageHeapMode());

#endif

Assert(recyclerSweep.InPartialCollectMode());

expectFreeByteCount = GetExpectedFreeBytes();

// PartialCollectSmallHeapBlockReuseMinFreeBytes is calculated by dwPageSize\* efficacy. If efficacy is

// high (== 1), and dwPageSize % objectSize != 0, all the pages in the bucket will be partial, and that

// could increase in thread sweep time.

// OTOH, if the object size is really large, the calculation below will reduce the chance for a page to be

// partial. we might need to watch out for that.

return (expectFreeByteCount + objectSize >= recyclerSweep.GetPartialCollectSmallHeapBlockReuseMinFreeBytes());

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::ClearAllAllocBytes()

{

this->oldFreeCount = this->lastFreeCount = this->freeCount;

}

#if DBG

// do debug assert for partial block that we are not going to sweep

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::SweepVerifyPartialBlock(Recycler \* recycler)

{

Assert(!this->IsLeafBlock());

// nothing in the partialHeapBlockList is sweepable

Assert(GetExpectedSweepObjectCount() == 0);

}

#endif

template <class TBlockAttributes>

uint

SmallHeapBlockT<TBlockAttributes>::GetAndClearUnaccountedAllocBytes()

{

Assert(this->lastFreeCount >= this->freeCount);

const ushort currentFreeCount = this->freeCount;

uint unaccountedAllocBytes = (this->lastFreeCount - currentFreeCount) \* this->objectSize;

this->lastFreeCount = currentFreeCount;

return unaccountedAllocBytes;

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::AdjustPartialUncollectedAllocBytes(RecyclerSweep& recyclerSweep, uint const expectSweepCount)

{

const uint allObjectCount = this->objectCount;

const ushort currentFreeCount = this->freeCount;

Assert(this->lastFreeCount == currentFreeCount);

uint newAllocatedCount = this->oldFreeCount - currentFreeCount;

this->oldFreeCount = currentFreeCount;

uint newObjectExpectSweepCount = expectSweepCount;

#ifdef CONCURRENT\_GC\_ENABLED

if (expectSweepCount != 0 && !recyclerSweep.InPartialCollect())

{

// We don't know which objects that we are going sweep are old and which object are new

// So just assume one way or the other by the amount of old vs. new object in the block

const uint allocatedObjectCount = allObjectCount - currentFreeCount;

Assert(allocatedObjectCount >= newAllocatedCount);

const uint oldObjectCount = allocatedObjectCount - newAllocatedCount;

if (oldObjectCount < newAllocatedCount)

{

// count all of the swept object as new, but don't exceed the amount we allocated

if (newObjectExpectSweepCount > newAllocatedCount)

{

newObjectExpectSweepCount = newAllocatedCount;

}

}

else

{

// count all of the swept object as old

newObjectExpectSweepCount = 0;

}

}

#endif

// The page can be old, or it is full (where we set lastFreeCount to 0)

// Otherwise, the newly allocated count must be bigger then the expect sweep count

Assert(newAllocatedCount >= newObjectExpectSweepCount);

Assert(this->lastUncollectedAllocBytes >= newObjectExpectSweepCount \* this->objectSize);

recyclerSweep.SubtractSweepNewObjectAllocBytes(newObjectExpectSweepCount \* this->objectSize);

}

#endif

#ifdef RECYCLER\_PAGE\_HEAP

template <class TBlockAttributes>

char \*

SmallHeapBlockT<TBlockAttributes>::GetPageHeapObjectAddress()

{

Assert(InPageHeapMode());

if (this->pageHeapMode == PageHeapMode::PageHeapModeBlockStart)

{

return this->address;

}

else if (this->pageHeapMode == PageHeapMode::PageHeapModeBlockEnd)

{

return this->address + (this->objectCount - 1) \* objectSize;

}

else

{

AssertMsg(false, "Unknown PageHeapMode");

return nullptr;

}

}

#endif

template <class TBlockAttributes>

template <bool pageheap>

uint

SmallHeapBlockT<TBlockAttributes>::GetMarkCountForSweep()

{

Assert(IsFreeBitsValid());

// Determine the number of marked, non-free objects in the block.

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap && InPageHeapMode())

{

uint bitIndex = this->GetAddressBitIndex(this->GetPageHeapObjectAddress());

bool marked = (this->GetMarkedBitVector()->Test(bitIndex) && !this->GetFreeBitVector()->Test(bitIndex));

return marked ? 1 : 0;

}

Assert(!InPageHeapMode());

#endif

// Make a local copy of mark bits, so we don't modify the actual mark bits.

SmallHeapBlockBitVector temp;

temp.Copy(this->GetMarkedBitVector());

// Remove any invalid bits that may have been set

temp.Minus(this->GetInvalidBitVector());

// Remove the mark bit for things that are still free

if (this->freeCount != 0)

{

temp.Minus(this->GetFreeBitVector());

}

return temp.Count();

}

template <class TBlockAttributes>

template <bool pageheap>

SweepState

SmallHeapBlockT<TBlockAttributes>::Sweep(RecyclerSweep& recyclerSweep, bool queuePendingSweep, bool allocable, ushort finalizeCount, bool hasPendingDispose)

{

Assert(this->address != nullptr);

Assert(this->segment != nullptr);

Assert(!this->isPendingConcurrentSweep);

DebugOnly(VerifyMarkBitVector());

if (allocable)

{

// This block has been allocated from since the last GC.

// We need to update its free bit vector so we can use it below.

Assert(freeCount == this->GetFreeBitVector()->Count());

#ifdef PARTIAL\_GC\_ENABLED

Assert(this->lastFreeCount == 0 || this->oldFreeCount == this->lastFreeCount);

#endif

this->EnsureFreeBitVector();

Assert(this->lastFreeCount >= this->freeCount);

#ifdef PARTIAL\_GC\_ENABLED

Assert(this->oldFreeCount >= this->freeCount);

#endif

#ifdef PARTIAL\_GC\_ENABLED

// Accounting for partial heuristics

recyclerSweep.AddUnaccountedNewObjectAllocBytes(this);

#endif

}

Assert(this->freeCount == this->GetFreeBitVector()->Count());

RECYCLER\_SLOW\_CHECK(CheckFreeBitVector(true));

const uint localMarkCount = this->GetMarkCountForSweep<pageheap>();

this->markCount = (ushort)localMarkCount;

Assert(markCount <= objectCount - this->freeCount);

const uint expectFreeCount = objectCount - localMarkCount;

Assert(expectFreeCount >= this->freeCount);

const uint expectSweepCount = expectFreeCount - this->freeCount;

Assert(!this->IsLeafBlock() || finalizeCount == 0);

Recycler \* recycler = recyclerSweep.GetRecycler();

RECYCLER\_STATS\_INC(recycler, heapBlockCount[this->GetHeapBlockType()]);

#ifdef PARTIAL\_GC\_ENABLED

if (recyclerSweep.DoAdjustPartialHeuristics() && allocable)

{

this->AdjustPartialUncollectedAllocBytes(recyclerSweep, expectSweepCount);

}

DebugOnly(this->lastUncollectedAllocBytes = 0);

#endif

bool noRealObjectsMarked = (localMarkCount == 0);

const bool isAllFreed = (finalizeCount == 0 && noRealObjectsMarked && !hasPendingDispose);

if (isAllFreed)

{

recycler->NotifyFree<pageheap>(this);

Assert(!this->HasPendingDisposeObjects());

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap)

{

if (InPageHeapMode())

{

PageHeapVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Heap block 0x%p is empty\n", this);

}

}

#endif

return SweepStateEmpty;

}

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap)

{

if (InPageHeapMode())

{

PageHeapVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Heap block 0x%p is not empty, local mark count is %d, expected sweep count is %d\n", this, localMarkCount, expectSweepCount);

}

}

#endif

RECYCLER\_STATS\_ADD(recycler, heapBlockFreeByteCount[this->GetHeapBlockType()], expectFreeCount \* this->objectSize);

Assert(!hasPendingDispose || (this->freeCount != 0));

SweepState state = SweepStateSwept;

if (hasPendingDispose)

{

state = SweepStatePendingDispose;

}

if (expectSweepCount == 0)

{

// nothing has been freed

return (this->freeCount == 0) ? SweepStateFull : state;

}

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap)

{

if (InPageHeapMode())

{

// If a real object (either the one at the beginning of the page or at

// the end of the page) was marked, then this block is full

if (noRealObjectsMarked == false)

{

Assert(localMarkCount == 1);

return SweepStateFull;

}

}

}

#endif

RECYCLER\_STATS\_INC(recycler, heapBlockSweptCount[this->GetHeapBlockType()]);

// We need to sweep in thread if there are any finalizable object.

// So that the PrepareFinalize() can be called before concurrent sweep

// and other finalizer. This gives the object an opportunity before any

// other script can be ran to clean up their references/states that are not

// valid since we determine the object is not live any more.

//

// An example is the ITrackable's tracking alias. The reference to the alias

// object needs to be clear so that the reference will not be given out again

// in other script during concurrent sweep or finalizer called before.

#ifdef CONCURRENT\_GC\_ENABLED

if (queuePendingSweep)

{

Assert(finalizeCount == 0);

Assert(!this->HasPendingDisposeObjects());

recyclerSweep.SetHasPendingSweepSmallHeapBlocks();

RECYCLER\_STATS\_INC(recycler, heapBlockConcurrentSweptCount[this->GetHeapBlockType()]);

// This heap block has objects that need to be swept concurrently.

this->isPendingConcurrentSweep = true;

return SweepStatePendingSweep;

}

#else

Assert(!recyclerSweep.IsBackground());

#endif

SweepObjects<pageheap, SweepMode\_InThread>(recycler);

if (HasPendingDisposeObjects())

{

Assert(finalizeCount != 0);

return SweepStatePendingDispose;

}

// Already swept, no more work to be done. Put it back to the queue

return state;

}

#if DBG

template <class TBlockAttributes>

uint

SmallHeapBlockT<TBlockAttributes>::GetMarkCountOnHeapBlockMap() const

{

uint heapBlockMapMarkCount = 0;

char\* startPage = this->GetAddress();

char\* endPage = this->GetEndAddress();

const HeapBlockMap& blockMap = this->GetRecycler()->heapBlockMap;

for (char\* page = startPage; page < endPage; page += AutoSystemInfo::PageSize)

{

heapBlockMapMarkCount += blockMap.GetPageMarkCount(page);

}

return heapBlockMapMarkCount;

}

#endif

template <class TBlockAttributes>

template <bool pageheap, SweepMode mode>

void

SmallHeapBlockT<TBlockAttributes>::SweepObjects(Recycler \* recycler)

{

Assert(mode == SweepMode\_InThread || this->isPendingConcurrentSweep);

Assert(mode == SweepMode\_InThread || !this->IsAnyFinalizableBlock());

Assert(this->IsFreeBitsValid());

Assert(this->markCount != 0 || this->isForceSweeping || this->IsAnyFinalizableBlock());

Assert(this->markCount == this->GetMarkCountForSweep<pageheap>());

DebugOnly(VerifyMarkBitVector());

SmallHeapBlockBitVector \* marked = this->GetMarkedBitVector();

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap && this->InPageHeapMode())

{

// Page heap blocks are always swept in thread.

Assert(mode == SweepMode\_InThread);

Assert(!this->isPendingConcurrentSweep);

char \* objectAddress = this->GetPageHeapObjectAddress();

uint bitIndex = this->GetAddressBitIndex(objectAddress);

uint objectIndex = this->GetAddressIndex(objectAddress);

// We know that the object is not live, or we wouldn't be here.

Assert(this->markCount == 0);

// However, it's possible that the PageHeap object is already free.

// This can happen when the block was in PendingDispose state, and then

// we did the dispose and freed the object in TransferDisposeObjects.

// So check for this.

if (this->GetFreeBitVector()->Test(bitIndex))

{

Assert(this->IsAnyFinalizableBlock());

Assert(this->freeCount == 1);

}

else

{

Assert(!marked->Test(bitIndex));

Assert((this->ObjectInfo(objectIndex) & ImplicitRootBit) == 0);

FreeObject\* addr = (FreeObject\*)objectAddress;

recycler->NotifyFree((char \*)addr, this->objectSize);

SweepObject<SweepMode\_InThread>(recycler, objectIndex, addr);

// Update the free bit vector

Assert(this->freeCount == 0);

this->GetFreeBitVector()->Set(bitIndex);

#ifdef PARTIAL\_GC\_ENABLED

this->oldFreeCount = this->lastFreeCount = this->freeCount = 1;

#endif

this->lastFreeObjectHead = this->freeObjectList;

}

}

else

#endif

{

#ifdef RECYCLER\_PAGE\_HEAP

Assert(!this->InPageHeapMode());

#endif

DebugOnly(const uint expectedSweepCount = objectCount - freeCount - markCount);

Assert(expectedSweepCount != 0 || this->isForceSweeping);

DebugOnly(uint sweepCount = 0);

const uint localSize = objectSize;

const uint localObjectCount = objectCount;

const char\* objectAddress = address;

uint objectBitDelta = this->GetObjectBitDelta();

for (uint objectIndex = 0, bitIndex = 0; objectIndex < localObjectCount; objectIndex++, bitIndex += objectBitDelta)

{

Assert(IsValidBitIndex(bitIndex));

RECYCLER\_STATS\_ADD(recycler, objectSweepScanCount, !isForceSweeping);

if (!marked->Test(bitIndex))

{

if (!this->GetFreeBitVector()->Test(bitIndex))

{

Assert((this->ObjectInfo(objectIndex) & ImplicitRootBit) == 0);

FreeObject\* addr = (FreeObject\*)objectAddress;

#ifdef PARTIAL\_GC\_ENABLED

if (mode != SweepMode\_ConcurrentPartial)

#endif

{

// Don't call NotifyFree if we are doing a partial sweep.

// Since we are not actually collecting the object, we will do the NotifyFree later

// when the object is actually collected in a future Sweep.

recycler->NotifyFree((char \*)addr, this->objectSize);

}

#if DBG

sweepCount++;

#endif

SweepObject<mode>(recycler, objectIndex, addr);

}

}

#if DBG

if (marked->Test(bitIndex))

{

Assert((ObjectInfo(objectIndex) & NewTrackBit) == 0);

}

#endif

objectAddress += localSize;

}

Assert(sweepCount == expectedSweepCount);

this->isPendingConcurrentSweep = false;

#ifdef PARTIAL\_GC\_ENABLED

if (mode == SweepMode\_ConcurrentPartial)

{

Assert(recycler->inPartialCollectMode);

// We didn't actually collect anything, so the free bit vector should still be valid.

Assert(IsFreeBitsValid());

}

else

#endif

{

// Update the free bit vector

// Need to update even if there are not swept object because finalizable object are

// consider freed but not on the free list.

#ifdef PARTIAL\_GC\_ENABLED

ushort currentFreeCount = GetExpectedFreeObjectCount();

#endif

this->GetFreeBitVector()->OrComplimented(marked);

this->GetFreeBitVector()->Minus(this->GetInvalidBitVector());

#ifdef PARTIAL\_GC\_ENABLED

this->oldFreeCount = this->lastFreeCount = this->freeCount = currentFreeCount;

#endif

this->lastFreeObjectHead = this->freeObjectList;

}

}

RECYCLER\_SLOW\_CHECK(CheckFreeBitVector(true));

// The count of marked, non-free objects should still be the same

Assert(this->markCount == this->GetMarkCountForSweep<pageheap>());

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::EnqueueProcessedObject(FreeObject \*\* list, void\* objectAddress, uint index)

{

Assert(GetAddressIndex(objectAddress) == index);

Assert(index != SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit);

Assert(this->objectCount != 1);

#if DBG || defined(RECYCLER\_STATS)

if (list == &this->freeObjectList)

{

BOOL isSet = this->GetDebugFreeBitVector()->TestAndSet(GetAddressBitIndex(objectAddress));

Assert(!isSet);

}

#endif

FillFreeMemory(objectAddress, objectSize);

FreeObject \* freeObject = (FreeObject \*)objectAddress;

freeObject->SetNext(\*list);

\*list = freeObject;

// clear the attributes so that when we are allocating a leaf, we don't have to set the attribute

this->ObjectInfo(index) = 0;

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::EnqueueProcessedObject(FreeObject \*\* list, FreeObject \*\* tail, void\* objectAddress, uint index)

{

if (\*tail == nullptr)

{

Assert(\*list == nullptr);

\*tail = (FreeObject \*)objectAddress;

}

EnqueueProcessedObject(list, objectAddress, index);

}

//

// This method transfers the list of objects starting at list and ending

// at tail to the free list.

// In debug mode, it also makes sure that none of the objects that are

// being prepended to the free list are already free

//

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::TransferProcessedObjects(FreeObject \* list, FreeObject \* tail)

{

Assert(tail != nullptr);

Assert(list);

#if DBG || defined(RECYCLER\_STATS)

// make sure that object we are transfered to the free list are not freed yet

tail->SetNext(nullptr);

FreeObject \* freeObject = list;

while (freeObject != nullptr)

{

Assert(!this->IsInFreeObjectList(freeObject));

BOOL isSet = this->GetDebugFreeBitVector()->TestAndSet(GetAddressBitIndex(freeObject));

Assert(!isSet);

freeObject = freeObject->GetNext();

}

#endif

tail->SetNext(this->freeObjectList);

this->freeObjectList = list;

RECYCLER\_SLOW\_CHECK(this->CheckDebugFreeBitVector(true));

}

template <class TBlockAttributes>

uint

SmallHeapBlockT<TBlockAttributes>::GetAndClearLastFreeCount()

{

uint lastFreeCount = this->lastFreeCount;

this->lastFreeCount = 0;

return lastFreeCount;

}

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::Check(bool expectFull, bool expectPending)

{

if (this->IsFreeBitsValid())

{

CheckFreeBitVector(false);

}

else

{

CheckDebugFreeBitVector(false);

}

Assert(expectPending == HasAnyDisposeObjects());

if (this->isInAllocator || this->isClearedFromAllocator)

{

Assert(expectFull && !expectPending);

}

else

{

Assert(expectFull == (!this->HasFreeObject() && !HasAnyDisposeObjects()));

}

}

template <class TBlockAttributes>

template <typename TBlockType>

bool

SmallHeapBlockT<TBlockAttributes>::GetFreeObjectListOnAllocatorImpl(FreeObject \*\* freeObjectList)

{

// not during collection, the allocator has the current info

SmallHeapBlockAllocator<TBlockType> \* head =

&((HeapBucketT<TBlockType> \*)this->heapBucket)->allocatorHead;

SmallHeapBlockAllocator<TBlockType> \* current = head;

do

{

if (current->GetHeapBlock() == this)

{

if (current->IsFreeListAllocMode())

{

\*freeObjectList = current->freeObjectList;

return true;

}

return false;

}

current = current->GetNext();

}

while (current != head);

return false;

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::CheckDebugFreeBitVector(bool isCollecting)

{

FreeObject \* freeObject = this->freeObjectList;

if (!isCollecting)

{

this->GetFreeObjectListOnAllocator(&freeObject);

}

uint verifyFreeCount = 0;

while (freeObject != nullptr)

{

uint index = this->GetAddressIndex(freeObject);

Assert(index != SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit);

Assert(this->GetDebugFreeBitVector()->Test(GetAddressBitIndex(freeObject)));

verifyFreeCount++;

freeObject = freeObject->GetNext();

}

Assert(this->GetDebugFreeBitVector()->Count() == verifyFreeCount);

Assert(verifyFreeCount <= this->lastFreeCount);

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::CheckFreeBitVector(bool isCollecting)

{

// during collection, the heap block has the current info when we are verifying

if (!isCollecting)

{

FreeObject \* freeObjectList;

this->GetFreeObjectListOnAllocator(&freeObjectList);

if (freeObjectList != this->freeObjectList)

{

// allocator has the current info and if we have already allocated some memory,

// the free bit vector isn't really correct, so we can't verify it.

// Just verify the debug free bit vector

this->CheckDebugFreeBitVector(false);

return;

}

}

SmallHeapBlockBitVector \* free = this->GetFreeBitVector();

// Shouldn't be any invalid bits set in the free bit vector

SmallHeapBlockBitVector temp;

temp.Copy(free);

temp.And(this->GetInvalidBitVector());

Assert(temp.IsAllClear());

uint verifyFreeCount = 0;

FreeObject \* freeObject = this->freeObjectList;

while (freeObject != nullptr)

{

uint bitIndex = GetAddressBitIndex(freeObject);

Assert(IsValidBitIndex(bitIndex));

Assert(this->GetDebugFreeBitVector()->Test(bitIndex));

Assert(free->Test(bitIndex));

verifyFreeCount++;

freeObject = freeObject->GetNext();

}

Assert(this->GetDebugFreeBitVector()->Count() == verifyFreeCount);

Assert(this->freeCount == this->GetFreeBitVector()->Count());

if (this->IsAnyFinalizableBlock())

{

auto finalizableBlock = this->AsFinalizableBlock<TBlockAttributes>();

// Include pending dispose objects

finalizableBlock->ForEachPendingDisposeObject([&] (uint index) {

uint bitIndex = ((uint)index) \* this->GetObjectBitDelta();

Assert(IsValidBitIndex(bitIndex));

Assert(!this->GetDebugFreeBitVector()->Test(bitIndex));

Assert(free->Test(bitIndex));

verifyFreeCount++;

});

// Include disposed objects

verifyFreeCount += finalizableBlock->CheckDisposedObjectFreeBitVector();

}

Assert(verifyFreeCount == this->freeCount);

Assert(verifyFreeCount <= this->lastFreeCount);

Assert(this->IsFreeBitsValid());

}

#endif

template <class TBlockAttributes>

typename SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector \*

SmallHeapBlockT<TBlockAttributes>::EnsureFreeBitVector()

{

if (this->IsFreeBitsValid())

{

// the free object list hasn't change, so the free vector should be valid

RECYCLER\_SLOW\_CHECK(CheckFreeBitVector(true));

return this->GetFreeBitVector();

}

return BuildFreeBitVector();

}

template <class TBlockAttributes>

typename SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector \*

SmallHeapBlockT<TBlockAttributes>::BuildFreeBitVector()

{

SmallHeapBlockBitVector \* free = this->GetFreeBitVector();

this->freeCount = this->BuildFreeBitVector(free);

this->lastFreeObjectHead = this->freeObjectList;

return free;

}

template <class TBlockAttributes>

ushort

SmallHeapBlockT<TBlockAttributes>::BuildFreeBitVector(SmallHeapBlockBitVector \* free)

{

free->ClearAll();

ushort freeCount = 0;

FreeObject \* freeObject = this->freeObjectList;

while (freeObject != nullptr)

{

uint bitIndex = GetAddressBitIndex(freeObject);

Assert(IsValidBitIndex(bitIndex));

Assert(this->GetDebugFreeBitVector()->Test(bitIndex));

free->Set(bitIndex);

freeCount++;

freeObject = freeObject->GetNext();

}

Assert(this->GetDebugFreeBitVector()->Count() == freeCount);

if (this->IsAnyFinalizableBlock())

{

auto finalizableBlock = this->AsFinalizableBlock<TBlockAttributes>();

// Include pending dispose objects

finalizableBlock->ForEachPendingDisposeObject([&] (uint index) {

uint bitIndex = ((uint)index) \* this->GetObjectBitDelta();

Assert(IsValidBitIndex(bitIndex));

Assert(!this->GetDebugFreeBitVector()->Test(bitIndex));

free->Set(bitIndex);

freeCount++;

});

// Include disposed objects

freeCount += finalizableBlock->AddDisposedObjectFreeBitVector(free);

}

Assert(freeCount <= this->lastFreeCount);

return freeCount;

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::MarkImplicitRoots()

{

uint localObjectCount = this->GetObjectCount();

uint localObjectBitDelta = this->GetObjectBitDelta();

uint localMarkCount = 0;

SmallHeapBlockBitVector \* mark = this->GetMarkedBitVector();

#if DBG

uint localObjectSize = this->GetObjectSize();

Assert(localObjectSize <= HeapConstants::MaxMediumObjectSize);

ushort markCountPerPage[TBlockAttributes::PageCount];

for (uint i = 0; i < TBlockAttributes::PageCount; i++)

{

markCountPerPage[i] = 0;

}

#endif

for (uint i = 0; i < localObjectCount; i++)

{

// REVIEW: This may include free object. It is okay to mark them and scan them

// But kind inefficient.

if (this->ObjectInfo(i) & ImplicitRootBit)

{

#if DBG

{

int index = (i \* localObjectSize) / AutoSystemInfo::PageSize;

Assert(index < TBlockAttributes::PageCount);

markCountPerPage[index]++;

}

#endif

mark->Set(localObjectBitDelta \* i);

localMarkCount++;

}

}

Assert(mark->Count() == localMarkCount);

this->markCount = (ushort)localMarkCount;

#if DBG

HeapBlockMap& map = this->GetRecycler()->heapBlockMap;

for (uint i = 0; i < TBlockAttributes::PageCount; i++)

{

map.SetPageMarkCount(this->address + (i \* AutoSystemInfo::PageSize), markCountPerPage[i]);

}

#endif

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size))

{

ForEachAllocatedObject(infoBits, [=](uint index, void \* objectAddress)

{

CallBackFunction(objectAddress, this->objectSize);

});

}

template <class TBlockAttributes>

\_\_inline

void SmallHeapBlockT<TBlockAttributes>::FillFreeMemory(\_\_in\_bcount(size) void \* address, size\_t size)

{

#ifdef RECYCLER\_MEMORY\_VERIFY

if (this->heapBucket->heapInfo->recycler->VerifyEnabled())

{

memset(address, Recycler::VerifyMemFill, size);

return;

}

#endif

if (this->IsLeafBlock()

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_THREAD\_PAGE

|| this->IsWithBarrier()

#endif

)

{

return;

}

// REVIEW: Do DbgMemFill on debug build?

#if defined(\_M\_IX86)

uint qwordCount = size / sizeof(uint64) ;

switch (qwordCount)

{

case 2:

((uint64\*)address)[0] = 0;

((uint64\*)address)[1] = 0;

break;

case 4:

((uint64\*)address)[0] = 0;

((uint64\*)address)[1] = 0;

((uint64\*)address)[2] = 0;

((uint64\*)address)[3] = 0;

break;

case 6:

((uint64\*)address)[0] = 0;

((uint64\*)address)[1] = 0;

((uint64\*)address)[2] = 0;

((uint64\*)address)[3] = 0;

((uint64\*)address)[4] = 0;

((uint64\*)address)[5] = 0;

break;

default:

memset(address, 0, size);

}

#else

memset(address, 0, size);

#endif

}

#ifdef RECYCLER\_MEMORY\_VERIFY

template <class TBlockAttributes>

void SmallHeapBlockT<TBlockAttributes>::VerifyBumpAllocated(\_In\_ char \* bumpAllocateAddress)

{

ushort verifyFinalizeCount = 0;

Recycler \* recycler = this->heapBucket->heapInfo->recycler;

char \* memBlock = this->GetAddress();

for (uint i = 0; i < objectCount; i++)

{

if (memBlock >= bumpAllocateAddress)

{

Recycler::VerifyCheckFill(memBlock + sizeof(FreeObject), this->GetObjectSize() - sizeof(FreeObject));

}

else

{

recycler->VerifyCheckPad(memBlock, this->GetObjectSize());

if ((this->ObjectInfo(i) & FinalizeBit) != 0)

{

if (this->IsFinalizableBlock())

{

verifyFinalizeCount++;

}

else

{

Recycler::VerifyCheck(false, L"Non-Finalizable block should not have finalizable objects",

this->GetAddress(), &this->ObjectInfo(i));

}

}

}

memBlock += this->GetObjectSize();

}

}

template <class TBlockAttributes>

void SmallHeapBlockT<TBlockAttributes>::Verify(bool pendingDispose)

{

ushort verifyFinalizeCount = 0;

SmallHeapBlockBitVector tempFree;

SmallHeapBlockBitVector \*free = &tempFree;

SmallHeapBlockBitVector tempPending;

this->BuildFreeBitVector(free);

Recycler \* recycler = this->heapBucket->heapInfo->recycler;

char \* memBlock = this->GetAddress();

uint objectBitDelta = this->GetObjectBitDelta();

Recycler::VerifyCheck(!pendingDispose || this->IsAnyFinalizableBlock(),

L"Non-finalizable block shouldn't be disposing. May have corrupted block type.",

this->GetAddress(), (void \*)&this->heapBlockType);

if (HasPendingDisposeObjects())

{

Assert(pendingDispose);

// Pending object are not free yet, they don't have memory cleared.

this->AsFinalizableBlock<TBlockAttributes>()->ForEachPendingDisposeObject([&](uint index) {

uint bitIndex = ((uint)index) \* this->GetObjectBitDelta();

Assert(IsValidBitIndex(bitIndex));

Assert(!this->GetDebugFreeBitVector()->Test(bitIndex));

Assert(free->Test(bitIndex));

tempPending.Set(bitIndex);

// We are a pending dispose block, so the finalize count hasn't been update yet.

// Including the pending objects in the finalize count

verifyFinalizeCount++;

});

}

for (uint i = 0; i < objectCount; i++)

{

if (free->Test(i \* objectBitDelta))

{

if (!tempPending.Test(i \* objectBitDelta))

{

char \* nextFree = (char \*)((FreeObject \*)memBlock)->GetNext();

Recycler::VerifyCheck(nextFree == nullptr

|| (nextFree >= address && nextFree < this->GetEndAddress()

&& free->Test(GetAddressBitIndex(nextFree))),

L"SmallHeapBlock memory written to after freed", memBlock, memBlock);

Recycler::VerifyCheckFill(memBlock + sizeof(FreeObject), this->GetObjectSize() - sizeof(FreeObject));

}

}

else

{

if (explicitFreeBits.Test(i \* objectBitDelta))

{

char \* nextFree = (char \*)((FreeObject \*)memBlock)->GetNext();

HeapBlock\* nextFreeHeapBlock = this;

if (nextFree != nullptr)

{

nextFreeHeapBlock = recycler->FindHeapBlock(nextFree);

}

Recycler::VerifyCheck(nextFree == nullptr

|| (nextFree >= address && nextFree < this->GetEndAddress()

&& explicitFreeBits.Test(GetAddressBitIndex(nextFree)))

|| nextFreeHeapBlock->GetObjectSize(nextFree) == this->objectSize,

L"SmallHeapBlock memory written to after freed", memBlock, memBlock);

recycler->VerifyCheckPadExplicitFreeList(memBlock, this->GetObjectSize());

}

else

{

recycler->VerifyCheckPad(memBlock, this->GetObjectSize());

}

if ((this->ObjectInfo(i) & FinalizeBit) != 0)

{

if (this->IsAnyFinalizableBlock())

{

verifyFinalizeCount++;

}

else

{

Recycler::VerifyCheck(false, L"Non-Finalizable block should not have finalizable objects",

this->GetAddress(), &this->ObjectInfo(i));

}

}

}

memBlock += this->GetObjectSize();

}

if (this->IsAnyFinalizableBlock())

{

Recycler::VerifyCheck(this->AsFinalizableBlock<TBlockAttributes>()->finalizeCount == verifyFinalizeCount,

L"SmallHeapBlock finalize count mismatch", this->GetAddress(), &this->AsFinalizableBlock<TBlockAttributes>()->finalizeCount);

}

else

{

Assert(verifyFinalizeCount == 0);

}

}

#endif

#ifdef DUMP\_FRAGMENTATION\_STATS

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::AggregateBlockStats(HeapBucketStats& stats, bool isAllocatorBlock, FreeObject\* freeObjectList, bool isBumpAllocated)

{

stats.totalBlockCount++;

ushort blockObjectCount = this->objectCount;

BVIndex blockFreeCount = this->GetFreeBitVector()->Count();

ushort blockObjectSize = this->objectSize;

if (this->segment == nullptr)

{

stats.emptyBlockCount++;

blockObjectCount = 0;

blockFreeCount = 0;

}

int objectCount = 0;

if (isBumpAllocated)

{

objectCount = ((char\*) freeObjectList - this->address) / blockObjectSize;

}

else

{

objectCount = blockObjectCount;

// If this is an allocator block, remove the free objects on the allocator

// from this count. Otherwise, remove the free objects found in the free bit vector

if (freeObjectList)

{

Assert(isAllocatorBlock);

FreeObject\* next = freeObjectList->GetNext();

while (next != nullptr && next != freeObjectList)

{

objectCount--;

next = next->GetNext();

}

}

else

{

objectCount -= blockFreeCount;

}

}

// If we have a block that's on the allocator, it could also be on the heap block list

// In that case, we need to make sure we don't double-count this. To do that, we take out

// the block's allocatorCount/freeCount and adjust it later when we see the block

if (isAllocatorBlock)

{

objectCount -= blockObjectCount;

objectCount += blockFreeCount;

}

// Don't count empty blocks as allocable

if (this->segment != nullptr)

{

stats.totalByteCount += AutoSystemInfo::PageSize;

}

stats.objectCount += objectCount;

stats.objectByteCount += (objectCount \* blockObjectSize);

if (!isAllocatorBlock)

{

if (this->IsAnyFinalizableBlock())

{

SmallFinalizableHeapBlock\* finalizableBlock = this->AsFinalizableBlock<TBlockAttributes>();

stats.finalizeBlockCount++;

stats.finalizeCount += (finalizableBlock->GetFinalizeCount());

}

}

}

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::UpdatePerfCountersOnFree()

{

Assert(markCount == 0);

Assert(this->IsFreeBitsValid());

size\_t usedCount = (objectCount - freeCount);

size\_t usedBytes = usedCount \* objectSize;

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockLiveObject, usedCount);

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockLiveObjectSize, usedBytes);

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockFreeObjectSize, this->GetPageCount() \* AutoSystemInfo::PageSize - usedBytes);

RECYCLER\_PERF\_COUNTER\_SUB(LiveObject, usedCount);

RECYCLER\_PERF\_COUNTER\_SUB(LiveObjectSize, usedBytes);

RECYCLER\_PERF\_COUNTER\_SUB(FreeObjectSize, this->GetPageCount() \* AutoSystemInfo::PageSize - usedBytes);

}

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

template <class TBlockAttributes>

void \*

SmallHeapBlockT<TBlockAttributes>::GetTrackerData(void \* address)

{

Assert(Recycler::DoProfileAllocTracker());

ushort index = this->GetAddressIndex(address);

Assert(index != SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit);

return this->GetTrackerDataArray()[index];

}

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::SetTrackerData(void \* address, void \* data)

{

Assert(Recycler::DoProfileAllocTracker());

ushort index = this->GetAddressIndex(address);

Assert(index != SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit);

void\* existingTrackerData = this->GetTrackerDataArray()[index];

Assert((existingTrackerData == nullptr || data == nullptr) ||

(existingTrackerData == &Recycler::TrackerData::ExplicitFreeListObjectData || data == &Recycler::TrackerData::ExplicitFreeListObjectData));

this->GetTrackerDataArray()[index] = data;

}

template <class TBlockAttributes>

void \*\*

SmallHeapBlockT<TBlockAttributes>::GetTrackerDataArray()

{

// See SmallHeapBlockT<TBlockAttributes>::GetAllocPlusSize for layout description

return (void \*\*)((char \*)this - SmallHeapBlockT<TBlockAttributes>::GetAllocPlusSize(this->objectCount));

}

#endif

#ifdef RECYCLER\_WRITE\_BARRIER

template <class TBlockAttributes>

bool

SmallHeapBlockT<TBlockAttributes>::IsWithBarrier() const

{

return IsNormalWriteBarrierBlock() || IsFinalizableWriteBarrierBlock();

}

#endif

// Instantiate the template

template class SmallHeapBlockT<SmallAllocationBlockAttributes>;

template class SmallHeapBlockT<MediumAllocationBlockAttributes>;

#define TBlockTypeAttributes SmallAllocationBlockAttributes

#include "SmallBlockDeclarations.inl"

#undef TBlockTypeAttributes

#define TBlockTypeAttributes MediumAllocationBlockAttributes

#include "SmallBlockDeclarations.inl"

#undef TBlockTypeAttributes

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

class ScriptMemoryDumper;

namespace Memory

{

#ifdef RECYCLER\_PAGE\_HEAP

enum class PageHeapBlockTypeFilter;

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

#define PageHeapVerboseTrace(flags, ...) \

if (flags.Verbose && flags.Trace.IsEnabled(Js::PageHeapPhase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#define PageHeapTrace(flags, ...) \

if (flags.Trace.IsEnabled(Js::PageHeapPhase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#else

#define PageHeapVerboseTrace(...)

#define PageHeapTrace(...)

#endif

#else

#define PageHeapVerboseTrace(...)

#define PageHeapTrace(...)

#endif

class Recycler;

class HeapBucket;

template <typename TBlockType> class HeapBucketT;

class RecyclerSweep;

class MarkContext;

#ifdef DUMP\_FRAGMENTATION\_STATS

struct HeapBucketStats

{

uint totalBlockCount;

uint emptyBlockCount;

uint finalizeBlockCount;

uint objectCount;

uint finalizeCount;

uint objectByteCount;

uint totalByteCount;

};

#endif

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(RECYCLER\_MEMORY\_VERIFY) || defined(MEMSPECT\_TRACKING) || defined(RECYCLER\_PERF\_COUNTERS) || defined(ETW\_MEMORY\_TRACKING)

#define RECYCLER\_TRACK\_NATIVE\_ALLOCATED\_OBJECTS

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

#define RECYCLER\_SLOW\_CHECK(x) x

#define RECYCLER\_SLOW\_CHECK\_IF(cond, x) if (cond) { x; }

#else

#define RECYCLER\_SLOW\_CHECK(x)

#define RECYCLER\_SLOW\_CHECK\_IF(cond, x)

#endif

// ObjectInfoBits is unsigned short, but only the lower byte is stored as the object attribute

// The upper bits are used to pass other information about allocation (e.g. NoDisposeBit)

//

enum ObjectInfoBits : unsigned short

{

// Bits that are actually stored in ObjectInfo

NoBit = 0x00, // assume an allocation is not leaf unless LeafBit is specified.

FinalizeBit = 0x80, // Indicates that the object has a finalizer

PendingDisposeBit = 0x40, // Indicates that the object is pending dispose

LeafBit = 0x20, // Indicates that the object is a leaf-object (objects without this bit need to be scanned)

TrackBit = 0x10, // Indicates that the object is a TrackableObject

ImplicitRootBit = 0x08,

NewTrackBit = 0x04, // Tracked object is newly allocated and hasn't been process by concurrent GC

MemoryProfilerOldObjectBit = 0x02,

EnumClass\_1\_Bit = 0x01, // This can be extended to add more enumerable classes (if we still have bits left)

// Mask for above bits

StoredObjectInfoBitMask = 0xFF,

// Bits that implied by the block type, and thus don't need to be stored (for small blocks)

// Note, LeafBit is used in finalizable blocks, thus is not always implied by the block type

// GC-TODO: FinalizeBit doesn't need to be stored since we have separate bucket for them.

// We can move it the upper byte.

#ifdef RECYCLER\_WRITE\_BARRIER

WithBarrierBit = 0x0100,

#endif

// Mask for above bits

InternalObjectInfoBitMask = 0x01FF,

// Bits that only affect allocation behavior, not mark/sweep/etc

ClientTrackedBit = 0x0200, // This allocation is client tracked

TraceBit = 0x0400,

// Additional definitions based on above

#ifdef RECYCLER\_STATS

NewFinalizeBit = NewTrackBit, // Use to detect if the background thread has counted the finalizable object in stats

#else

NewFinalizeBit = 0x00,

#endif

#ifdef RECYCLER\_WRITE\_BARRIER

FinalizableWithBarrierBit = WithBarrierBit | FinalizeBit,

#endif

// Allocation bits

FinalizableLeafBits = NewFinalizeBit | FinalizeBit | LeafBit,

FinalizableObjectBits = NewFinalizeBit | FinalizeBit ,

#ifdef RECYCLER\_WRITE\_BARRIER

FinalizableWithBarrierObjectBits = NewFinalizeBit | FinalizableWithBarrierBit,

#endif

ClientFinalizableObjectBits = NewFinalizeBit | ClientTrackedBit | FinalizeBit,

ClientTrackableLeafBits = NewTrackBit | ClientTrackedBit | TrackBit | FinalizeBit | LeafBit,

ClientTrackableObjectBits = NewTrackBit | ClientTrackedBit | TrackBit | FinalizeBit,

WeakReferenceEntryBits = LeafBit,

ImplicitRootLeafBits = LeafBit | ImplicitRootBit,

// Pending dispose objects should have LeafBit set and no others

PendingDisposeObjectBits = PendingDisposeBit | LeafBit,

#ifdef RECYCLER\_WRITE\_BARRIER

GetBlockTypeBitMask = FinalizeBit | LeafBit | WithBarrierBit,

#else

GetBlockTypeBitMask = FinalizeBit | LeafBit,

#endif

CollectionBitMask = LeafBit | FinalizeBit | TrackBit | NewTrackBit, // Bits relevant to collection

EnumClassMask = EnumClass\_1\_Bit,

};

enum ResetMarkFlags

{

ResetMarkFlags\_None = 0x0,

ResetMarkFlags\_Background = 0x1,

ResetMarkFlags\_ScanImplicitRoot = 0x2,

// For in thread GC

ResetMarkFlags\_InThread = ResetMarkFlags\_None,

ResetMarkFlags\_InThreadImplicitRoots = ResetMarkFlags\_None | ResetMarkFlags\_ScanImplicitRoot,

// For background GC

ResetMarkFlags\_InBackgroundThread = ResetMarkFlags\_Background,

ResetMarkFlags\_InBackgroundThreadImplicitRoots = ResetMarkFlags\_Background | ResetMarkFlags\_ScanImplicitRoot,

// For blocking synchronized GC

ResetMarkFlags\_Synchronized = ResetMarkFlags\_None,

ResetMarkFlags\_SynchronizedImplicitRoots = ResetMarkFlags\_None | ResetMarkFlags\_ScanImplicitRoot,

// For heap enumeration

ResetMarkFlags\_HeapEnumeration = ResetMarkFlags\_None,

};

enum RescanFlags

{

RescanFlags\_None = 0x0,

RescanFlags\_ResetWriteWatch = 0x1

};

enum FindHeapObjectFlags

{

FindHeapObjectFlags\_NoFlags = 0x0,

FindHeapObjectFlags\_ClearedAllocators = 0x1, // Assumes that the allocator is already cleared

FindHeapObjectFlags\_VerifyFreeBitForAttribute = 0x2, // Don't recompute the free bit vector if there is no pending objects, the attributes will always be correct

FindHeapObjectFlags\_NoFreeBitVerify = 0x4, // No checking whether the address is free or not.

FindHeapObjectFlags\_AllowInterior = 0x8, // Allow finding heap objects for interior pointers.

};

template <class TBlockAttributes> class SmallNormalHeapBlockT;

template <class TBlockAttributes> class SmallLeafHeapBlockT;

template <class TBlockAttributes> class SmallFinalizableHeapBlockT;

#ifdef RECYCLER\_WRITE\_BARRIER

template <class TBlockAttributes> class SmallNormalWithBarrierHeapBlockT;

template <class TBlockAttributes> class SmallFinalizableWithBarrierHeapBlockT;

#define EXPLICIT\_INSTANTIATE\_WITH\_SMALL\_HEAP\_BLOCK\_TYPE(TemplateType) \

template class TemplateType<SmallNormalHeapBlock>; \

template class TemplateType<SmallLeafHeapBlock>; \

template class TemplateType<SmallFinalizableHeapBlock>; \

template class TemplateType<SmallNormalWithBarrierHeapBlock>; \

template class TemplateType<SmallFinalizableWithBarrierHeapBlock>; \

template class TemplateType<MediumNormalHeapBlock>; \

template class TemplateType<MediumLeafHeapBlock>; \

template class TemplateType<MediumFinalizableHeapBlock>; \

template class TemplateType<MediumNormalWithBarrierHeapBlock>; \

template class TemplateType<MediumFinalizableWithBarrierHeapBlock>; \

#else

#define EXPLICIT\_INSTANTIATE\_WITH\_SMALL\_HEAP\_BLOCK\_TYPE(TemplateType) \

template class TemplateType<SmallNormalHeapBlock>; \

template class TemplateType<SmallLeafHeapBlock>; \

template class TemplateType<SmallFinalizableHeapBlock>; \

template class TemplateType<MediumNormalHeapBlock>; \

template class TemplateType<MediumLeafHeapBlock>; \

template class TemplateType<MediumFinalizableHeapBlock>; \

#endif

class RecyclerHeapObjectInfo;

class HeapBlock

{

public:

enum HeapBlockType : byte

{

FreeBlockType = 0, // Only used in HeapBlockMap. Actual HeapBlock structures should never have this.

SmallNormalBlockType = 1,

SmallLeafBlockType = 2,

SmallFinalizableBlockType = 3,

#ifdef RECYCLER\_WRITE\_BARRIER

SmallNormalBlockWithBarrierType = 4,

SmallFinalizableBlockWithBarrierType = 5,

#endif

MediumNormalBlockType = 6,

MediumLeafBlockType = 7,

MediumFinalizableBlockType = 8,

#ifdef RECYCLER\_WRITE\_BARRIER

MediumNormalBlockWithBarrierType = 9,

MediumFinalizableBlockWithBarrierType = 10,

#endif

LargeBlockType = 11,

SmallAllocBlockTypeCount = 6, // Actual number of types for blocks containing small allocations

MediumAllocBlockTypeCount = 5, // Actual number of types for blocks containing medium allocations

SmallBlockTypeCount = 11, // Distinct block types independent of allocation size using SmallHeapBlockT

BlockTypeCount = 12,

};

bool IsNormalBlock() const { return this->GetHeapBlockType() == SmallNormalBlockType || this->GetHeapBlockType() == MediumNormalBlockType; }

bool IsLeafBlock() const { return this->GetHeapBlockType() == SmallLeafBlockType || this->GetHeapBlockType() == MediumLeafBlockType; }

bool IsFinalizableBlock() const { return this->GetHeapBlockType() == SmallFinalizableBlockType || this->GetHeapBlockType() == MediumFinalizableBlockType; }

#ifdef RECYCLER\_WRITE\_BARRIER

bool IsAnyNormalBlock() const { return IsNormalBlock() || IsNormalWriteBarrierBlock(); }

bool IsAnyFinalizableBlock() const { return IsFinalizableBlock() || IsFinalizableWriteBarrierBlock(); }

bool IsNormalWriteBarrierBlock() const { return this->GetHeapBlockType() == SmallNormalBlockWithBarrierType || this->GetHeapBlockType() == MediumNormalBlockWithBarrierType; }

bool IsFinalizableWriteBarrierBlock() const { return this->GetHeapBlockType() == SmallFinalizableBlockWithBarrierType || this->GetHeapBlockType() == MediumFinalizableBlockWithBarrierType; }

#else

bool IsAnyFinalizableBlock() const { return IsFinalizableBlock(); }

bool IsAnyNormalBlock() const { return IsNormalBlock(); }

#endif

bool IsLargeHeapBlock() const { return this->GetHeapBlockType() == LargeBlockType; }

char \* GetAddress() const { return address; }

Segment \* GetSegment() const { return segment; }

template <typename TBlockAttributes>

SmallNormalHeapBlockT<TBlockAttributes> \* AsNormalBlock();

template <typename TBlockAttributes>

SmallLeafHeapBlockT<TBlockAttributes> \* AsLeafBlock();

template <typename TBlockAttributes>

SmallFinalizableHeapBlockT<TBlockAttributes> \* AsFinalizableBlock();

#ifdef RECYCLER\_WRITE\_BARRIER

template <typename TBlockAttributes>

SmallNormalWithBarrierHeapBlockT<TBlockAttributes> \* AsNormalWriteBarrierBlock();

template <typename TBlockAttributes>

SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes> \* AsFinalizableWriteBarrierBlock();

#endif

protected:

char \* address;

Segment \* segment;

HeapBlockType const heapBlockType;

bool needOOMRescan; // Set if we OOMed while marking a particular object

#ifdef CONCURRENT\_GC\_ENABLED

bool isPendingConcurrentSweep;

#endif

#ifdef RECYCLER\_PAGE\_HEAP

PageHeapMode pageHeapMode;

DWORD guardPageOldProtectFlags;

char\* guardPageAddress;

StackBackTrace\* pageHeapAllocStack;

StackBackTrace\* pageHeapFreeStack;

public:

\_\_inline bool InPageHeapMode() const { return pageHeapMode != PageHeapMode::PageHeapModeOff; }

void CapturePageHeapAllocStack();

void CapturePageHeapFreeStack();

#endif

public:

template <typename Fn>

bool UpdateAttributesOfMarkedObjects(MarkContext \* markContext, void \* objectAddress, size\_t objectSize, unsigned char attributes, Fn fn);

void SetNeedOOMRescan(Recycler \* recycler);

public:

HeapBlock(HeapBlockType heapBlockType) :

heapBlockType(heapBlockType),

needOOMRescan(false)

#ifdef RECYCLER\_PAGE\_HEAP

, pageHeapAllocStack(nullptr), pageHeapFreeStack(nullptr)

#endif

{

Assert(GetHeapBlockType() <= HeapBlock::HeapBlockType::BlockTypeCount);

}

HeapBlockType const GetHeapBlockType() const

{

return (heapBlockType);

}

IdleDecommitPageAllocator\* GetPageAllocator(Recycler\* recycler);

bool GetAndClearNeedOOMRescan()

{

if (this->needOOMRescan)

{

this->needOOMRescan = false;

return true;

}

return false;

}

#if DBG

virtual BOOL IsFreeObject(void\* objectAddress) = 0;

#endif

virtual BOOL IsValidObject(void\* objectAddress) = 0;

virtual byte\* GetRealAddressFromInterior(void\* interiorAddress) = 0;

virtual size\_t GetObjectSize(void\* object) = 0;

virtual bool FindHeapObject(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject) = 0;

virtual bool TestObjectMarkedBit(void\* objectAddress) = 0;

virtual void SetObjectMarkedBit(void\* objectAddress) = 0;

#ifdef RECYCLER\_VERIFY\_MARK

virtual void VerifyMark(void \* objectAddress) = 0;

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

virtual void \* GetTrackerData(void \* address) = 0;

virtual void SetTrackerData(void \* address, void \* data) = 0;

#endif

#if DBG || defined(RECYCLER\_STATS) || defined(RECYCLER\_PAGE\_HEAP)

bool isForceSweeping;

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

virtual void UpdatePerfCountersOnFree() = 0;

#endif

};

enum SweepMode

{

SweepMode\_InThread,

#ifdef CONCURRENT\_GC\_ENABLED

SweepMode\_Concurrent,

#ifdef PARTIAL\_GC\_ENABLED

SweepMode\_ConcurrentPartial

#endif

#endif

};

// enum indicating the result of a sweep

enum SweepState

{

SweepStateEmpty, // the block is completely empty and can be released

SweepStateSwept, // the block is partially allocated, no object needs to be swept or finalized

SweepStateFull, // the block is full, no object needs to be swept or finalized

SweepStatePendingDispose, // the block has object that needs to be finalized

#ifdef CONCURRENT\_GC\_ENABLED

SweepStatePendingSweep, // the block has object that needs to be swept

#endif

};

template <class TBlockAttributes>

class ValidPointers

{

public:

ValidPointers(ushort const \* validPointers);

ushort GetInteriorAddressIndex(uint index) const;

ushort GetAddressIndex(uint index) const;

private:

ushort const \* validPointers;

};

template <class TBlockAttributes>

class SmallHeapBlockT : public HeapBlock

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

friend class ScriptMemoryDumper;

#endif

template <typename TBlockType>

friend class SmallHeapBlockAllocator;

friend class HeapInfo;

friend class RecyclerSweep;

template <typename TBlockType>

friend class SmallNormalHeapBucketBase;

#ifdef JD\_PRIVATE

friend class HeapBlockHelper;

friend class EXT\_CLASS;

#endif

public:

static const ushort InvalidAddressBit = 0xFFFF;

typedef BVStatic<TBlockAttributes::BitVectorCount> SmallHeapBlockBitVector;

struct BlockInfo

{

ushort lastObjectIndexOnPage;

ushort pageObjectCount;

};

bool FindImplicitRootObject(void\* candidate, Recycler\* recycler, RecyclerHeapObjectInfo& heapObject);

SmallHeapBlockT\* next;

FreeObject\* freeObjectList;

FreeObject\* lastFreeObjectHead;

ValidPointers<TBlockAttributes> validPointers;

HeapBucket \* heapBucket;

// Review: Should GetBucketIndex return a short instead of an int?

const uint bucketIndex;

const ushort objectSize; // size in bytes

const ushort objectCount;

ushort freeCount;

ushort lastFreeCount;

ushort markCount;

#ifdef PARTIAL\_GC\_ENABLED

ushort oldFreeCount;

#endif

bool isInAllocator;

#if DBG

bool isClearedFromAllocator;

bool isIntegratedBlock;

uint lastUncollectedAllocBytes;

#endif

SmallHeapBlockBitVector\* markBits;

SmallHeapBlockBitVector freeBits;

#if DBG || defined(RECYCLER\_STATS)

SmallHeapBlockBitVector debugFreeBits;

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

SmallHeapBlockBitVector explicitFreeBits;

#endif

bool IsFreeBitsValid() const

{

return this->freeObjectList == this->lastFreeObjectHead;

}

PageSegment \* GetPageSegment() const { return (PageSegment \*)GetSegment(); }

public:

~SmallHeapBlockT();

#ifdef RECYCLER\_WRITE\_BARRIER

bool IsWithBarrier() const;

#endif

void RemoveFromHeapBlockMap(Recycler\* recycler);

char\* GetAddress() const { return address; }

char \* GetEndAddress() const { return address + (this->GetPageCount() \* AutoSystemInfo::PageSize); }

uint GetObjectWordCount() const { return this->objectSize / sizeof(void \*); }

uint GetPageCount() const;

template<bool checkPageHeap=true>

bool HasFreeObject() const

{

#ifdef RECYCLER\_PAGE\_HEAP

// in pageheap, we point freeObjectList to end of the allocable block to cheat the system.

// but sometimes we need to know if it's really no free block or not.

if (checkPageHeap)

{

if (this->pageHeapMode != PageHeapMode::PageHeapModeOff)

{

return false;

}

}

#endif

return freeObjectList != nullptr;

}

bool IsInAllocator() const;

bool HasPendingDisposeObjects();

bool HasAnyDisposeObjects();

#if DBG

void VerifyMarkBitVector();

bool IsClearedFromAllocator() const;

void SetIsClearedFromAllocator(bool value);

void SetIsIntegratedBlock() { this->isIntegratedBlock = true; }

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void SetExplicitFreeBitForObject(void\* object);

void ClearExplicitFreeBitForObject(void\* object);

#endif

#ifdef RECYCLER\_STRESS

void InduceFalsePositive(Recycler \* recycler);

#endif

#ifdef DUMP\_FRAGMENTATION\_STATS

void AggregateBlockStats(HeapBucketStats& stats, bool isAllocatorBlock = false, FreeObject\* freeObjectList = nullptr, bool isBumpAllocated = false);

#endif

/\*

\* Quick description of the bit vectors

\*

\* The free bit vector is created by EnsureFreeBitVector. It's created by walking through the free list

\* for the heap block and setting the corresponding bit indices in the bit vector.

\*

\* The mark bit vector is more complicated. In most cases, it represents the objects that are alive (marked)

\* + the objects the are free (i.e in the free list). This is so that when we sweep, we don't bother sweeping over objects

\* that are already in the free list, we sweep over objects that were allocated and no longer alive since the last GC.

\* However, during rescan, the mark bit vector represents the objects that are actually alive. We set the marked bit

\* vector to this state before calling RescanObjects, so that we scan through only the objects that are actually alive.

\* This means that we don't rescan newly allocated objects during rescan, because rescan doesn't change add new mark bits.

\* Instead, these objects are marked after rescan during in-thread mark if they're actually alive.

\*/

SmallHeapBlockBitVector \* GetMarkedBitVector() { return markBits; }

SmallHeapBlockBitVector \* GetFreeBitVector() { return &freeBits; }

SmallHeapBlockBitVector const \* GetInvalidBitVector();

BlockInfo const \* GetBlockInfo();

ushort GetObjectBitDelta();

static uint GetObjectBitDeltaForBucketIndex(uint bucketIndex);

static char\* GetBlockStartAddress(char\* address)

{

uintptr\_t mask = ~((TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) - 1);

return (char\*)((uintptr\_t)address & mask);

}

bool IsValidBitIndex(uint bitIndex)

{

Assert(bitIndex < TBlockAttributes::BitVectorCount);

return bitIndex % GetObjectBitDelta() == 0;

}

void MarkImplicitRoots();

void SetNextBlock(SmallHeapBlockT \* next) { this->next=next; }

SmallHeapBlockT \* GetNextBlock() const { return next; }

uint GetObjectSize() const { return objectSize; }

uint GetObjectCount() const { return objectCount; }

uint GetMarkedCount() const { return markCount; }

// Valid during sweep time

ushort GetExpectedFreeObjectCount() const;

uint GetExpectedFreeBytes() const;

ushort GetExpectedSweepObjectCount() const;

#if DBG || defined(RECYCLER\_STATS)

SmallHeapBlockBitVector \* GetDebugFreeBitVector() { return &debugFreeBits; }

#endif

#if DBG

virtual BOOL IsFreeObject(void\* objectAddress) override;

#endif

virtual BOOL IsValidObject(void\* objectAddress) override;

byte\* GetRealAddressFromInterior(void\* interiorAddress) override sealed;

bool TestObjectMarkedBit(void\* objectAddress) override sealed;

void SetObjectMarkedBit(void\* objectAddress) override;

virtual size\_t GetObjectSize(void\* object) override { return objectSize; }

#ifdef RECYCLER\_PAGE\_HEAP

char \* GetPageHeapObjectAddress();

#endif

template <bool pageheap>

uint GetMarkCountForSweep();

template <bool pageheap>

SweepState Sweep(RecyclerSweep& recyclerSweep, bool queuePendingSweep, bool allocable, ushort finalizeCount = 0, bool hasPendingDispose = false);

template <bool pageheap, SweepMode mode>

void SweepObjects(Recycler \* recycler);

uint GetAndClearLastFreeCount();

#ifdef PARTIAL\_GC\_ENABLED

void ClearAllAllocBytes(); // Reset all unaccounted alloc bytes and the new alloc count

uint GetAndClearUnaccountedAllocBytes();

void AdjustPartialUncollectedAllocBytes(RecyclerSweep& recyclerSweep, uint const expectSweepCount);

bool DoPartialReusePage(RecyclerSweep const& recyclerSweep, uint& expectFreeByteCount);

#if DBG || defined(RECYCLER\_STATS)

void SweepVerifyPartialBlock(Recycler \* recycler);

#endif

#endif

void TransferProcessedObjects(FreeObject \* list, FreeObject \* tail);

template<bool pageheap>

BOOL ReassignPages(Recycler \* recycler);

template<bool pageheap>

\_\_inline const uint GetPageHeapModePageCount() const;

#ifdef RECYCLER\_PAGE\_HEAP

void ClearPageHeapState();

#endif

template<bool pageheap>

BOOL SetPage(\_\_in\_ecount\_pagesize char \* baseAddress, PageSegment \* pageSegment, Recycler \* recycler);

template<bool pageheap>

void ReleasePages(Recycler \* recycler);

template<bool pageheap>

void ReleasePagesSweep(Recycler \* recycler);

void ReleasePagesShutdown(Recycler \* recycler);

template<bool pageheap>

void BackgroundReleasePagesSweep(Recycler\* recycler);

void Reset();

void EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size));

bool IsImplicitRoot(uint objectIndex)

{

return (this->ObjectInfo(objectIndex) & ImplicitRootBit) != 0;

}

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

void Check(bool expectFull, bool expectPending);

#endif

#ifdef RECYCLER\_PAGE\_HEAP

void VerifyPageHeapAllocation(\_In\_ char\* allocation, PageHeapMode mode);

void EnablePageHeap();

void ClearPageHeap();

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void Verify(bool pendingDispose = false);

void VerifyBumpAllocated(\_In\_ char \* bumpAllocatedAddres);

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void VerifyMark();

virtual void VerifyMark(void \* objectAddress) override;

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

virtual void UpdatePerfCountersOnFree() override sealed;

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

virtual void \* GetTrackerData(void \* address) override;

virtual void SetTrackerData(void \* address, void \* data) override;

#endif

static ushort GetAddressBitIndex(void \* objectAddress);

static void \* GetRealAddressFromInterior(void \* objectAddress, uint objectSize, byte bucketIndex);

protected:

static size\_t GetAllocPlusSize(uint objectCount);

\_\_inline void SetAttributes(void \* address, unsigned char attributes);

SmallHeapBlockT(HeapBucket \* bucket, ushort objectSize, ushort objectCount, HeapBlockType heapBlockType);

ushort GetAddressIndex(void \* objectAddress);

ushort GetInteriorAddressIndex(void \* interorAddress);

ushort GetObjectIndexFromBitIndex(ushort bitIndex);

template <SweepMode mode>

void SweepObject(Recycler \* recycler, uint index, void \* addr);

void EnqueueProcessedObject(FreeObject \*\*list, void \* objectAddress, uint index);

void EnqueueProcessedObject(FreeObject \*\*list, FreeObject \*\* tail, void \* objectAddress, uint index);

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <typename TBlockType>

bool GetFreeObjectListOnAllocatorImpl(FreeObject \*\* freeObjectList);

virtual bool GetFreeObjectListOnAllocator(FreeObject \*\* freeObjectList) = 0;

void CheckDebugFreeBitVector(bool isCollecting);

void CheckFreeBitVector(bool isCollecting);

#endif

SmallHeapBlockBitVector \* EnsureFreeBitVector();

SmallHeapBlockBitVector \* BuildFreeBitVector();

ushort BuildFreeBitVector(SmallHeapBlockBitVector \* bv);

BOOL IsInFreeObjectList(void \* objectAddress);

void ClearObjectInfoList();

byte& ObjectInfo(uint index);

\_\_inline void FillFreeMemory(\_\_in\_bcount(size) void \* address, size\_t size);

template <typename TBlockType>

bool FindHeapObjectImpl(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject);

protected:

void Init(ushort objectSize, ushort objectCount);

void ConstructorCommon(HeapBucket \* bucket, ushort objectSize, ushort objectCount, HeapBlockType heapBlockType);

template <typename Fn>

void ForEachAllocatedObject(Fn fn);

template <typename Fn>

void ForEachAllocatedObject(ObjectInfoBits attributes, Fn fn);

template <typename Fn>

void ScanNewImplicitRootsBase(Fn fn);

// This is public for code readability but this

// returns a value only on debug builds. On retail builds

// this returns null

Recycler \* GetRecycler() const;

#if DBG

uint GetMarkCountOnHeapBlockMap() const;

#endif

private:

#ifdef PROFILE\_RECYCLER\_ALLOC

void \*\* GetTrackerDataArray();

#endif

};

// Declare the class templates

typedef SmallHeapBlockT<SmallAllocationBlockAttributes> SmallHeapBlock;

typedef SmallHeapBlockT<MediumAllocationBlockAttributes> MediumHeapBlock;

extern template class SmallHeapBlockT<SmallAllocationBlockAttributes>;

extern template class SmallHeapBlockT<MediumAllocationBlockAttributes>;

extern template class ValidPointers<SmallAllocationBlockAttributes>;

extern template class ValidPointers<MediumAllocationBlockAttributes>;

class HeapBlockList

{

public:

template <typename TBlockType, typename Fn>

static void ForEach(TBlockType \* list, Fn fn)

{

ForEach<TBlockType, Fn>(list, nullptr, fn);

}

template <typename TBlockType, typename Fn>

static void ForEach(TBlockType \* list, TBlockType \* tail, Fn fn)

{

TBlockType \* heapBlock = list;

while (heapBlock != tail)

{

fn(heapBlock);

heapBlock = heapBlock->GetNextBlock();

}

}

template <typename TBlockType, typename Fn>

static void ForEachEditing(TBlockType \* list, Fn fn)

{

ForEachEditing<TBlockType, Fn>(list, nullptr, fn);

};

template <typename TBlockType, typename Fn>

static void ForEachEditing(TBlockType \* list, TBlockType \* tail, Fn fn)

{

TBlockType \* heapBlock = list;

while (heapBlock != tail)

{

TBlockType \* nextBlock = heapBlock->GetNextBlock();

fn(heapBlock);

heapBlock = nextBlock;

}

};

template <typename TBlockType>

static size\_t Count(TBlockType \* list)

{

size\_t currentHeapBlockCount = 0;

HeapBlockList::ForEach(list, [&currentHeapBlockCount](TBlockType \* heapBlock)

{

currentHeapBlockCount++;

});

return currentHeapBlockCount;

};

template <typename TBlockType>

static TBlockType \* Tail(TBlockType \* list)

{

TBlockType \* tail = nullptr;

HeapBlockList::ForEach(list, [&tail](TBlockType \* heapBlock)

{

tail = heapBlock;

});

return tail;

}

#if DBG

template <typename TBlockType>

static bool Contains(TBlockType \* block, TBlockType \* list, TBlockType \* tail = nullptr)

{

TBlockType \* heapBlock = list;

while (heapBlock != tail)

{

if (heapBlock == block)

{

return true;

}

heapBlock = heapBlock->GetNextBlock();

}

return false;

}

#endif

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template <class TBlockAttributes>

void

SmallHeapBlockT<TBlockAttributes>::SetAttributes(void \* address, unsigned char attributes)

{

Assert(this->address != nullptr);

Assert(this->segment != nullptr);

Assert(this->ObjectInfo(GetAddressIndex(address)) == 0);

ushort index = GetAddressIndex(address);

Assert(index != SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit);

ObjectInfo(index) = attributes;

}

\_\_inline

IdleDecommitPageAllocator\*

HeapBlock::GetPageAllocator(Recycler\* recycler)

{

switch (this->GetHeapBlockType())

{

case SmallLeafBlockType:

case MediumLeafBlockType:

return recycler->GetRecyclerLeafPageAllocator();

case LargeBlockType:

return recycler->GetRecyclerLargeBlockPageAllocator();

#ifdef RECYCLER\_WRITE\_BARRIER

case SmallNormalBlockWithBarrierType:

case SmallFinalizableBlockWithBarrierType:

case MediumNormalBlockWithBarrierType:

case MediumFinalizableBlockWithBarrierType:

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_THREAD\_PAGE

return recycler->GetRecyclerLeafPageAllocator();

#elif defined(RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE)

return recycler->GetRecyclerWithBarrierPageAllocator();

#endif

#endif

default:

return recycler->GetRecyclerPageAllocator();

};

}

template <class TBlockAttributes>

template <class Fn>

void SmallHeapBlockT<TBlockAttributes>::ForEachAllocatedObject(Fn fn)

{

uint const objectBitDelta = this->GetObjectBitDelta();

SmallHeapBlockBitVector \* free = this->EnsureFreeBitVector();

char \* address = this->GetAddress();

uint objectSize = this->GetObjectSize();

for (uint i = 0; i < objectCount; i++)

{

if (!free->Test(i \* objectBitDelta))

{

fn(i, address + i \* objectSize);

}

}

}

template <class TBlockAttributes>

template <typename Fn>

void SmallHeapBlockT<TBlockAttributes>::ForEachAllocatedObject(ObjectInfoBits attributes, Fn fn)

{

ForEachAllocatedObject([=](uint index, void \* objectAddress)

{

if ((ObjectInfo(index) & attributes) != 0)

{

fn(index, objectAddress);

}

});

};

template <class TBlockAttributes>

template <typename Fn>

void SmallHeapBlockT<TBlockAttributes>::ScanNewImplicitRootsBase(Fn fn)

{

uint const localObjectCount = this->objectCount;

// NOTE: we no longer track the mark count as we mark. So this value

// is basically the mark count we set during the initial implicit root scan

// plus any subsequent new implicit root scan.

uint localMarkCount = this->markCount;

if (localMarkCount == localObjectCount)

{

// The block is full when we first do the initial implicit root scan

// So there can't be any new implicit roots

return;

}

#if DBG

HeapBlockMap& map = this->GetRecycler()->heapBlockMap;

ushort newlyMarkedCountForPage[TBlockAttributes::PageCount];

for (uint i = 0; i < TBlockAttributes::PageCount; i++)

{

newlyMarkedCountForPage[i] = 0;

}

#endif

uint const localObjectBitDelta = this->GetObjectBitDelta();

uint const localObjectSize = this->GetObjectSize();

Assert(localObjectSize <= HeapConstants::MaxMediumObjectSize);

SmallHeapBlockBitVector \* mark = this->GetMarkedBitVector();

char \* address = this->GetAddress();

for (uint i = 0; i < localObjectCount; i++)

{

if ((this->ObjectInfo(i) & ImplicitRootBit) != 0

&& !mark->TestAndSet(i \* localObjectBitDelta))

{

uint objectOffset = i \* localObjectSize;

localMarkCount++;

#if DBG

uint pageNumber = objectOffset / AutoSystemInfo::PageSize;

Assert(pageNumber < TBlockAttributes::PageCount);

newlyMarkedCountForPage[pageNumber]++;

#endif

fn(address + objectOffset, localObjectSize);

}

}

Assert(localMarkCount <= USHRT\_MAX);

#if DBG

// Add newly marked count

for (uint i = 0; i < TBlockAttributes::PageCount; i++)

{

char\* pageAddress = address + (AutoSystemInfo::PageSize \* i);

ushort oldPageMarkCount = map.GetPageMarkCount(pageAddress);

map.SetPageMarkCount(pageAddress, oldPageMarkCount + newlyMarkedCountForPage[i]);

}

#endif

this->markCount = (ushort)localMarkCount;

}

template <class TBlockAttributes>

bool

SmallHeapBlockT<TBlockAttributes>::FindImplicitRootObject(void\* candidate, Recycler\* recycler, RecyclerHeapObjectInfo& heapObject)

{

ushort index = GetAddressIndex(candidate);

if (index == InvalidAddressBit)

{

return false;

}

byte& attributes = ObjectInfo(index);

heapObject = RecyclerHeapObjectInfo(candidate, recycler, this, &attributes);

return true;

}

template <typename Fn>

bool

HeapBlock::UpdateAttributesOfMarkedObjects(MarkContext \* markContext, void \* objectAddress, size\_t objectSize, unsigned char attributes, Fn fn)

{

bool noOOMDuringMark = true;

if (attributes & TrackBit)

{

FinalizableObject \* trackedObject = (FinalizableObject \*)objectAddress;

if (!markContext->GetRecycler()->inPartialCollectMode)

{

if (markContext->GetRecycler()->DoQueueTrackedObject())

{

if (!markContext->AddTrackedObject(trackedObject))

{

noOOMDuringMark = false;

}

}

else

{

// Process the tracked object right now

markContext->MarkTrackedObject(trackedObject);

}

}

if (noOOMDuringMark)

{

// Object has been successfully processed, so clear NewTrackBit

attributes &= ~NewTrackBit;

}

else

{

// Set the NewTrackBit, so that the main thread will redo tracking

attributes |= NewTrackBit;

noOOMDuringMark = false;

}

fn(attributes);

}

// only need to scan non-leaf objects

if ((attributes & LeafBit) == 0)

{

if (!markContext->AddMarkedObject(objectAddress, objectSize))

{

noOOMDuringMark = false;

}

}

#ifdef RECYCLER\_STATS

RECYCLER\_STATS\_INTERLOCKED\_INC(markContext->GetRecycler(), markData.markCount);

RECYCLER\_STATS\_INTERLOCKED\_ADD(markContext->GetRecycler(), markData.markBytes, objectSize);

// Don't count track or finalize it if we still have to process it in thread because of OOM

if ((attributes & (TrackBit | NewTrackBit)) != (TrackBit | NewTrackBit))

{

// Only count those we have queued, so we don't double count

if (attributes & TrackBit)

{

RECYCLER\_STATS\_INTERLOCKED\_INC(markContext->GetRecycler(), trackCount);

}

if (attributes & FinalizeBit)

{

// we counted the finalizable object here,

// turn off the new bit so we don't count it again

// on Rescan

attributes &= ~NewFinalizeBit;

fn(attributes);

RECYCLER\_STATS\_INTERLOCKED\_INC(markContext->GetRecycler(), finalizeCount);

}

}

#endif

return noOOMDuringMark;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#if defined(\_M\_X64\_OR\_ARM64)

HeapBlockMap32::HeapBlockMap32(\_\_in char \* startAddress) :

startAddress(startAddress),

#else

HeapBlockMap32::HeapBlockMap32() :

#endif

count(0)

{

memset(map, 0, sizeof(map));

#if defined(\_M\_X64\_OR\_ARM64)

Assert(((size\_t)startAddress) % TotalSize == 0);

#endif

}

HeapBlockMap32::~HeapBlockMap32()

{

for (uint i = 0; i < \_countof(map); i++)

{

L2MapChunk \* chunk = map[i];

if (chunk)

{

NoMemProtectHeapDelete(chunk);

}

}

}

HeapBlock \*

HeapBlockMap32::GetHeapBlock(void \* address)

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* l2map = map[id1];

if (l2map == nullptr)

{

return nullptr;

}

return l2map->Get(address);

}

bool

HeapBlockMap32::EnsureHeapBlock(void \* address, uint pageCount)

{

uint id1 = GetLevel1Id(address);

uint id2 = GetLevel2Id(address);

uint currentPageCount = min(pageCount, L2Count - id2);

while (true)

{

if (map[id1] == nullptr)

{

L2MapChunk \* newChunk = NoMemProtectHeapNewNoThrowZ(L2MapChunk);

if (newChunk == nullptr)

{

// Leave any previously allocated L2MapChunks in place --

// the concurrent thread may have already accessed them.

// These will be cleaned up in the Cleanup method, after Sweep is complete.

return false;

}

map[id1] = newChunk;

count++;

}

pageCount -= currentPageCount;

if (pageCount == 0)

{

break;

}

id2 = 0;

id1++;

currentPageCount = min(pageCount, L2Count);

}

return true;

}

void

HeapBlockMap32::SetHeapBlockNoCheck(void \* address, uint pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex)

{

uint id1 = GetLevel1Id(address);

uint id2 = GetLevel2Id(address);

uint currentPageCount = min(pageCount, L2Count - id2);

while (true)

{

Assert(map[id1] != nullptr);

map[id1]->Set(id2, currentPageCount, heapBlock, blockType, bucketIndex);

pageCount -= currentPageCount;

if (pageCount == 0)

{

return;

}

id2 = 0;

id1++;

currentPageCount = min(pageCount, L2Count);

}

}

bool

HeapBlockMap32::SetHeapBlock(void \* address, uint pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex)

{

// First, make sure we have all the necessary L2MapChunks we'll need.

// This ensures that in case of failure, the concurrent thread won't see an inconsistent state.

if (!EnsureHeapBlock(address, pageCount))

{

return false;

}

// Now, do the actual set, which cannot fail.

SetHeapBlockNoCheck(address, pageCount, heapBlock, blockType, bucketIndex);

return true;

}

void

HeapBlockMap32::ClearHeapBlock(void \* address, uint pageCount)

{

uint id1 = GetLevel1Id(address);

uint id2 = GetLevel2Id(address);

uint currentPageCount = min(pageCount, L2Count - id2);

while (true)

{

Assert(map[id1] != nullptr);

map[id1]->Clear(id2, currentPageCount);

pageCount -= currentPageCount;

if (pageCount == 0)

{

return;

}

id2 = 0;

id1++;

currentPageCount = min(pageCount, L2Count);

}

}

HeapBlockMap32::PageMarkBitVector \*

HeapBlockMap32::GetPageMarkBitVector(void \* address)

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* l2map = map[id1];

if (l2map == nullptr)

{

return nullptr;

}

return l2map->GetPageMarkBitVector(address);

}

template <size\_t BitCount>

BVStatic<BitCount>\*

HeapBlockMap32::GetMarkBitVectorForPages(void \* address)

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* l2map = map[id1];

if (l2map == nullptr)

{

return nullptr;

}

return l2map->GetMarkBitVectorForPages<BitCount>(address);

}

template BVStatic<SmallAllocationBlockAttributes::BitVectorCount>\* HeapBlockMap32::GetMarkBitVectorForPages<SmallAllocationBlockAttributes::BitVectorCount>(void \* address);

template BVStatic<MediumAllocationBlockAttributes::BitVectorCount>\* HeapBlockMap32::GetMarkBitVectorForPages<MediumAllocationBlockAttributes::BitVectorCount>(void \* address);

uint

HeapBlockMap32::GetMarkCount(void \* address, uint pageCount)

{

uint markCount = 0;

ForEachChunkInAddressRange(address, pageCount, [&](L2MapChunk\* l2Map, uint chunkId)

{

markCount += l2Map->GetPageMarkBitVector(chunkId /\* pageIndex \*/)->Count();

});

return markCount;

}

template <class Fn>

void

HeapBlockMap32::ForEachChunkInAddressRange(void \* address, size\_t pageCount, Fn fn)

{

uint id1 = GetLevel1Id(address);

uint id2 = GetLevel2Id(address);

while (true)

{

L2MapChunk \* l2map = map[id1];

Assert(l2map != nullptr);

if (l2map != nullptr)

{

while (id2 < L2Count)

{

fn(l2map, id2);

id2++;

pageCount--;

if (pageCount == 0)

{

return;

}

}

id2 = 0;

id1++;

}

}

}

bool

HeapBlockMap32::IsMarked(void \* address) const

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* chunk = map[id1];

Assert(chunk != nullptr);

return chunk->IsMarked(address);

}

void

HeapBlockMap32::SetMark(void \* address)

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* chunk = map[id1];

Assert(chunk != nullptr);

return chunk->SetMark(address);

}

bool

HeapBlockMap32::TestAndSetMark(void \* address)

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* chunk = map[id1];

if (chunk == nullptr)

{

// False reference

return false;

}

uint bitIndex = chunk->GetMarkBitIndex(address);

return (chunk->markBits.TestAndSet(bitIndex) != 0);

}

void

HeapBlockMap32::ResetMarks()

{

for (uint i = 0; i < L1Count; i++)

{

L2MapChunk \* chunk = map[i];

if (chunk == nullptr)

{

continue;

}

chunk->markBits.ClearAll();

#ifdef RECYCLER\_VERIFY\_MARK

chunk->isNewChunk = false;

#endif

#if DBG

for (uint j = 0; j < L2Count; j++)

{

chunk->pageMarkCount[j] = 0;

}

#endif

}

}

#if DBG

ushort

HeapBlockMap32::GetPageMarkCount(void \* address) const

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* l2map = map[id1];

Assert(l2map != nullptr);

uint id2 = GetLevel2Id(address);

return l2map->pageMarkCount[id2];

}

void

HeapBlockMap32::SetPageMarkCount(void \* address, ushort markCount)

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* l2map = map[id1];

Assert(l2map != nullptr);

uint id2 = GetLevel2Id(address);

// Callers should already have updated the mark bits by the time they call this,

// so check that the new count is correct for the current mark bits.

// Not true right now, will be true...

Assert(l2map->GetPageMarkBitVector(id2)->Count() == markCount);

l2map->pageMarkCount[id2] = markCount;

}

template void HeapBlockMap32::VerifyMarkCountForPages<SmallAllocationBlockAttributes::BitVectorCount>(void\* address, uint pageCount);

template void HeapBlockMap32::VerifyMarkCountForPages<MediumAllocationBlockAttributes::BitVectorCount>(void\* address, uint pageCount);

template <uint BitVectorCount>

void

HeapBlockMap32::VerifyMarkCountForPages(void \* address, uint pageCount)

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* l2map = map[id1];

Assert(l2map != nullptr);

uint id2 = GetLevel2Id(address);

Assert(id2 + pageCount <= L2Count);

for (uint i = id2; i < pageCount + id2; i++)

{

uint markCountForPage = l2map->GetPageMarkBitVector(i)->Count();

Assert(markCountForPage == l2map->pageMarkCount[i]);

}

}

#endif

HeapBlockMap32::L2MapChunk::L2MapChunk()

{

// We are zero-initialized so don't need to actually init.

// Mark bits should be cleared by default

Assert(markBits.Count() == 0);

#ifdef RECYCLER\_VERIFY\_MARK

this->isNewChunk = true;

#endif

#if DBG

for (uint i = 0; i < L2Count; i++)

{

Assert(pageMarkCount[i] == 0);

}

#endif

}

HeapBlockMap32::L2MapChunk::~L2MapChunk()

{

// In debug builds, we guarantee that the heap block is clear on shutdown.

// In free builds, we skip this to save time.

// So this assert is only true in debug builds.

Assert(IsEmpty());

}

HeapBlock \*

HeapBlockMap32::L2MapChunk::Get(void \* address)

{

uint id2 = GetLevel2Id(address);

Assert(id2 < L2Count);

\_\_analysis\_assume(id2 < L2Count);

return map[id2];

}

void

HeapBlockMap32::L2MapChunk::Set(uint id2, uint pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex)

{

uint id2End = id2 + pageCount;

Assert(id2 < L2Count);

Assert(id2End <= L2Count);

for (uint i = id2; i < id2End; i++)

{

\_\_analysis\_assume(i < L2Count);

Assert(map[i] == nullptr);

Assert(blockInfo[i].blockType == HeapBlock::HeapBlockType::FreeBlockType);

// Set the blockType last, because we will test this first during marking.

// If it's not FreeBlock, then we expect bucketIndex and heapBlock to be valid.

map[i] = heapBlock;

blockInfo[i].bucketIndex = bucketIndex;

// We need memory barrier here for ARM to ensure that the blockType is set last.

#if defined(\_M\_ARM32\_OR\_ARM64)

MemoryBarrier();

#endif

blockInfo[i].blockType = blockType;

}

}

void

HeapBlockMap32::L2MapChunk::Clear(uint id2, uint pageCount)

{

uint id2End = id2 + pageCount;

Assert(id2 < L2Count);

Assert(id2End <= L2Count);

for (uint i = id2; i < id2End; i++)

{

\_\_analysis\_assume(i < L2Count);

Assert(map[i] != nullptr);

Assert(blockInfo[i].blockType != HeapBlock::HeapBlockType::FreeBlockType);

// This shouldn't be called when concurrent marking is happening, so order does not matter.

// Regardless, set the blockType first just to be internally consistent.

// We don't actually clear the bucketIndex because it doesn't matter if the blockType is FreeBlock.

blockInfo[i].blockType = HeapBlock::HeapBlockType::FreeBlockType;

map[i] = nullptr;

}

}

bool

HeapBlockMap32::L2MapChunk::IsEmpty() const

{

for (uint i = 0; i < L2Count; i++)

{

if (map[i] != nullptr)

{

return false;

}

}

return true;

}

HeapBlockMap32::PageMarkBitVector \*

HeapBlockMap32::L2MapChunk::GetPageMarkBitVector(void \* address)

{

uint id2 = GetLevel2Id(address);

Assert(id2 < L2Count);

\_\_analysis\_assume(id2 < L2Count);

return GetPageMarkBitVector(id2);

}

HeapBlockMap32::PageMarkBitVector \*

HeapBlockMap32::L2MapChunk::GetPageMarkBitVector(uint pageIndex)

{

return markBits.GetRange<PageMarkBitCount>(pageIndex \* PageMarkBitCount);

}

template <size\_t BitCount>

BVStatic<BitCount> \*

HeapBlockMap32::L2MapChunk::GetMarkBitVectorForPages(void \* address)

{

uint id2 = GetLevel2Id(address);

Assert(id2 < L2Count);

\_\_analysis\_assume(id2 < L2Count);

return GetMarkBitVectorForPages<BitCount>(id2);

}

template <size\_t BitCount>

BVStatic<BitCount> \*

HeapBlockMap32::L2MapChunk::GetMarkBitVectorForPages(uint pageIndex)

{

return markBits.GetRange<BitCount>(pageIndex \* PageMarkBitCount);

}

bool

HeapBlockMap32::L2MapChunk::IsMarked(void \* address) const

{

return markBits.Test(GetMarkBitIndex(address)) == TRUE;

}

void

HeapBlockMap32::L2MapChunk::SetMark(void \* address)

{

markBits.Set(GetMarkBitIndex(address));

}

#ifdef RECYCLER\_STRESS

void

HeapBlockMap32::InduceFalsePositives(Recycler \* recycler)

{

for (uint i = 0; i < L1Count; i++)

{

L2MapChunk \* chunk = map[i];

if (chunk == nullptr)

{

continue;

}

for (uint j = 0; j < L2Count; j++)

{

HeapBlock \* block = chunk->map[j];

if (block == nullptr)

{

// Unallocated block. Try to mark the first offset, in case

// we are simultaneously allocating this block on the main thread.

recycler->TryMarkNonInterior((void \*)GetAddressFromIds(i, j), nullptr);

}

else if (!block->IsLargeHeapBlock())

{

((SmallHeapBlock \*)block)->InduceFalsePositive(recycler);

}

}

}

}

#endif

#ifdef RECYCLER\_VERIFY\_MARK

bool

HeapBlockMap32::IsAddressInNewChunk(void \* address)

{

uint id1 = GetLevel1Id(address);

L2MapChunk \* l2map = map[id1];

Assert(l2map != nullptr);

return l2map->isNewChunk;

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

template <class Fn>

void

HeapBlockMap32::ForEachSegment(Recycler \* recycler, Fn func)

{

Segment \* currentSegment = nullptr;

for (uint i = 0; i < L1Count; i++)

{

L2MapChunk \* chunk = map[i];

if (chunk == nullptr)

{

continue;

}

for (uint j = 0; j < L2Count; j++)

{

HeapBlock \* block = chunk->map[j];

if (block == nullptr)

{

continue;

}

Assert(block->GetSegment() != nullptr);

if (block->GetSegment() == currentSegment)

{

Assert(currentSegment != nullptr);

Assert(currentSegment->IsInSegment(block->GetAddress()));

continue;

}

// New segment.

Assert(currentSegment == nullptr || !currentSegment->IsInSegment(block->GetAddress()));

currentSegment = block->GetSegment();

AnalysisAssert(currentSegment != nullptr);

char \* segmentStart = currentSegment->GetAddress();

size\_t segmentLength = currentSegment->GetPageCount() \* PageSize;

PageAllocator\* segmentPageAllocator = (PageAllocator\*)currentSegment->GetAllocator();

Assert(segmentPageAllocator == block->GetPageAllocator(recycler));

#if defined(\_M\_X64\_OR\_ARM64)

// On 64 bit, the segment may span multiple HeapBlockMap32 structures.

// Limit the processing to the portion of the segment in this HeapBlockMap32.

// We'll process other portions when we visit the other HeapBlockMap32 structures.

if (segmentStart < this->startAddress)

{

Assert(segmentLength > (size\_t)(this->startAddress - segmentStart));

segmentLength -= (this->startAddress - segmentStart);

segmentStart = this->startAddress;

}

if ((segmentStart - this->startAddress) + segmentLength > HeapBlockMap32::TotalSize)

{

segmentLength = HeapBlockMap32::TotalSize - (segmentStart - this->startAddress);

}

#endif

func(segmentStart, segmentLength, currentSegment, segmentPageAllocator);

}

}

}

void

HeapBlockMap32::ResetWriteWatch(Recycler \* recycler)

{

this->ForEachSegment(recycler, [=] (char \* segmentStart, size\_t segmentLength, Segment \* segment, PageAllocator \* segmentPageAllocator) {

Assert(segmentLength % AutoSystemInfo::PageSize == 0);

if (segmentPageAllocator == recycler->GetRecyclerPageAllocator() ||

segmentPageAllocator == recycler->GetRecyclerLargeBlockPageAllocator())

{

// Call ResetWriteWatch for Small non-leaf and Large segments.

UINT ret = ::ResetWriteWatch(segmentStart, segmentLength);

Assert(ret == 0);

}

#ifdef RECYCLER\_WRITE\_BARRIER

else if (segmentPageAllocator == recycler->GetRecyclerWithBarrierPageAllocator())

{

// Reset software write barrier for barrier segments.

RecyclerWriteBarrierManager::ResetWriteBarrier(segmentStart, segmentLength / AutoSystemInfo::PageSize);

}

#endif

});

}

bool

HeapBlockMap32::RescanPage(void \* dirtyPage, bool\* anyObjectsMarkedOnPage, Recycler \* recycler)

{

uint id1 = GetLevel1Id(dirtyPage);

L2MapChunk \* chunk = map[id1];

if (chunk != nullptr)

{

uint id2 = GetLevel2Id(dirtyPage);

HeapBlock::HeapBlockType blockType = chunk->blockInfo[id2].blockType;

// Determine block type and process as appropriate

switch (blockType)

{

case HeapBlock::HeapBlockType::FreeBlockType:

// We had a false reference to a free block. Do nothing.

break;

case HeapBlock::HeapBlockType::SmallNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::SmallNormalBlockWithBarrierType:

#endif

return RescanHeapBlock<SmallNormalHeapBlock>(dirtyPage, blockType, chunk, id2, anyObjectsMarkedOnPage, recycler);

case HeapBlock::HeapBlockType::SmallFinalizableBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::SmallFinalizableBlockWithBarrierType:

#endif

return RescanHeapBlock<SmallFinalizableHeapBlock>(dirtyPage, blockType, chunk, id2, anyObjectsMarkedOnPage, recycler);

case HeapBlock::HeapBlockType::MediumNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::MediumNormalBlockWithBarrierType:

#endif

return RescanHeapBlock<MediumNormalHeapBlock>(dirtyPage, blockType, chunk, id2, anyObjectsMarkedOnPage, recycler);

case HeapBlock::HeapBlockType::MediumFinalizableBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::MediumFinalizableBlockWithBarrierType:

#endif

return RescanHeapBlock<MediumFinalizableHeapBlock>(dirtyPage, blockType, chunk, id2, anyObjectsMarkedOnPage, recycler);

default:

// Shouldn't be here -- leaf blocks aren't rescanned, and large blocks are handled separately

Assert(false);

break;

}

}

return false;

}

template bool HeapBlockMap32::RescanHeapBlock<SmallNormalHeapBlock>(void \* dirtyPage, HeapBlock::HeapBlockType blockType, L2MapChunk\* chunk, uint id2, bool\* anyObjectsMarkedOnPage, Recycler \* recycler);

template bool HeapBlockMap32::RescanHeapBlock<SmallFinalizableHeapBlock>(void \* dirtyPage, HeapBlock::HeapBlockType blockType, L2MapChunk\* chunk, uint id2, bool\* anyObjectsMarkedOnPage, Recycler \* recycler);

template bool HeapBlockMap32::RescanHeapBlock<MediumNormalHeapBlock>(void \* dirtyPage, HeapBlock::HeapBlockType blockType, L2MapChunk\* chunk, uint id2, bool\* anyObjectsMarkedOnPage, Recycler \* recycler);

template bool HeapBlockMap32::RescanHeapBlock<MediumFinalizableHeapBlock>(void \* dirtyPage, HeapBlock::HeapBlockType blockType, L2MapChunk\* chunk, uint id2, bool\* anyObjectsMarkedOnPage, Recycler \* recycler);

template <class TBlockType>

bool

HeapBlockMap32::RescanHeapBlock(void \* dirtyPage, HeapBlock::HeapBlockType blockType, L2MapChunk\* chunk, uint id2, bool\* anyObjectsMarkedOnPage, Recycler \* recycler)

{

Assert(chunk != nullptr);

char\* heapBlockPageAddress = TBlockType::GetBlockStartAddress((char\*) dirtyPage);

typedef TBlockType::HeapBlockAttributes TBlockAttributes;

// We need to check the entire mark bit vector here. It's not sufficient to just check the page's

// mark bit vector because the object that's dirty on the page could have started on an earlier page

auto markBits = chunk->GetMarkBitVectorForPages<TBlockAttributes::BitVectorCount>(heapBlockPageAddress);

if (!markBits->IsAllClear())

{

Assert(chunk->map[id2]->GetHeapBlockType() == blockType);

// Small finalizable heap blocks require the HeapBlock \* (to look up object attributes).

// For others, this is null

TBlockType\* block = GetHeapBlockForRescan<TBlockType>(chunk, id2);

uint bucketIndex = chunk->blockInfo[id2].bucketIndex;

if (!SmallNormalHeapBucketBase<TBlockType>::RescanObjectsOnPage(block,

(char \*)dirtyPage, heapBlockPageAddress, markBits, HeapInfo::GetObjectSizeForBucketIndex<TBlockAttributes>(bucketIndex), bucketIndex, anyObjectsMarkedOnPage, recycler))

{

// Failed due to OOM

((TBlockType\*) chunk->map[id2])->SetNeedOOMRescan(recycler);

return false;

}

return true;

}

// Didn't actually rescan the block.

return false;

}

template <typename TBlockType>

TBlockType\*

HeapBlockMap32::GetHeapBlockForRescan(HeapBlockMap32::L2MapChunk\* chunk, uint id2) const

{

return nullptr;

}

template <>

SmallFinalizableHeapBlock\*

HeapBlockMap32::GetHeapBlockForRescan(HeapBlockMap32::L2MapChunk\* chunk, uint id2) const

{

return (SmallFinalizableHeapBlock\*) chunk->map[id2];

}

template <>

MediumFinalizableHeapBlock\*

HeapBlockMap32::GetHeapBlockForRescan(HeapBlockMap32::L2MapChunk\* chunk, uint id2) const

{

return (MediumFinalizableHeapBlock\*)chunk->map[id2];

}

void

HeapBlockMap32::MakeAllPagesReadOnly(Recycler\* recycler)

{

this->ChangeProtectionLevel(recycler, PAGE\_READONLY, PAGE\_READWRITE);

}

void

HeapBlockMap32::MakeAllPagesReadWrite(Recycler\* recycler)

{

this->ChangeProtectionLevel(recycler, PAGE\_READWRITE, PAGE\_READONLY);

}

void

HeapBlockMap32::ChangeProtectionLevel(Recycler\* recycler, DWORD protectFlags, DWORD expectedOldFlags)

{

this->ForEachSegment(recycler, [&](char\* segmentStart, size\_t segmentLength, Segment\* currentSegment, PageAllocator\* segmentPageAllocator)

{

// Ideally, we shouldn't to exclude LargeBlocks here but guest arenas are allocated

// from this allocator and we touch them during marking if they're pending delete

if ((segmentPageAllocator != recycler->GetRecyclerLeafPageAllocator())

&& (segmentPageAllocator != recycler->GetRecyclerLargeBlockPageAllocator()))

{

Assert(currentSegment->IsPageSegment());

((PageSegment\*)currentSegment)->ChangeSegmentProtection(protectFlags, expectedOldFlags);

}

});

}

///

/// The GetWriteWatch API can fail under low-mem situations if called to retrieve write-watch for a large number of pages

/// (On Win10, > 255 pages). This helper is to handle the failure case. In the case of failure, we degrade to retrieving

/// the write-watch one page at a time since that's expected to succeed

///

UINT

HeapBlockMap32::GetWriteWatchHelper(Recycler \* recycler, DWORD writeWatchFlags, void\* baseAddress, size\_t regionSize,

void\*\* addresses, ULONG\_PTR\* count, LPDWORD granularity)

{

UINT ret = 0;

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (recycler->GetRecyclerFlagsTable().ForceGetWriteWatchOOM)

{

if (regionSize != AutoSystemInfo::PageSize)

{

ret = (UINT) -1;

}

}

else

#endif

{

ret = ::GetWriteWatch(writeWatchFlags, baseAddress, regionSize, addresses, count, granularity);

}

if (ret != 0 && regionSize != AutoSystemInfo::PageSize)

{

ret = GetWriteWatchHelperOnOOM(writeWatchFlags, baseAddress, regionSize, addresses, count, granularity);

}

Assert(ret == 0);

return ret;

}

// OOM codepath- Retrieve write-watch one page at a time

// It's slow, but we are okay with that during OOM

// Factored into its own function to help the compiler inline the parent

UINT

HeapBlockMap32::GetWriteWatchHelperOnOOM(DWORD writeWatchFlags, \_In\_ void\* baseAddress, size\_t regionSize,

\_Out\_writes\_(\*count) void\*\* addresses, \_Inout\_ ULONG\_PTR\* count, LPDWORD granularity)

{

const size\_t pageCount = (regionSize / AutoSystemInfo::PageSize);

// Ensure target buffer

AnalysisAssertMsg(\*count >= pageCount, "Not enough space in the buffer to store the write watch state for the given region size");

void\* result = nullptr;

size\_t dirtyCount = 0;

for (size\_t i = 0; i < pageCount; i++)

{

result = nullptr;

char\* pageAddress = ((char\*)baseAddress) + (i \* AutoSystemInfo::PageSize);

ULONG\_PTR resultBufferCount = 1;

DWORD r = ::GetWriteWatch(writeWatchFlags, pageAddress, AutoSystemInfo::PageSize, &result, &resultBufferCount, granularity);

Assert(r == 0);

Assert(resultBufferCount <= 1);

AnalysisAssert(dirtyCount <= pageCount);

// The requested page was dirty

if (resultBufferCount == 1)

{

Assert(result == pageAddress);

addresses[dirtyCount] = pageAddress;

dirtyCount++;

}

}

Assert(dirtyCount <= \*count);

\*count = dirtyCount;

return 0;

}

uint

HeapBlockMap32::Rescan(Recycler \* recycler, bool resetWriteWatch)

{

// Loop through segments and find dirty pages.

const DWORD writeWatchFlags = (resetWriteWatch ? WRITE\_WATCH\_FLAG\_RESET : 0);

uint scannedPageCount = 0;

bool anyObjectsScannedOnPage = false;

this->ForEachSegment(recycler, [&] (char \* segmentStart, size\_t segmentLength, Segment \* currentSegment, PageAllocator \* segmentPageAllocator) {

Assert(segmentLength % AutoSystemInfo::PageSize == 0);

// Call GetWriteWatch for Small non-leaf segments.

// Large blocks have their own separate write watch handling.

if (segmentPageAllocator == recycler->GetRecyclerPageAllocator())

{

// array for WW results

void \* dirtyPageAddresses[MaxGetWriteWatchPages];

Assert(segmentLength <= MaxGetWriteWatchPages \* PageSize);

ULONG\_PTR pageCount = MaxGetWriteWatchPages;

DWORD pageSize = PageSize;

UINT ret = HeapBlockMap32::GetWriteWatchHelper(recycler, writeWatchFlags, segmentStart, segmentLength, dirtyPageAddresses, &pageCount, &pageSize);

Assert(ret == 0);

Assert(pageSize == PageSize);

Assert(pageCount <= MaxGetWriteWatchPages);

// Process results:

// Loop through reported dirty pages and set their write watch bit.

for (uint i = 0; i < pageCount; i++)

{

char \* dirtyPage = (char \*)dirtyPageAddresses[i];

Assert((((size\_t)dirtyPage) % PageSize) == 0);

Assert(dirtyPage >= segmentStart);

Assert(dirtyPage < segmentStart + segmentLength);

#if defined(\_M\_X64\_OR\_ARM64)

Assert(HeapBlockMap64::GetNodeStartAddress(dirtyPage) == this->startAddress);

#endif

if (RescanPage(dirtyPage, &anyObjectsScannedOnPage, recycler) && anyObjectsScannedOnPage)

{

scannedPageCount++;

}

}

}

#ifdef RECYCLER\_WRITE\_BARRIER

else if (segmentPageAllocator == recycler->GetRecyclerWithBarrierPageAllocator())

{

// Loop through pages for this segment and check write barrier.

size\_t pageCount = segmentLength / AutoSystemInfo::PageSize;

for (size\_t i = 0; i < pageCount; i++)

{

char \* pageAddress = segmentStart + (i \* AutoSystemInfo::PageSize);

Assert((size\_t)(pageAddress - segmentStart) < segmentLength);

#if defined(\_M\_X64\_OR\_ARM64)

Assert(HeapBlockMap64::GetNodeStartAddress(pageAddress) == this->startAddress);

#endif

// TODO: We are not resetting the write barrier here when RescanFlags\_ResetWriteWatch is passed.

// We never have previously, but it still seems like we should.

BYTE writeBarrierByte = RecyclerWriteBarrierManager::GetWriteBarrier(pageAddress);

SwbVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Address: 0x%p, Write Barrier value: %u\n", pageAddress, writeBarrierByte);

bool isDirty = (writeBarrierByte == 1);

if (isDirty)

{

if (RescanPage(pageAddress, &anyObjectsScannedOnPage, recycler) && anyObjectsScannedOnPage)

{

scannedPageCount++;

}

}

}

}

#endif

else

{

Assert(segmentPageAllocator == recycler->GetRecyclerLeafPageAllocator() ||

segmentPageAllocator == recycler->GetRecyclerLargeBlockPageAllocator());

}

});

return scannedPageCount;

}

bool

HeapBlockMap32::OOMRescan(Recycler \* recycler)

{

this->anyHeapBlockRescannedDuringOOM = false;

bool noHeapBlockNeedsRescan = true;

// Loop through segments and find pages that need OOM Rescan.

this->ForEachSegment(recycler, [=, &noHeapBlockNeedsRescan] (char \* segmentStart, size\_t segmentLength, Segment \* currentSegment, PageAllocator \* segmentPageAllocator) {

Assert(segmentLength % AutoSystemInfo::PageSize == 0);

// Process Small non-leaf segments (including write barrier blocks).

// Large blocks have their own separate write watch handling.

if (segmentPageAllocator == recycler->GetRecyclerPageAllocator()

#ifdef RECYCLER\_WRITE\_BARRIER

|| segmentPageAllocator == recycler->GetRecyclerWithBarrierPageAllocator()

#endif

)

{

if (recycler->NeedOOMRescan())

{

// We hit OOM again. Don't try to process any more blocks, leave them for the next OOM pass.

return;

}

// Loop through pages for this segment and check OOM flag.

size\_t pageCount = segmentLength / AutoSystemInfo::PageSize;

for (size\_t i = 0; i < pageCount; i++)

{

char \* pageAddress = segmentStart + (i \* AutoSystemInfo::PageSize);

Assert((size\_t)(pageAddress - segmentStart) < segmentLength);

#if defined(\_M\_X64\_OR\_ARM64)

Assert(HeapBlockMap64::GetNodeStartAddress(pageAddress) == this->startAddress);

#endif

uint id1 = GetLevel1Id(pageAddress);

L2MapChunk \* chunk = map[id1];

if (chunk != nullptr)

{

uint id2 = GetLevel2Id(pageAddress);

HeapBlock \* heapBlock = chunk->map[id2];

if (heapBlock != nullptr && heapBlock->GetAddress() == pageAddress)

{

if (heapBlock->GetAndClearNeedOOMRescan())

{

noHeapBlockNeedsRescan = false;

HeapBlock::HeapBlockType blockType = chunk->blockInfo[id2].blockType;

// Determine block type and process as appropriate

switch (blockType)

{

case HeapBlock::HeapBlockType::FreeBlockType:

// Can't have a free block that has OOMRescan flag set

Assert(false);

break;

case HeapBlock::HeapBlockType::SmallNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::SmallNormalBlockWithBarrierType:

#endif

if (!RescanHeapBlockOnOOM<SmallNormalHeapBlock>((SmallNormalHeapBlock\*)heapBlock, pageAddress, blockType, chunk->blockInfo[id2].bucketIndex, chunk, recycler))

{

return;

}

break;

case HeapBlock::HeapBlockType::SmallFinalizableBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::SmallFinalizableBlockWithBarrierType:

#endif

if (!RescanHeapBlockOnOOM<SmallFinalizableHeapBlock>((SmallFinalizableHeapBlock\*) heapBlock, pageAddress, blockType, chunk->blockInfo[id2].bucketIndex, chunk, recycler))

{

return;

}

break;

case HeapBlock::HeapBlockType::MediumNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::MediumNormalBlockWithBarrierType:

#endif

if (!RescanHeapBlockOnOOM<MediumNormalHeapBlock>((MediumNormalHeapBlock\*)heapBlock, pageAddress, blockType, chunk->blockInfo[id2].bucketIndex, chunk, recycler))

{

return;

}

break;

case HeapBlock::HeapBlockType::MediumFinalizableBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::MediumFinalizableBlockWithBarrierType:

#endif

if (!RescanHeapBlockOnOOM<MediumFinalizableHeapBlock>((MediumFinalizableHeapBlock\*) heapBlock, pageAddress, blockType, chunk->blockInfo[id2].bucketIndex, chunk, recycler))

{

return;

}

break;

default:

// Shouldn't be here -- leaf blocks aren't rescanned, and large blocks are handled separately

Assert(false);

break;

}

}

}

}

}

}

else

{

Assert(segmentPageAllocator == recycler->GetRecyclerLeafPageAllocator() ||

segmentPageAllocator == recycler->GetRecyclerLargeBlockPageAllocator());

}

});

// TODO: Enable this assert post-Win10

// Assert(this->anyHeapBlockRescannedDuringOOM);

// Success if:

// No heap block needs OOM rescan OR

// A single heap block was rescanned

return noHeapBlockNeedsRescan || this->anyHeapBlockRescannedDuringOOM;

}

template bool HeapBlockMap32::RescanHeapBlockOnOOM<SmallNormalHeapBlock>(SmallNormalHeapBlock\* heapBlock, char\* pageAddress, HeapBlock::HeapBlockType blockType, uint bucketIndex, L2MapChunk \* chunk, Recycler \* recycler);

template bool HeapBlockMap32::RescanHeapBlockOnOOM<SmallFinalizableHeapBlock>(SmallFinalizableHeapBlock\* heapBlock, char\* pageAddress, HeapBlock::HeapBlockType blockType, uint bucketIndex, L2MapChunk \* chunk, Recycler \* recycler);

template bool HeapBlockMap32::RescanHeapBlockOnOOM<MediumNormalHeapBlock>(MediumNormalHeapBlock\* heapBlock, char\* pageAddress, HeapBlock::HeapBlockType blockType, uint bucketIndex, L2MapChunk \* chunk, Recycler \* recycler);

template bool HeapBlockMap32::RescanHeapBlockOnOOM<MediumFinalizableHeapBlock>(MediumFinalizableHeapBlock\* heapBlock, char\* pageAddress, HeapBlock::HeapBlockType blockType, uint bucketIndex, L2MapChunk \* chunk, Recycler \* recycler);

template <class TBlockType>

bool

HeapBlockMap32::RescanHeapBlockOnOOM(TBlockType\* heapBlock, char\* pageAddress, HeapBlock::HeapBlockType blockType, uint bucketIndex, L2MapChunk \* chunk, Recycler \* recycler)

{

// In the OOM codepath, we expect the heap block to be dereferenced since perf is not critical

Assert(heapBlock != nullptr);

Assert(heapBlock->GetHeapBlockType() == blockType);

auto markBits = this->GetMarkBitVectorForPages<TBlockType::HeapBlockAttributes::BitVectorCount>(heapBlock->GetAddress());

char\* blockStartAddress = TBlockType::GetBlockStartAddress(pageAddress);

// Rescan all pages in this block

// The following assert makes sure that this method is called only once per heap block

Assert(blockStartAddress == pageAddress);

for (int i = 0; i < TBlockType::HeapBlockAttributes::PageCount; i++)

{

char\* pageAddressToScan = blockStartAddress + (i \* AutoSystemInfo::PageSize);

if (!SmallNormalHeapBucketBase<TBlockType>::RescanObjectsOnPage(heapBlock,

pageAddressToScan, blockStartAddress, markBits, HeapInfo::GetObjectSizeForBucketIndex<TBlockType::HeapBlockAttributes>(bucketIndex), bucketIndex, nullptr, recycler))

{

// Failed due to OOM

((TBlockType\*)heapBlock)->SetNeedOOMRescan(recycler);

}

if (recycler->NeedOOMRescan())

{

// We hit OOM again. Don't try to process any more blocks, leave them for the next OOM pass.

return false;

}

}

this->anyHeapBlockRescannedDuringOOM = true;

return true;

}

#endif

// This function is called in-thread after Sweep, to find empty L2 Maps and release them.

void

HeapBlockMap32::Cleanup(bool concurrentFindImplicitRoot)

{

for (uint id1 = 0; id1 < L1Count; id1++)

{

L2MapChunk \* l2map = map[id1];

if (l2map != nullptr && l2map->IsEmpty())

{

// Concurrent searches for implicit roots will never see empty L2 maps.

map[id1] = nullptr;

NoMemProtectHeapDelete(l2map);

Assert(count > 0);

count--;

}

}

}

#if defined(\_M\_X64\_OR\_ARM64)

HeapBlockMap64::HeapBlockMap64():

list(nullptr)

{

}

HeapBlockMap64::~HeapBlockMap64()

{

Node \* node = list;

list = nullptr;

while (node != nullptr)

{

Node \* next = node->next;

NoMemProtectHeapDelete(node);

node = next;

}

}

bool

HeapBlockMap64::EnsureHeapBlock(void \* address, size\_t pageCount)

{

uint lowerBitsAddress = ::Math::PointerCastToIntegralTruncate<uint>(address);

size\_t pageCountLeft = pageCount;

uint nodePages = HeapBlockMap64::PagesPer4GB - lowerBitsAddress / AutoSystemInfo::PageSize;

if (pageCountLeft < nodePages)

{

nodePages = (uint)pageCountLeft;

}

do

{

Node \* node = FindOrInsertNode(address);

if (node == nullptr || !node->map.EnsureHeapBlock(address, nodePages))

{

return false;

}

pageCountLeft -= nodePages;

if (pageCountLeft == 0)

{

return true;

}

address = (void \*)((size\_t)address + (nodePages \* AutoSystemInfo::PageSize));

nodePages = HeapBlockMap64::PagesPer4GB;

if (pageCountLeft < HeapBlockMap64::PagesPer4GB)

{

nodePages = (uint)pageCountLeft;

}

}

while (true);

}

void

HeapBlockMap64::SetHeapBlockNoCheck(void \* address, size\_t pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex)

{

ForEachNodeInAddressRange(address, pageCount, [&](Node \* node, void \* address, uint nodePages)

{

Assert(node != nullptr);

node->map.SetHeapBlockNoCheck(address, nodePages, heapBlock, blockType, bucketIndex);

});

}

bool

HeapBlockMap64::SetHeapBlock(void \* address, size\_t pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex)

{

if (!EnsureHeapBlock(address, pageCount))

{

return false;

}

SetHeapBlockNoCheck(address, pageCount, heapBlock, blockType, bucketIndex);

return true;

}

void HeapBlockMap64::ClearHeapBlock(void \* address, size\_t pageCount)

{

ForEachNodeInAddressRange(address, pageCount, [&](Node\* node, void\* address, uint nodePages)

{

Assert(node != nullptr);

node->map.ClearHeapBlock(address, nodePages);

});

}

template <class Fn>

void HeapBlockMap64::ForEachNodeInAddressRange(void \* address, size\_t pageCount, Fn fn)

{

uint lowerBitsAddress = ::Math::PointerCastToIntegralTruncate<uint>(address);

uint nodePages = HeapBlockMap64::PagesPer4GB - lowerBitsAddress / AutoSystemInfo::PageSize;

if (pageCount < nodePages)

{

nodePages = (uint)pageCount;

}

do

{

Node \* node = FindNode(address);

fn(node, address, nodePages);

pageCount -= nodePages;

if (pageCount == 0)

{

break;

}

address = (void \*)((size\_t)address + (nodePages \* AutoSystemInfo::PageSize));

nodePages = HeapBlockMap64::PagesPer4GB;

if (pageCount < HeapBlockMap64::PagesPer4GB)

{

nodePages = (uint)pageCount;

}

} while (true);

}

HeapBlock \*

HeapBlockMap64::GetHeapBlock(void \* address)

{

Node \* node = FindNode(address);

if (node == nullptr)

{

return nullptr;

}

return node->map.GetHeapBlock(address);

}

HeapBlockMap32::PageMarkBitVector \*

HeapBlockMap64::GetPageMarkBitVector(void \* address)

{

Node \* node = FindNode(address);

Assert(node != nullptr);

return node->map.GetPageMarkBitVector(address);

}

template <size\_t BitCount>

BVStatic<BitCount>\* HeapBlockMap64::GetMarkBitVectorForPages(void \* address)

{

Node \* node = FindNode(address);

Assert(node != nullptr);

return node->map.GetMarkBitVectorForPages<BitCount>(address);

}

template BVStatic<SmallAllocationBlockAttributes::BitVectorCount>\* HeapBlockMap64::GetMarkBitVectorForPages<SmallAllocationBlockAttributes::BitVectorCount>(void \* address);

template BVStatic<MediumAllocationBlockAttributes::BitVectorCount>\* HeapBlockMap64::GetMarkBitVectorForPages<MediumAllocationBlockAttributes::BitVectorCount>(void \* address);

uint

HeapBlockMap64::GetMarkCount(void \* address, uint pageCount)

{

uint markCount = 0;

ForEachNodeInAddressRange(address, pageCount, [&](Node\* node, void\* address, uint nodePageCount)

{

Assert(node != nullptr);

markCount += node->map.GetMarkCount(address, nodePageCount);

});

return markCount;

}

bool

HeapBlockMap64::IsMarked(void \* address) const

{

Node \* node = FindNode(address);

if (node != nullptr)

{

return node->map.IsMarked(address);

}

return false;

}

void

HeapBlockMap64::SetMark(void \* address)

{

Node \* node = FindNode(address);

if (node != nullptr)

{

node->map.SetMark(address);

}

}

bool

HeapBlockMap64::TestAndSetMark(void \* address)

{

Node \* node = FindNode(address);

if (node == nullptr)

{

return false;

}

return node->map.TestAndSetMark(address);

}

HeapBlockMap64::Node \*

HeapBlockMap64::FindOrInsertNode(void \* address)

{

Node \* node = FindNode(address);

if (node == nullptr)

{

node = NoMemProtectHeapNewNoThrowZ(Node, GetNodeStartAddress(address));

if (node != nullptr)

{

node->nodeIndex = GetNodeIndex(address);

node->next = list;

#ifdef \_M\_ARM64

// For ARM we need to make sure that the list remains traversable during this insert.

MemoryBarrier();

#endif

list = node;

}

}

return node;

}

HeapBlockMap64::Node \*

HeapBlockMap64::FindNode(void \* address) const

{

uint index = GetNodeIndex(address);

Node \* node = list;

while (node != nullptr)

{

if (node->nodeIndex == index)

{

return node;

}

node = node->next;

}

return nullptr;

}

void

HeapBlockMap64::ResetMarks()

{

Node \* node = this->list;

while (node != nullptr)

{

node->map.ResetMarks();

node = node->next;

}

}

#ifdef CONCURRENT\_GC\_ENABLED

void

HeapBlockMap64::ResetWriteWatch(Recycler \* recycler)

{

Node \* node = this->list;

while (node != nullptr)

{

node->map.ResetWriteWatch(recycler);

node = node->next;

}

}

void

HeapBlockMap64::MakeAllPagesReadOnly(Recycler\* recycler)

{

Node \* node = this->list;

while (node != nullptr)

{

node->map.MakeAllPagesReadOnly(recycler);

node = node->next;

}

}

void

HeapBlockMap64::MakeAllPagesReadWrite(Recycler\* recycler)

{

Node \* node = this->list;

while (node != nullptr)

{

node->map.MakeAllPagesReadWrite(recycler);

node = node->next;

}

}

uint

HeapBlockMap64::Rescan(Recycler \* recycler, bool resetWriteWatch)

{

uint scannedPageCount = 0;

Node \* node = this->list;

while (node != nullptr)

{

scannedPageCount += node->map.Rescan(recycler, resetWriteWatch);

node = node->next;

}

return scannedPageCount;

}

bool

HeapBlockMap64::OOMRescan(Recycler \* recycler)

{

Node \* node = this->list;

while (node != nullptr)

{

if (!node->map.OOMRescan(recycler))

{

return false;

}

node = node->next;

}

return true;

}

void

HeapBlockMap64::Cleanup(bool concurrentFindImplicitRoot)

{

Node \*\* prevnext = &this->list;

Node \* node = \*prevnext;

while (node != nullptr)

{

node->map.Cleanup(concurrentFindImplicitRoot);

Node \* nextNode = node->next;

if (!concurrentFindImplicitRoot && node->map.Empty())

{

// Concurrent traversals of the node list would result in a race and possible UAF.

// Currently we simply defer node free for the lifetime of the heap (only affects MemProtect).

\*prevnext = node->next;

NoMemProtectHeapDelete(node);

}

else

{

prevnext = &node->next;

}

node = nextNode;

}

}

#if DBG

ushort

HeapBlockMap64::GetPageMarkCount(void \* address) const

{

Node \* node = FindNode(address);

Assert(node != nullptr);

return node->map.GetPageMarkCount(address);

}

void

HeapBlockMap64::SetPageMarkCount(void \* address, ushort markCount)

{

Node \* node = FindNode(address);

Assert(node != nullptr);

node->map.SetPageMarkCount(address, markCount);

}

template void HeapBlockMap64::VerifyMarkCountForPages<SmallAllocationBlockAttributes::BitVectorCount>(void\* address, uint pageCount);

template void HeapBlockMap64::VerifyMarkCountForPages<MediumAllocationBlockAttributes::BitVectorCount>(void\* address, uint pageCount);

template <uint BitVectorCount>

void

HeapBlockMap64::VerifyMarkCountForPages(void \* address, uint pageCount)

{

Node \* node = FindNode(address);

Assert(node != nullptr);

node->map.VerifyMarkCountForPages<BitVectorCount>(address, pageCount);

}

#endif

#endif

#ifdef RECYCLER\_STRESS

void

HeapBlockMap64::InduceFalsePositives(Recycler \* recycler)

{

Node \* node = this->list;

while (node != nullptr)

{

node->map.InduceFalsePositives(recycler);

node = node->next;

}

}

#endif

#ifdef RECYCLER\_VERIFY\_MARK

bool

HeapBlockMap64::IsAddressInNewChunk(void \* address)

{

Node \* node = FindNode(address);

Assert(node != nullptr);

return node->map.IsAddressInNewChunk(address);

}

#endif

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CommonDefines.h"

namespace Memory

{

class HeapBlockMap32

{

public:

// Segment mapping

static const uint L1Count = 4096;

static const uint L2Count = 256;

static const uint PageSize = AutoSystemInfo::PageSize; // 4096

// Mark bit definitions

static const uint PageMarkBitCount = PageSize / HeapConstants::ObjectGranularity;

static const uint L2ChunkMarkBitCount = L2Count \* PageMarkBitCount;

#if defined(\_M\_X64\_OR\_ARM64) && !defined(JD\_PRIVATE)

static const size\_t TotalSize = 0x100000000; // 4GB

#endif

typedef BVStatic<PageMarkBitCount> PageMarkBitVector;

typedef BVStatic<L2ChunkMarkBitCount> L2ChunkMarkBitVector;

// Max size of GetWriteWatch batching.

// This must be >= PageSegment page count for small heap block segments,

// so set it to the MaxPageCount for PageSegments.

static const uint MaxGetWriteWatchPages = PageSegment::MaxPageCount;

#if defined(\_M\_X64\_OR\_ARM64)

HeapBlockMap32(\_\_in char \* startAddress);

#else

HeapBlockMap32();

#endif

~HeapBlockMap32();

bool EnsureHeapBlock(void \* address, uint pageCount);

void SetHeapBlockNoCheck(void \* address, uint pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex);

bool SetHeapBlock(void \* address, uint pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex);

void ClearHeapBlock(void \* address, uint pageCount);

HeapBlock \* GetHeapBlock(void \* address);

bool Empty() const { return count == 0; }

// Heap block map marking

PageMarkBitVector \* GetPageMarkBitVector(void \* address);

template <size\_t BitCount>

BVStatic<BitCount>\* GetMarkBitVectorForPages(void \* address);

uint GetMarkCount(void\* address, uint pageCount);

template <bool interlocked>

void Mark(void \* candidate, MarkContext \* markContext);

template <bool interlocked>

void MarkInterior(void \* candidate, MarkContext \* markContext);

bool IsMarked(void \* address) const;

void SetMark(void \* address);

bool TestAndSetMark(void \* address);

void ResetMarks();

#ifdef CONCURRENT\_GC\_ENABLED

void ResetWriteWatch(Recycler \* recycler);

uint Rescan(Recycler \* recycler, bool resetWriteWatch);

void MakeAllPagesReadOnly(Recycler\* recycler);

void MakeAllPagesReadWrite(Recycler\* recycler);

#endif

void Cleanup(bool concurrentFindImplicitRoot);

bool OOMRescan(Recycler \* recycler);

#ifdef RECYCLER\_STRESS

void InduceFalsePositives(Recycler \* recycler);

#endif

#ifdef RECYCLER\_VERIFY\_MARK

bool IsAddressInNewChunk(void \* address);

#endif

private:

friend class PageSegmentBase<VirtualAllocWrapper>;

#ifdef JD\_PRIVATE

friend class EXT\_CLASS;

friend class HeapBlockHelper;

#endif

template <class Fn>

void ForEachSegment(Recycler \* recycler, Fn func);

void ChangeProtectionLevel(Recycler\* recycler, DWORD protectFlags, DWORD expectedOldFlags);

static uint GetLevel1Id(void \* address)

{

return ::Math::PointerCastToIntegralTruncate<uint>(address) / L2Count / PageSize;

}

public:

static uint GetLevel2Id(void \* address)

{

return ::Math::PointerCastToIntegralTruncate<uint>(address) % (L2Count \* PageSize) / PageSize;

}

private:

static UINT GetWriteWatchHelper(Recycler \* recycler, DWORD writeWatchFlags, void\* baseAddress, size\_t regionSize,

void\*\* addresses, ULONG\_PTR\* count, LPDWORD granularity);

static UINT GetWriteWatchHelperOnOOM(DWORD writeWatchFlags, \_In\_ void\* baseAddress, size\_t regionSize,

\_Out\_writes\_(\*count) void\*\* addresses, \_Inout\_ ULONG\_PTR\* count, LPDWORD granularity);

static void \* GetAddressFromIds(uint id1, uint id2)

{

Assert(id1 < L1Count);

Assert(id2 < L2Count);

return (void \*)(((id1 \* L2Count) + id2) \* PageSize);

}

struct HeapBlockInfo

{

HeapBlock::HeapBlockType blockType;

byte bucketIndex;

};

// We want HeapBlockInfo to be as small as possible to get the best cache locality for this info.

CompileAssert(sizeof(HeapBlockInfo) == sizeof(ushort));

class L2MapChunk

{

public:

L2MapChunk();

~L2MapChunk();

HeapBlock \* Get(void \* address);

void Set(uint id2, uint pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex);

void Clear(uint id2, uint pageCount);

bool IsEmpty() const;

PageMarkBitVector \* GetPageMarkBitVector(void \* address);

PageMarkBitVector \* GetPageMarkBitVector(uint pageIndex);

template <size\_t BitCount> BVStatic<BitCount>\* GetMarkBitVectorForPages(void \* address);

template <size\_t BitCount> BVStatic<BitCount>\* GetMarkBitVectorForPages(uint pageIndex);

bool IsMarked(void \* address) const;

void SetMark(void \* address);

static uint GetMarkBitIndex(void \* address)

{

uint bitIndex = (::Math::PointerCastToIntegralTruncate<uint>(address) % (L2Count \* PageSize)) / HeapConstants::ObjectGranularity;

Assert(bitIndex < L2ChunkMarkBitCount);

return bitIndex;

}

// Mark bits for objects that live in this particular chunk

// Each L2 chunk has 1 bit for each object that lives in this chunk

// The L2 chunk represents 256 \* 4096 bytes = 1 MB of address space

// This means, that on 32 bit systems, where each object is at least 16 bytes, we can have 64K objects

// On 64 bit systems, where each object is at least 32 bytes, we can have 32K objects

// Therefore, that on 32 bit systems, the mark bits take up 8KB, and on 64 bit systems, they take up 4KB

// This is more general purpose than if the mark bits are on the heap block, and is more runtime efficient

// However, it's less memory efficient (e.g. a large object which is 1 MB + 1 byte, rounded up to 1028 KB,

// would before take up 1 byte for it's mark bits- now it'll have a cost of 16KB, one for each of the L2 segments it spans)

L2ChunkMarkBitVector markBits;

// HeapBlockInfo for each page in our range

HeapBlockInfo blockInfo[L2Count];

// HeapBlock \* for each page in our range (or nullptr, if no block)

HeapBlock\* map[L2Count];

#ifdef RECYCLER\_VERIFY\_MARK

bool isNewChunk;

#endif

#if DBG

ushort pageMarkCount[L2Count];

#endif

};

template <bool interlocked>

bool MarkInternal(L2MapChunk \* chunk, void \* candidate);

template <bool interlocked, bool updateChunk>

bool MarkInteriorInternal(MarkContext \* markContext, L2MapChunk \*& chunk, void \* originalCandidate, void \* realCandidate);

template <typename TBlockType>

TBlockType\* GetHeapBlockForRescan(L2MapChunk\* chunk, uint id2) const;

template <class TBlockType>

bool RescanHeapBlock(void \* dirtyPage, HeapBlock::HeapBlockType blockType, L2MapChunk\* chunk, uint id2, bool\* anyObjectsMarkedOnPage, Recycler \* recycler);

template <class TBlockType>

bool RescanHeapBlockOnOOM(TBlockType\* heapBlock, char\* pageAddress, HeapBlock::HeapBlockType blockType, uint bucketIndex, L2MapChunk \* chunk, Recycler \* recycler);

bool RescanPage(void \* dirtyPage, bool\* anyObjectsMarkedOnPage, Recycler \* recycler);

uint count;

L2MapChunk \* map[L1Count];

bool anyHeapBlockRescannedDuringOOM;

#if defined(\_M\_X64\_OR\_ARM64)

// On 64 bit, this structure only maps one particular 32 bit space.

// Store the startAddress of that 32 bit space so we know which it is.

// This value should always be 4GB aligned.

char \* startAddress;

#endif

public:

#if DBG

ushort GetPageMarkCount(void \* address) const;

void SetPageMarkCount(void \* address, ushort markCount);

template <uint BitVectorCount>

void VerifyMarkCountForPages(void \* address, uint pageCount);

#endif

private:

template <class Fn>

void ForEachChunkInAddressRange(void \* address, size\_t pageCount, Fn fn);

};

#if defined(\_M\_X64\_OR\_ARM64)

class HeapBlockMap64

{

public:

HeapBlockMap64();

~HeapBlockMap64();

bool EnsureHeapBlock(void \* address, size\_t pageCount);

void SetHeapBlockNoCheck(void \* address, size\_t pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex);

bool SetHeapBlock(void \* address, size\_t pageCount, HeapBlock \* heapBlock, HeapBlock::HeapBlockType blockType, byte bucketIndex);

void ClearHeapBlock(void \* address, size\_t pageCount);

HeapBlock \* GetHeapBlock(void \* address);

HeapBlockMap32::PageMarkBitVector \* GetPageMarkBitVector(void \* address);

template <size\_t BitCount>

BVStatic<BitCount>\* GetMarkBitVectorForPages(void \* address);

uint GetMarkCount(void\* address, uint pageCount);

template <bool interlocked>

void Mark(void \* candidate, MarkContext \* markContext);

template <bool interlocked>

void MarkInterior(void \* candidate, MarkContext \* markContext);

bool IsMarked(void \* address) const;

void SetMark(void \* address);

bool TestAndSetMark(void \* address);

void ResetMarks();

#ifdef CONCURRENT\_GC\_ENABLED

void ResetWriteWatch(Recycler \* recycler);

uint Rescan(Recycler \* recycler, bool resetWriteWatch);

void MakeAllPagesReadOnly(Recycler\* recycler);

void MakeAllPagesReadWrite(Recycler\* recycler);

#endif

void Cleanup(bool concurrentFindImplicitRoot);

bool OOMRescan(Recycler \* recycler);

#ifdef RECYCLER\_STRESS

void InduceFalsePositives(Recycler \* recycler);

#endif

#ifdef RECYCLER\_VERIFY\_MARK

bool IsAddressInNewChunk(void \* address);

#endif

private:

friend class HeapBlockMap32;

#ifdef JD\_PRIVATE

friend class HeapBlockHelper;

#endif

struct Node

{

Node(\_\_in char \* startAddress) : map(startAddress) { }

uint nodeIndex;

Node \* next;

HeapBlockMap32 map;

};

static const uint PagesPer4GB = 1 << 20; // = 1M, assume page size = 4K

static uint GetNodeIndex(void \* address)

{

return GetNodeIndex((ULONG64)address);

}

static uint GetNodeIndex(ULONG64 address)

{

return (uint)((ULONG64)address >> 32);

}

Node \* FindOrInsertNode(void \* address);

Node \* FindNode(void \* address) const;

template <class Fn>

void ForEachNodeInAddressRange(void \* address, size\_t pageCount, Fn fn);

Node \* list;

public:

#if DBG

ushort GetPageMarkCount(void \* address) const;

void SetPageMarkCount(void \* address, ushort markCount);

template <uint BitVectorCount>

void VerifyMarkCountForPages(void \* address, uint pageCount);

#endif

#if !defined(JD\_PRIVATE)

static char \* GetNodeStartAddress(void \* address)

{

return (char \*)(((size\_t)address) & ~(HeapBlockMap32::TotalSize - 1));

}

#endif

};

typedef HeapBlockMap64 HeapBlockMap;

#else

typedef HeapBlockMap32 HeapBlockMap;

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template <bool interlocked>

\_\_inline

bool

HeapBlockMap32::MarkInternal(L2MapChunk \* chunk, void \* candidate)

{

uint bitIndex = chunk->GetMarkBitIndex(candidate);

if (interlocked)

{

// Use an interlocked BTS instruction to ensure atomicity.

// Since this is expensive, do a non-interlocked test first.

// Mark bits never go from set to clear during marking, so if we find the bit is already set, we're done.

if (chunk->markBits.TestIntrinsic(bitIndex))

{

// Already marked; no further processing needed

return true;

}

if (chunk->markBits.TestAndSetInterlocked(bitIndex))

{

// Already marked; no further processing needed

return true;

}

}

else

{

if (chunk->markBits.TestAndSet(bitIndex))

{

// Already marked; no further processing needed

return true;

}

}

#if DBG

InterlockedIncrement16((short \*)&chunk->pageMarkCount[GetLevel2Id(candidate)]);

#endif

return false;

}

//

// Mark a particular object

// If the object is already marked, or if it's invalid, return true

// (indicating there's no further processing to be done for this object)

// If the object is newly marked, then the out param heapBlock is written to, and false is returned

//

template <bool interlocked>

\_\_inline

void

HeapBlockMap32::Mark(void \* candidate, MarkContext \* markContext)

{

uint id1 = GetLevel1Id(candidate);

L2MapChunk \* chunk = map[id1];

if (chunk == nullptr)

{

// False refernce; no further processing needed.

return;

}

if (MarkInternal<interlocked>(chunk, candidate))

{

return;

}

uint id2 = GetLevel2Id(candidate);

HeapBlock::HeapBlockType blockType = chunk->blockInfo[id2].blockType;

Assert(blockType == HeapBlock::HeapBlockType::FreeBlockType || chunk->map[id2]->GetHeapBlockType() == blockType);

// Switch on the HeapBlockType to determine how to process the newly marked object.

switch (blockType)

{

case HeapBlock::HeapBlockType::FreeBlockType:

// False reference. Do nothing.

break;

case HeapBlock::HeapBlockType::SmallLeafBlockType:

case HeapBlock::HeapBlockType::MediumLeafBlockType:

// Leaf blocks don't need to be scanned. Do nothing.

break;

case HeapBlock::HeapBlockType::SmallNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::SmallNormalBlockWithBarrierType:

#endif

{

byte bucketIndex = chunk->blockInfo[id2].bucketIndex;

// See if it's an invalid offset using the invalid bit vector and if so, do nothing.

if (!HeapInfo::GetInvalidBitVectorForBucket<SmallAllocationBlockAttributes>(bucketIndex)->Test(SmallHeapBlock::GetAddressBitIndex(candidate)))

{

uint objectSize = HeapInfo::GetObjectSizeForBucketIndex<SmallAllocationBlockAttributes>(bucketIndex);

if (!markContext->AddMarkedObject(candidate, objectSize))

{

// Failed to mark due to OOM.

((SmallHeapBlock \*)chunk->map[id2])->SetNeedOOMRescan(markContext->GetRecycler());

}

}

}

break;

case HeapBlock::HeapBlockType::MediumNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::MediumNormalBlockWithBarrierType:

#endif

{

byte bucketIndex = chunk->blockInfo[id2].bucketIndex;

// See if it's an invalid offset using the invalid bit vector and if so, do nothing.

if (!HeapInfo::GetInvalidBitVectorForBucket<MediumAllocationBlockAttributes>(bucketIndex)->Test(MediumHeapBlock::GetAddressBitIndex(candidate)))

{

uint objectSize = HeapInfo::GetObjectSizeForBucketIndex<MediumAllocationBlockAttributes>(bucketIndex);

if (!markContext->AddMarkedObject(candidate, objectSize))

{

// Failed to mark due to OOM.

((MediumHeapBlock \*)chunk->map[id2])->SetNeedOOMRescan(markContext->GetRecycler());

}

}

}

break;

case HeapBlock::HeapBlockType::SmallFinalizableBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::SmallFinalizableBlockWithBarrierType:

#endif

((SmallFinalizableHeapBlock\*)chunk->map[id2])->ProcessMarkedObject(candidate, markContext);

break;

case HeapBlock::HeapBlockType::MediumFinalizableBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::MediumFinalizableBlockWithBarrierType:

#endif

((MediumFinalizableHeapBlock\*)chunk->map[id2])->ProcessMarkedObject(candidate, markContext);

break;

case HeapBlock::HeapBlockType::LargeBlockType:

((LargeHeapBlock\*)chunk->map[id2])->Mark(candidate, markContext);

break;

#if DBG

default:

AssertMsg(false, "what's the new heap block type?");

#endif

}

}

template <bool interlocked, bool updateChunk>

\_\_inline

bool

HeapBlockMap32::MarkInteriorInternal(MarkContext \* markContext, L2MapChunk \*& chunk, void \* originalCandidate, void \* realCandidate)

{

if (originalCandidate == realCandidate)

{

// The initial mark performed was correct (we had a base pointer)

return false;

}

if (realCandidate == nullptr)

{

// We had an invalid interior pointer, so we bail out

return true;

}

if (updateChunk)

{

#if defined(\_M\_X64\_OR\_ARM64)

if (HeapBlockMap64::GetNodeIndex(originalCandidate) != HeapBlockMap64::GetNodeIndex(realCandidate))

{

// We crossed a node boundary (very rare) so we should just re-start from the real candidate.

// In this case we are no longer marking an interior reference.

markContext->GetRecycler()->heapBlockMap.Mark<interlocked>(realCandidate, markContext);

// This mark code therefore has nothing to do (it has already happened).

return true;

}

#endif

// Update the chunk as the interior pointer may cross an L2 boundary (e.g., a large object)

chunk = map[GetLevel1Id(realCandidate)];

}

// Perform the actual mark for the interior pointer

return MarkInternal<interlocked>(chunk, realCandidate);

}

template <bool interlocked>

\_\_inline

void

HeapBlockMap32::MarkInterior(void \* candidate, MarkContext \* markContext)

{

// Align the candidate to object granularity

candidate = reinterpret\_cast<void\*>(reinterpret\_cast<size\_t>(candidate) & ~HeapInfo::ObjectAlignmentMask);

uint id1 = GetLevel1Id(candidate);

L2MapChunk \* chunk = map[id1];

if (chunk == nullptr)

{

// False reference; no further processing needed.

return;

}

if (MarkInternal<interlocked>(chunk, candidate))

{

// Already marked (mark internal-then-actual first)

return;

}

uint id2 = GetLevel2Id(candidate);

HeapBlock::HeapBlockType blockType = chunk->blockInfo[id2].blockType;

// Switch on the HeapBlockType to determine how to map interior->base and process object.

switch (blockType)

{

case HeapBlock::HeapBlockType::FreeBlockType:

// False reference. Do nothing.

break;

case HeapBlock::HeapBlockType::SmallLeafBlockType:

case HeapBlock::HeapBlockType::MediumLeafBlockType:

// Leaf blocks don't need to be scanned. Do nothing.

break;

case HeapBlock::HeapBlockType::SmallNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::SmallNormalBlockWithBarrierType:

#endif

{

byte bucketIndex = chunk->blockInfo[id2].bucketIndex;

uint objectSize = HeapInfo::GetObjectSizeForBucketIndex<SmallAllocationBlockAttributes>(bucketIndex);

void \* realCandidate = SmallHeapBlock::GetRealAddressFromInterior(candidate, objectSize, bucketIndex);

if (MarkInteriorInternal<interlocked, false>(markContext, chunk, candidate, realCandidate))

{

break;

}

if (!markContext->AddMarkedObject(realCandidate, objectSize))

{

// Failed to mark due to OOM.

((SmallHeapBlock \*)chunk->map[id2])->SetNeedOOMRescan(markContext->GetRecycler());

}

}

break;

case HeapBlock::HeapBlockType::MediumNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::MediumNormalBlockWithBarrierType:

#endif

{

byte bucketIndex = chunk->blockInfo[id2].bucketIndex;

uint objectSize = HeapInfo::GetObjectSizeForBucketIndex<MediumAllocationBlockAttributes>(bucketIndex);

void \* realCandidate = MediumHeapBlock::GetRealAddressFromInterior(candidate, objectSize, bucketIndex);

if (MarkInteriorInternal<interlocked, false>(markContext, chunk, candidate, realCandidate))

{

break;

}

if (!markContext->AddMarkedObject(realCandidate, objectSize))

{

// Failed to mark due to OOM.

((MediumHeapBlock \*)chunk->map[id2])->SetNeedOOMRescan(markContext->GetRecycler());

}

}

break;

case HeapBlock::HeapBlockType::SmallFinalizableBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::SmallFinalizableBlockWithBarrierType:

#endif

{

void \* realCandidate = ((SmallFinalizableHeapBlock\*)chunk->map[id2])->GetRealAddressFromInterior(candidate);

if (MarkInteriorInternal<interlocked, false>(markContext, chunk, candidate, realCandidate))

{

break;

}

((SmallFinalizableHeapBlock\*)chunk->map[id2])->ProcessMarkedObject(realCandidate, markContext);

}

break;

case HeapBlock::HeapBlockType::MediumFinalizableBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::MediumFinalizableBlockWithBarrierType:

#endif

{

void \* realCandidate = ((MediumFinalizableHeapBlock\*)chunk->map[id2])->GetRealAddressFromInterior(candidate);

if (MarkInteriorInternal<interlocked, false>(markContext, chunk, candidate, realCandidate))

{

break;

}

((MediumFinalizableHeapBlock\*)chunk->map[id2])->ProcessMarkedObject(realCandidate, markContext);

}

break;

case HeapBlock::HeapBlockType::LargeBlockType:

{

void \* realCandidate = ((LargeHeapBlock\*)chunk->map[id2])->GetRealAddressFromInterior(candidate);

if (MarkInteriorInternal<interlocked, true>(markContext, chunk, candidate, realCandidate))

{

break;

}

((LargeHeapBlock\*)chunk->map[GetLevel2Id(realCandidate)])->Mark(realCandidate, markContext);

}

break;

#if DBG

default:

AssertMsg(false, "what's the new heap block type?");

#endif

}

}

#if defined(\_M\_X64\_OR\_ARM64)

//

// 64-bit Mark

// See HeapBlockMap32::Mark for explanation of return values

//

template <bool interlocked>

\_\_inline

void

HeapBlockMap64::Mark(void \* candidate, MarkContext \* markContext)

{

uint index = GetNodeIndex(candidate);

Node \* node = list;

while (node != nullptr)

{

if (node->nodeIndex == index)

{

// Found the correct Node.

// Process the mark and return.

node->map.Mark<interlocked>(candidate, markContext);

return;

}

node = node->next;

}

// No Node found; must be an invalid reference. Do nothing.

}

template <bool interlocked>

\_\_inline

void

HeapBlockMap64::MarkInterior(void \* candidate, MarkContext \* markContext)

{

uint index = GetNodeIndex(candidate);

Node \* node = list;

while (node != nullptr)

{

if (node->nodeIndex == index)

{

// Found the correct Node.

// Process the mark and return.

node->map.MarkInterior<interlocked>(candidate, markContext);

return;

}

node = node->next;

}

// No Node found; must be an invalid reference. Do nothing.

}

#endif // defined(\_M\_X64\_OR\_ARM64)

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

HeapBucket::HeapBucket() :

heapInfo(nullptr),

sizeCat(0)

{

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

heapBlockCount = 0;

newHeapBlockCount = 0;

emptyHeapBlockCount = 0;

#endif

#ifdef RECYCLER\_PAGE\_HEAP

isPageHeapEnabled = false;

#endif

}

uint

HeapBucket::GetBucketIndex() const

{

return HeapInfo::GetBucketIndex(this->sizeCat);

}

uint

HeapBucket::GetMediumBucketIndex() const

{

return HeapInfo::GetMediumBucketIndex(this->sizeCat);

}

EXPLICIT\_INSTANTIATE\_WITH\_SMALL\_HEAP\_BLOCK\_TYPE(HeapBucketT);

template <typename TBlockType>

HeapBucketT<TBlockType>::HeapBucketT() :

nextAllocableBlockHead(nullptr),

emptyBlockList(nullptr),

fullBlockList(nullptr),

heapBlockList(nullptr),

explicitFreeList(nullptr),

lastExplicitFreeListAllocator(nullptr)

{

#ifdef RECYCLER\_PAGE\_HEAP

explicitFreeLockBlockList = nullptr;

#endif

#if DBG

isAllocationStopped = false;

#endif

}

template <typename TBlockType>

HeapBucketT<TBlockType>::~HeapBucketT()

{

DeleteHeapBlockList(this->heapBlockList);

DeleteHeapBlockList(this->fullBlockList);

Assert(this->heapBlockCount + this->newHeapBlockCount == 0);

RECYCLER\_SLOW\_CHECK(Assert(this->emptyHeapBlockCount == HeapBlockList::Count(this->emptyBlockList)));

DeleteEmptyHeapBlockList(this->emptyBlockList);

Assert(this->heapBlockCount + this->newHeapBlockCount + this->emptyHeapBlockCount == 0);

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::DeleteHeapBlockList(TBlockType \* list, Recycler \* recycler)

{

HeapBlockList::ForEachEditing(list, [recycler](TBlockType \* heapBlock)

{

#if DBG

heapBlock->ReleasePagesShutdown(recycler);

#endif

TBlockType::Delete(heapBlock);

});

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::DeleteEmptyHeapBlockList(TBlockType \* list)

{

HeapBlockList::ForEachEditing(list, [](TBlockType \* heapBlock)

{

TBlockType::Delete(heapBlock);

});

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::DeleteHeapBlockList(TBlockType \* list)

{

DeleteHeapBlockList(list, this->heapInfo->recycler);

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::Initialize(HeapInfo \* heapInfo, uint sizeCat)

{

this->heapInfo = heapInfo;

#ifdef RECYCLER\_PAGE\_HEAP

this->isPageHeapEnabled = heapInfo->IsPageHeapEnabledForBlock<TBlockType::HeapBlockAttributes>(sizeCat);

#endif

this->sizeCat = sizeCat;

allocatorHead.Initialize();

#ifdef PROFILE\_RECYCLER\_ALLOC

allocatorHead.bucket = this;

#endif

this->lastExplicitFreeListAllocator = &allocatorHead;

}

template <typename TBlockType>

template <class Fn>

void

HeapBucketT<TBlockType>::ForEachAllocator(Fn fn)

{

TBlockAllocatorType \* current = &allocatorHead;

do

{

fn(current);

current = current->GetNext();

}

while (current != &allocatorHead);

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::UpdateAllocators()

{

ForEachAllocator([](TBlockAllocatorType \* allocator) { allocator->UpdateHeapBlock(); });

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::ClearAllocators()

{

ForEachAllocator([](TBlockAllocatorType \* allocator) { ClearAllocator(allocator); });

#ifdef RECYCLER\_PAGE\_HEAP

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

FreeObject\* freeObject = this->explicitFreeList;

while (freeObject)

{

HeapBlock\* heapBlock = this->GetRecycler()->FindHeapBlock((void\*)freeObject);

Assert(heapBlock != nullptr);

Assert(!heapBlock->IsLargeHeapBlock());

TBlockType\* smallBlock = (TBlockType\*)heapBlock;

smallBlock->ClearExplicitFreeBitForObject((void\*)freeObject);

freeObject = freeObject->GetNext();

}

#endif

this->explicitFreeList = nullptr;

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::PrepareSweep()

{

// CONCURRENT-TODO: Technically, We don't really need to invalidate allocators here,

// but currently invalidating may update the unallocateCount which is

// used to calculate the partial heuristics, so it needs to be done

// before sweep. When the partial heuristic changes, we can remove this

// (And remove rescan from leaf bucket, so this function doesn't need to exist)

ClearAllocators();

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::AddAllocator(TBlockAllocatorType \* allocator)

{

Assert(allocator != &this->allocatorHead);

allocator->Initialize();

allocator->next = this->allocatorHead.next;

allocator->prev = &this->allocatorHead;

allocator->next->prev = allocator;

this->allocatorHead.next = allocator;

#ifdef PROFILE\_RECYCLER\_ALLOC

allocator->bucket = this;

#endif

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::RemoveAllocator(TBlockAllocatorType \* allocator)

{

Assert(allocator != &this->allocatorHead);

ClearAllocator(allocator);

allocator->next->prev = allocator->prev;

allocator->prev->next = allocator->next;

if (allocator == this->lastExplicitFreeListAllocator)

{

this->lastExplicitFreeListAllocator = &allocatorHead;

}

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::ClearAllocator(TBlockAllocatorType \* allocator)

{

allocator->Clear();

}

template <typename TBlockType>

bool

HeapBucketT<TBlockType>::IntegrateBlock(char \* blockAddress, PageSegment \* segment, Recycler \* recycler)

{

// Add a new heap block

TBlockType \* heapBlock = GetUnusedHeapBlock();

if (heapBlock == nullptr)

{

return false;

}

// TODO: Consider supporting guard pages for this codepath

#ifdef RECYCLER\_PAGE\_HEAP

heapBlock->ClearPageHeap();

#endif

if (!heapBlock->SetPage<false>(blockAddress, segment, recycler))

{

FreeHeapBlock(heapBlock);

return false;

}

heapBlock->SetNextBlock(this->fullBlockList);

this->fullBlockList = heapBlock;

RECYCLER\_SLOW\_CHECK(this->heapBlockCount++);

this->heapInfo->uncollectedAllocBytes += heapBlock->GetAndClearLastFreeCount() \* heapBlock->GetObjectSize();

RecyclerMemoryTracking::ReportAllocation(recycler, blockAddress, heapBlock->GetObjectSize() \* heapBlock->GetObjectCount());

RECYCLER\_PERF\_COUNTER\_ADD(LiveObject,heapBlock->GetObjectCount());

RECYCLER\_PERF\_COUNTER\_ADD(LiveObjectSize, heapBlock->GetObjectSize() \* heapBlock->GetObjectCount());

if (heapBlock->IsLargeHeapBlock())

{

RECYCLER\_PERF\_COUNTER\_ADD(LargeHeapBlockLiveObject,heapBlock->GetObjectCount());

RECYCLER\_PERF\_COUNTER\_ADD(LargeHeapBlockLiveObjectSize, heapBlock->GetObjectSize() \* heapBlock->GetObjectCount());

}

else

{

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockLiveObject,heapBlock->GetObjectCount());

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockLiveObjectSize, heapBlock->GetObjectSize() \* heapBlock->GetObjectCount());

}

#if DBG

heapBlock->SetIsIntegratedBlock();

#endif

return true;

}

#if DBG

template <typename TBlockType>

bool

HeapBucketT<TBlockType>::AllocatorsAreEmpty() const

{

TBlockAllocatorType const \* current = &allocatorHead;

do

{

if (current->GetHeapBlock() != nullptr || current->GetExplicitFreeList() != nullptr)

{

return false;

}

current = current->GetNext();

}

while (current != &allocatorHead);

return true;

}

template <typename TBlockType>

bool

HeapBucketT<TBlockType>::HasPendingDisposeHeapBlocks() const

{

#ifdef RECYCLER\_WRITE\_BARRIER

return (IsFinalizableBucket || IsFinalizableWriteBarrierBucket) && ((SmallFinalizableHeapBucketT<TBlockType::HeapBlockAttributes> \*)this)->pendingDisposeList != nullptr;

#else

return IsFinalizableBucket && ((SmallFinalizableHeapBucketT<TBlockType::HeapBlockAttributes> \*)this)->pendingDisposeList != nullptr;

#endif

}

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

template <typename TBlockType>

size\_t

HeapBucketT<TBlockType>::GetNonEmptyHeapBlockCount(bool checkCount) const

{

size\_t currentHeapBlockCount = HeapBlockList::Count(fullBlockList);

currentHeapBlockCount += HeapBlockList::Count(heapBlockList);

#ifdef CONCURRENT\_GC\_ENABLED

// Recycler can be null if we have OOM in the ctor

if (this->GetRecycler() && this->GetRecycler()->recyclerSweep != nullptr)

{

currentHeapBlockCount += this->GetRecycler()->recyclerSweep->GetHeapBlockCount(this);

}

#endif

RECYCLER\_SLOW\_CHECK(Assert(!checkCount || heapBlockCount == currentHeapBlockCount));

return currentHeapBlockCount;

}

template <typename TBlockType>

size\_t

HeapBucketT<TBlockType>::GetEmptyHeapBlockCount() const

{

size\_t count = HeapBlockList::Count(this->emptyBlockList);

RECYCLER\_SLOW\_CHECK(Assert(count == this->emptyHeapBlockCount));

return count;

}

#endif

template <typename TBlockType>

char \*

HeapBucketT<TBlockType>::TryAlloc(Recycler \* recycler, TBlockAllocatorType \* allocator, size\_t sizeCat, ObjectInfoBits attributes)

{

AUTO\_NO\_EXCEPTION\_REGION;

Assert((attributes & InternalObjectInfoBitMask) == attributes);

ClearAllocator(allocator);

TBlockType \* heapBlock = this->nextAllocableBlockHead;

if (heapBlock != nullptr)

{

Assert(!this->IsAllocationStopped());

this->nextAllocableBlockHead = heapBlock->GetNextBlock();

allocator->Set(heapBlock);

}

else if (this->explicitFreeList != nullptr)

{

allocator->SetExplicitFreeList(this->explicitFreeList);

this->lastExplicitFreeListAllocator = allocator;

this->explicitFreeList = nullptr;

}

else

{

return nullptr;

}

// We just found a block we can allocate on

char \* memBlock = allocator->SlowAlloc<false /\* disallow fault injection \*/>(recycler, sizeCat, attributes);

Assert(memBlock != nullptr);

return memBlock;

}

template <typename TBlockType>

char \*

HeapBucketT<TBlockType>::TryAllocFromNewHeapBlock(Recycler \* recycler, TBlockAllocatorType \* allocator, size\_t sizeCat, ObjectInfoBits attributes)

{

AUTO\_NO\_EXCEPTION\_REGION;

Assert((attributes & InternalObjectInfoBitMask) == attributes);

#ifdef RECYCLER\_PAGE\_HEAP

if (IsPageHeapEnabled())

{

return this->PageHeapAlloc(recycler, sizeCat, attributes, this->heapInfo->pageHeapMode, true);

}

#endif

TBlockType \* heapBlock = CreateHeapBlock<false>(recycler);

if (heapBlock == nullptr)

{

return nullptr;

}

// new heap block added, allocate from that.

allocator->SetNew(heapBlock);

// We just created a block we can allocate on

char \* memBlock = allocator->SlowAlloc<false /\* disallow fault injection \*/>(recycler, sizeCat, attributes);

Assert(memBlock != nullptr || IS\_FAULTINJECT\_NO\_THROW\_ON);

return memBlock;

}

#ifdef RECYCLER\_PAGE\_HEAP

template <typename TBlockType>

char \*

HeapBucketT<TBlockType>::PageHeapAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes, PageHeapMode mode, bool nothrow)

{

AllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"In PageHeapAlloc [Size: 0x%x, Attributes: 0x%x]\n", sizeCat, attributes);

Assert(sizeCat == this->sizeCat);

char \* memBlock = nullptr;

TBlockAllocatorType\* allocator = &this->allocatorHead;

memBlock = allocator->PageHeapAlloc(recycler, sizeCat, attributes, mode);

if (memBlock == nullptr)

{

TBlockType\* heapBlock = nullptr;

allocator->Clear();

{

AUTO\_NO\_EXCEPTION\_REGION;

heapBlock = CreateHeapBlock<true>(recycler);

}

if (heapBlock != nullptr)

{

// new heap block added, allocate from that.

allocator->SetNew(heapBlock);

memBlock = allocator->PageHeapAlloc(recycler, sizeCat, attributes, mode);

Assert(memBlock != nullptr || IS\_FAULTINJECT\_NO\_THROW\_ON);

}

if (memBlock == nullptr)

{

recycler->CollectNow<CollectNowForceInThread>();

memBlock = allocator->PageHeapAlloc(recycler, sizeCat, attributes, mode);

if (memBlock == nullptr)

{

// Although we collected in thread, PostCollectCallback may

// allocate memory and populated the allocator again, let's clear it again

allocator->Clear();

TBlockType\* heapBlock = nullptr;

{

AUTO\_NO\_EXCEPTION\_REGION;

heapBlock = CreateHeapBlock<true>(recycler);

}

if (heapBlock != nullptr)

{

// new heap block added, allocate from that.

allocator->SetNew(heapBlock);

memBlock = allocator->PageHeapAlloc(recycler, sizeCat, attributes, mode);

Assert(memBlock != nullptr || IS\_FAULTINJECT\_NO\_THROW\_ON);

}

if (memBlock == nullptr)

{

// If nothrow is false, that means throwing is allowed

// Since we have no more memory to allocate here, throw an OOM exception

// If nothrow is true, then simply return null

if (nothrow == false)

{

recycler->OutOfMemory();

}

else

{

return nullptr;

}

}

}

}

}

Assert(memBlock != nullptr);

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

if ((attributes & ObjectInfoBits::LeafBit) == 0

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_THREAD\_PAGE

&& ((attributes & ObjectInfoBits::WithBarrierBit) == 0)

#endif

)

{

// Skip the first and the last pointer objects- the first may have next pointer for the free list

// the last might have the old size of the object if this was allocated from an explicit free list

recycler->VerifyZeroFill(memBlock + sizeof(FreeObject), sizeCat - (2 \* sizeof(FreeObject)));

}

#endif

return memBlock;

}

#endif

template <typename TBlockType>

char \*

HeapBucketT<TBlockType>::SnailAlloc(Recycler \* recycler, TBlockAllocatorType \* allocator, size\_t sizeCat, ObjectInfoBits attributes, bool nothrow)

{

AllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"In SnailAlloc [Size: 0x%x, Attributes: 0x%x]\n", sizeCat, attributes);

Assert(sizeCat == this->sizeCat);

Assert((attributes & InternalObjectInfoBitMask) == attributes);

char \* memBlock = this->TryAlloc(recycler, allocator, sizeCat, attributes);

if (memBlock != nullptr)

{

return memBlock;

}

// No free memory, try to collect with allocated bytes and time heuristic, and concurrently

BOOL collected = recycler->disableCollectOnAllocationHeuristics ? recycler->FinishConcurrent<FinishConcurrentOnAllocation>() :

recycler->CollectNow<CollectOnAllocation>();

AllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"TryAlloc failed, forced collection on allocation [Collected: %d]\n", collected);

if (!collected)

{

// We didn't collect, try to add a new heap block

memBlock = TryAllocFromNewHeapBlock(recycler, allocator, sizeCat, attributes);

if (memBlock != nullptr)

{

return memBlock;

}

// Can't even allocate a new block, we need force a collection and

//allocate some free memory, add a new heap block again, or throw out of memory

AllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"TryAllocFromNewHeapBlock failed, forcing in-thread collection\n");

recycler->CollectNow<CollectNowForceInThread>();

}

// Collection might trigger finalizer, which might allocate memory. So the allocator

// might have a heap block already, try to allocate from that first

memBlock = allocator->SlowAlloc<true /\* allow fault injection \*/>(recycler, sizeCat, attributes);

if (memBlock != nullptr)

{

return memBlock;

}

AllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"SlowAlloc failed\n");

// do the allocation

memBlock = this->TryAlloc(recycler, allocator, sizeCat, attributes);

if (memBlock != nullptr)

{

return memBlock;

}

AllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"TryAlloc failed\n");

// add a heap block if there are no preallocated memory left.

memBlock = TryAllocFromNewHeapBlock(recycler, allocator, sizeCat, attributes);

if (memBlock != nullptr)

{

return memBlock;

}

AllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"TryAllocFromNewHeapBlock failed- triggering OOM handler");

if (nothrow == false)

{

// Can't add a heap block, we are out of memory

// Since we're allowed to throw, throw right here

recycler->OutOfMemory();

}

return nullptr;

}

template <typename TBlockType>

TBlockType\*

HeapBucketT<TBlockType>::GetUnusedHeapBlock()

{

// Add a new heap block

TBlockType \* heapBlock = emptyBlockList;

if (heapBlock == nullptr)

{

// We couldn't find a reusable heap block

heapBlock = TBlockType::New(this);

RECYCLER\_SLOW\_CHECK(Assert(this->emptyHeapBlockCount == 0));

}

else

{

emptyBlockList = heapBlock->GetNextBlock();

RECYCLER\_SLOW\_CHECK(this->emptyHeapBlockCount--);

}

return heapBlock;

}

template <typename TBlockType>

template<bool pageheap>

TBlockType \*

HeapBucketT<TBlockType>::CreateHeapBlock(Recycler \* recycler)

{

FAULTINJECT\_MEMORY\_NOTHROW(L"HeapBlock", sizeof(TBlockType));

// Add a new heap block

TBlockType \* heapBlock = GetUnusedHeapBlock();

if (heapBlock == nullptr)

{

return nullptr;

}

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap)

{

heapBlock->EnablePageHeap();

}

#endif

if (!heapBlock->ReassignPages<pageheap>(recycler))

{

FreeHeapBlock(heapBlock);

return nullptr;

}

// Add it to head of heap block list so we will keep track of the block

recycler->autoHeap.AppendNewHeapBlock(heapBlock, this);

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

#ifdef CONCURRENT\_GC\_ENABLED

::InterlockedIncrement(&this->newHeapBlockCount);

#else

this->heapBlockCount++;

#endif

#endif

return heapBlock;

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::FreeHeapBlock(TBlockType \* heapBlock)

{

heapBlock->Reset();

heapBlock->SetNextBlock(emptyBlockList);

emptyBlockList = heapBlock;

RECYCLER\_SLOW\_CHECK(this->emptyHeapBlockCount++);

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::ResetMarks(ResetMarkFlags flags)

{

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount((flags & ResetMarkFlags\_Background) != 0));

#ifndef CONCURRENT\_GC\_ENABLED

Assert((flags & ResetMarkFlags\_Background) == 0);

#endif

if ((flags & ResetMarkFlags\_Background) == 0)

{

// The is equivalent to the ClearAllocators in Rescan

// But since we are not doing concurrent, we need to do it here.

ClearAllocators();

}

// Note, mark bits are now cleared in HeapBlockMap32::ResetMarks, so we don't need to clear them here.

if ((flags & ResetMarkFlags\_ScanImplicitRoot) != 0)

{

HeapBlockList::ForEach(fullBlockList, [flags](TBlockType \* heapBlock)

{

heapBlock->MarkImplicitRoots();

Assert(!heapBlock->HasFreeObject());

});

HeapBlockList::ForEach(heapBlockList, [flags](TBlockType \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

}

#if DBG

if ((flags & ResetMarkFlags\_Background) == 0)

{

// Verify that if you are in the heapBlockList, before the nextAllocableBlockHead, we have fully allocated from

// the block already, except if we have cleared from the allocator, or it is still in the allocator

HeapBlockList::ForEach(heapBlockList, nextAllocableBlockHead, [](TBlockType \* heapBlock)

{

// If the heap block is in the allocator, then the heap block may or may not have free object still

// So we can't assert. Otherwise, we have free object iff we were cleared from allocator

Assert(heapBlock->IsInAllocator() || heapBlock->HasFreeObject() == heapBlock->IsClearedFromAllocator());

});

// We should still have allocable free object after nextAllocableBlockHead

HeapBlockList::ForEach(nextAllocableBlockHead, [](TBlockType \* heapBlock)

{

Assert(heapBlock->HasFreeObject());

});

}

#endif

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::ScanNewImplicitRoots(Recycler \* recycler)

{

HeapBlockList::ForEach(fullBlockList, [recycler](TBlockType \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

HeapBlockList::ForEach(heapBlockList, [recycler](TBlockType \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

}

#if DBG

template <typename TBlockType>

void

HeapBucketT<TBlockType>::VerifyBlockConsistencyInList(TBlockType \* heapBlock, RecyclerVerifyListConsistencyData& recyclerSweep)

{

bool\* expectFull = nullptr;

bool\* expectDispose = nullptr;

HeapBlock\* nextAllocableBlockHead = nullptr;

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

expectFull = &recyclerSweep.smallBlockVerifyListConsistencyData.expectFull;

expectDispose = &recyclerSweep.smallBlockVerifyListConsistencyData.expectDispose;

nextAllocableBlockHead = recyclerSweep.smallBlockVerifyListConsistencyData.nextAllocableBlockHead;

}

else if (TBlockType::HeapBlockAttributes::IsMediumBlock)

{

expectFull = &recyclerSweep.mediumBlockVerifyListConsistencyData.expectFull;

expectDispose = &recyclerSweep.mediumBlockVerifyListConsistencyData.expectDispose;

nextAllocableBlockHead = recyclerSweep.mediumBlockVerifyListConsistencyData.nextAllocableBlockHead;

}

else

{

Assert(false);

}

if (heapBlock == nextAllocableBlockHead)

{

(\*expectFull) = false;

}

if (heapBlock->IsClearedFromAllocator())

{

Assert(\*expectFull && !\*expectDispose);

Assert(heapBlock->HasFreeObject());

Assert(!heapBlock->HasAnyDisposeObjects());

}

else if (\*expectDispose)

{

Assert(heapBlock->IsAnyFinalizableBlock() && heapBlock->AsFinalizableBlock<TBlockType::HeapBlockAttributes>()->IsPendingDispose());

Assert(heapBlock->HasAnyDisposeObjects());

}

else

{

Assert(!heapBlock->HasAnyDisposeObjects());

// ExpectFull is a bit of a misnomer if the list in question is the heap block list. It's there to check

// of the heap block in question is before the nextAllocableBlockHead or not. This is to ensure that

// blocks before nextAllocableBlockHead that are not being bump allocated from must be considered "full".

// However, the exception is if this is the only heap block in this bucket, in which case nextAllocableBlockHead

// would be null

Assert(\*expectFull == (!heapBlock->HasFreeObject() || heapBlock->IsInAllocator()) || nextAllocableBlockHead == nullptr);

}

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::VerifyBlockConsistencyInList(TBlockType \* heapBlock, RecyclerVerifyListConsistencyData const& recyclerSweep, SweepState state)

{

bool expectFull = false;

bool expectDispose = false;

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

expectFull = recyclerSweep.smallBlockVerifyListConsistencyData.expectFull;

expectDispose = recyclerSweep.smallBlockVerifyListConsistencyData.expectDispose;

}

else if (TBlockType::HeapBlockAttributes::IsMediumBlock)

{

expectFull = recyclerSweep.mediumBlockVerifyListConsistencyData.expectFull;

expectDispose = recyclerSweep.mediumBlockVerifyListConsistencyData.expectDispose;

}

else

{

Assert(false);

}

if (heapBlock->IsClearedFromAllocator())

{

// this function is called during sweep and we are recreating the heap block list

// which would make all the block to be in it's rightful place

heapBlock->SetIsClearedFromAllocator(false);

Assert(SweepStateFull != state);

}

else

{

// You can still be full only if you are full before.

Assert(expectFull || SweepStateFull != state);

}

// If you were pending dispose before, you can only be pending dispose after

Assert(!expectDispose || SweepStatePendingDispose == state);

}

#endif // DBG

#ifdef PARTIAL\_GC\_ENABLED

template <typename TBlockType>

bool

HeapBucketT<TBlockType>::DoQueuePendingSweep(Recycler \* recycler)

{

return IsNormalBucket && recycler->inPartialCollectMode;

}

template <typename TBlockType>

bool

HeapBucketT<TBlockType>::DoPartialReuseSweep(Recycler \* recycler)

{

// With leaf, we don't need to do a partial sweep

// WriteBarrier-TODO: We shouldn't need to do this for write barrier heap buckets either

return !IsLeafBucket && recycler->inPartialCollectMode;

}

#endif

template void HeapBucketT<SmallLeafHeapBlock>::SweepHeapBlockList<true>(RecyclerSweep&, SmallLeafHeapBlock \*, bool);

template void HeapBucketT<SmallLeafHeapBlock>::SweepHeapBlockList<false>(RecyclerSweep&, SmallLeafHeapBlock \*, bool);

template void HeapBucketT<SmallNormalHeapBlock>::SweepHeapBlockList<true>(RecyclerSweep&, SmallNormalHeapBlock \*, bool);

template void HeapBucketT<SmallNormalHeapBlock>::SweepHeapBlockList<false>(RecyclerSweep&, SmallNormalHeapBlock \*, bool);

template void HeapBucketT<SmallFinalizableHeapBlock>::SweepHeapBlockList<true>(RecyclerSweep&, SmallFinalizableHeapBlock \*, bool);

template void HeapBucketT<SmallFinalizableHeapBlock>::SweepHeapBlockList<false>(RecyclerSweep&, SmallFinalizableHeapBlock \*, bool);

#ifdef RECYCLER\_WRITE\_BARRIER

template void HeapBucketT<SmallNormalWithBarrierHeapBlock>::SweepHeapBlockList<true>(RecyclerSweep&, SmallNormalWithBarrierHeapBlock \*, bool);

template void HeapBucketT<SmallNormalWithBarrierHeapBlock>::SweepHeapBlockList<false>(RecyclerSweep&, SmallNormalWithBarrierHeapBlock \*, bool);

template void HeapBucketT<SmallFinalizableWithBarrierHeapBlock>::SweepHeapBlockList<true>(RecyclerSweep&, SmallFinalizableWithBarrierHeapBlock \*, bool);

template void HeapBucketT<SmallFinalizableWithBarrierHeapBlock>::SweepHeapBlockList<false>(RecyclerSweep&, SmallFinalizableWithBarrierHeapBlock \*, bool);

#endif

template <typename TBlockType>

template <bool pageheap>

void

HeapBucketT<TBlockType>::SweepHeapBlockList(RecyclerSweep& recyclerSweep, TBlockType \* heapBlockList, bool allocable)

{

#if DBG

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

Assert(recyclerSweep.smallBlockVerifyListConsistencyData.hasSetupVerifyListConsistencyData);

recyclerSweep.smallBlockVerifyListConsistencyData.hasSetupVerifyListConsistencyData = false;

}

else if (TBlockType::HeapBlockAttributes::IsMediumBlock)

{

Assert(recyclerSweep.mediumBlockVerifyListConsistencyData.hasSetupVerifyListConsistencyData);

recyclerSweep.mediumBlockVerifyListConsistencyData.hasSetupVerifyListConsistencyData = false;

}

else

{

Assert(false);

}

#endif

Recycler \* recycler = recyclerSweep.GetRecycler();

// Whether we run in thread or background thread, we want to queue up pending sweep

// only if we are doing partial GC so we can calculate the heuristics before

// determinate we want to fully sweep the block or partially sweep the block

#ifdef PARTIAL\_GC\_ENABLED

// CONCURRENT-TODO: Add a mode where we can do in thread sweep, and concurrent partial sweep?

bool const queuePendingSweep = this->DoQueuePendingSweep(recycler);

#else

bool const queuePendingSweep = false;

#endif

Assert(this->IsAllocationStopped());

HeapBlockList::ForEachEditing(heapBlockList, [=, &recyclerSweep](TBlockType \* heapBlock)

{

// The whole list need to be consistent

DebugOnly(VerifyBlockConsistencyInList(heapBlock, recyclerSweep));

SweepState state = heapBlock->Sweep<pageheap>(recyclerSweep, queuePendingSweep, allocable);

DebugOnly(VerifyBlockConsistencyInList(heapBlock, recyclerSweep, state));

switch (state)

{

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

case SweepStatePendingSweep:

{

Assert(IsNormalBucket);

// blocks that have swept object. Queue up the block for concurrent sweep.

Assert(queuePendingSweep);

TBlockType \*& pendingSweepList = recyclerSweep.GetPendingSweepBlockList(this);

heapBlock->SetNextBlock(pendingSweepList);

pendingSweepList = heapBlock;

#ifdef PARTIAL\_GC\_ENABLED

recyclerSweep.NotifyAllocableObjects(heapBlock);

#endif

break;

}

#endif

case SweepStatePendingDispose:

{

Assert(!recyclerSweep.IsBackground());

#ifdef RECYCLER\_WRITE\_BARRIER

Assert(IsFinalizableBucket || IsFinalizableWriteBarrierBucket);

#else

Assert(IsFinalizableBucket);

#endif

DebugOnly(heapBlock->AsFinalizableBlock<TBlockType::HeapBlockAttributes>()->SetIsPendingDispose());

// These are the blocks that have swept finalizable object

// We already transferred the non finalizable swept objects when we are not doing

// concurrent collection, so we only need to queue up the blocks that have

// finalizable objects, so that we can go through and call the dispose, and then

// transfer the finalizable object back to the free list.

SmallFinalizableHeapBucketT<TBlockType::HeapBlockAttributes> \* finalizableHeapBucket = (SmallFinalizableHeapBucketT<TBlockType::HeapBlockAttributes>\*)this;

heapBlock->AsFinalizableBlock<TBlockType::HeapBlockAttributes>()->SetNextBlock(finalizableHeapBucket->pendingDisposeList);

finalizableHeapBucket->pendingDisposeList = heapBlock->AsFinalizableBlock<TBlockType::HeapBlockAttributes>();

Assert(!recycler->hasPendingTransferDisposedObjects);

recycler->hasDisposableObject = true;

break;

}

case SweepStateSwept:

{

Assert(this->nextAllocableBlockHead == nullptr);

Assert(heapBlock->HasFreeObject());

heapBlock->SetNextBlock(this->heapBlockList);

this->heapBlockList = heapBlock;

#ifdef PARTIAL\_GC\_ENABLED

recyclerSweep.NotifyAllocableObjects(heapBlock);

#endif

break;

}

case SweepStateFull:

{

Assert(!heapBlock->HasFreeObject());

heapBlock->SetNextBlock(this->fullBlockList);

this->fullBlockList = heapBlock;

break;

}

case SweepStateEmpty:

{

// the block is empty, just free them

#ifdef RECYCLER\_MEMORY\_VERIFY

// Let's verify it before we free it

if (recycler->VerifyEnabled())

{

heapBlock->Verify();

}

#endif

RECYCLER\_STATS\_INC(recycler, numEmptySmallBlocks[heapBlock->GetHeapBlockType()]);

// CONCURRENT-TODO: Finalizable block never have background == true and always be processed

// in thread, so it will not queue up the pages even if we are doing concurrent GC

if (recyclerSweep.IsBackground())

{

#ifdef RECYCLER\_WRITE\_BARRIER

Assert(!(IsFinalizableBucket || IsFinalizableWriteBarrierBucket));

#else

Assert(!IsFinalizableBucket);

#endif

// CONCURRENT-TODO: We will zero heap block even if the number free page pool exceed

// the maximum and will get decommitted anyway

recyclerSweep.QueueEmptyHeapBlock<TBlockType, pageheap>(this, heapBlock);

RECYCLER\_STATS\_INC(recycler, numZeroedOutSmallBlocks);

}

else

{

// Just free the page in thread (and zero the page)

heapBlock->ReleasePagesSweep<pageheap>(recycler);

FreeHeapBlock(heapBlock);

RECYCLER\_SLOW\_CHECK(this->heapBlockCount--);

}

break;

}

}

});

}

template <typename TBlockType>

template <bool pageheap>

void

HeapBucketT<TBlockType>::SweepBucket(RecyclerSweep& recyclerSweep)

{

DebugOnly(TBlockType \* savedNextAllocableBlockHead);

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount(recyclerSweep.IsBackground()));

if (recyclerSweep.HasSetupBackgroundSweep())

{

// SetupBackgroundSweep set nextAllocableBlockHead to null already

Assert(IsAllocationStopped());

DebugOnly(savedNextAllocableBlockHead = recyclerSweep.GetSavedNextAllocableBlockHead(this));

}

else

{

Assert(AllocatorsAreEmpty());

DebugOnly(savedNextAllocableBlockHead = this->nextAllocableBlockHead);

this->StopAllocationBeforeSweep();

}

// We just started sweeping. These pending lists should be empty

#ifdef CONCURRENT\_GC\_ENABLED

Assert(recyclerSweep.GetPendingSweepBlockList(this) == nullptr);

#else

Assert(!recyclerSweep.IsBackground());

#endif

#if DBG

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

recyclerSweep.SetupVerifyListConsistencyDataForSmallBlock((SmallHeapBlock\*) savedNextAllocableBlockHead, true, false);

}

else if (TBlockType::HeapBlockAttributes::IsMediumBlock)

{

recyclerSweep.SetupVerifyListConsistencyDataForMediumBlock((MediumHeapBlock\*) savedNextAllocableBlockHead, true, false);

}

else

{

Assert(false);

}

#endif

// Move the list locally. We will relink them during sweep

TBlockType \* currentFullBlockList = fullBlockList;

TBlockType \* currentHeapBlockList = heapBlockList;

this->heapBlockList = nullptr;

this->fullBlockList = nullptr;

this->SweepHeapBlockList<pageheap>(recyclerSweep, currentHeapBlockList, true);

#if DBG

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

recyclerSweep.SetupVerifyListConsistencyDataForSmallBlock(nullptr, true, false);

}

else if (TBlockType::HeapBlockAttributes::IsMediumBlock)

{

recyclerSweep.SetupVerifyListConsistencyDataForMediumBlock(nullptr, true, false);

}

else

{

Assert(false);

}

#endif

this->SweepHeapBlockList<pageheap>(recyclerSweep, currentFullBlockList, false);

// We shouldn't have allocate from any block yet

Assert(this->nextAllocableBlockHead == nullptr);

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::StopAllocationBeforeSweep()

{

Assert(!this->IsAllocationStopped());

DebugOnly(this->isAllocationStopped = true);

this->nextAllocableBlockHead = nullptr;

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::StartAllocationAfterSweep()

{

Assert(this->IsAllocationStopped());

DebugOnly(this->isAllocationStopped = false);

this->nextAllocableBlockHead = this->heapBlockList;

}

#if DBG

template <typename TBlockType>

bool

HeapBucketT<TBlockType>::IsAllocationStopped() const

{

if (this->isAllocationStopped)

{

Assert(this->nextAllocableBlockHead == nullptr);

return true;

}

return false;

}

#endif

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

template <typename TBlockType>

uint

HeapBucketT<TBlockType>::Rescan(Recycler \* recycler, RescanFlags flags)

{

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount(!!recycler->IsConcurrentMarkState()));

// If we do the final rescan concurrently, the main thread will prepare for sweep concurrently

// If we do rescan in thread, we will need to prepare sweep here.

// However, if we are in the rescan for OOM, we have already done it, so no need to do it again

if (!recycler->IsConcurrentMarkState() && !recycler->inEndMarkOnLowMemory)

{

this->PrepareSweep();

}

// By default heap bucket doesn't rescan anything

return 0;

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

template <typename TBlockType>

void

HeapBucketT<TBlockType>::MergeNewHeapBlock(TBlockType \* heapBlock)

{

Assert(heapBlock->GetObjectSize() == this->sizeCat);

heapBlock->SetNextBlock(this->heapBlockList);

this->heapBlockList = heapBlock;

RECYCLER\_SLOW\_CHECK(::InterlockedDecrement(&this->newHeapBlockCount));

RECYCLER\_SLOW\_CHECK(this->heapBlockCount++);

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::SetupBackgroundSweep(RecyclerSweep& recyclerSweep)

{

// Don't allocate from existing block temporary when concurrent sweeping

// Currently Rescan clear allocators, if we remove the uncollectedAllocBytes there, we can

// avoid it there and do it here.

Assert(this->AllocatorsAreEmpty());

DebugOnly(recyclerSweep.SaveNextAllocableBlockHead(this));

Assert(recyclerSweep.GetPendingSweepBlockList(this) == nullptr);

this->StopAllocationBeforeSweep();

}

#ifdef RECYCLER\_PAGE\_HEAP

template <typename TBlockType>

void

HeapBucketT<TBlockType>::PageHeapCheckSweepLists(RecyclerSweep& recyclerSweep)

{

#if DBG

if (IsPageHeapEnabled())

{

// in page heap mode. both the heapBlockList and pendingSweepList should not contain page heap block

// Since all block should be "empty" if there are swept object.

// But the list might not be Null because page heap is not enabled for integrated page heapblock

HeapBlockList::ForEach(this->heapBlockList, [](TBlockType \* heapBlock)

{

Assert(!heapBlock->InPageHeapMode());

});

HeapBlockList::ForEach(recyclerSweep.GetPendingSweepBlockList(this), [](TBlockType \* heapBlock)

{

Assert(!heapBlock->InPageHeapMode());

});

}

#endif

}

#endif

template <typename TBlockType>

void

HeapBucketT<TBlockType>::AppendAllocableHeapBlockList(TBlockType \* list)

{

// Add the list to the end of the current list

TBlockType \* currentHeapBlockList = this->heapBlockList;

if (currentHeapBlockList == nullptr)

{

// There weren't any heap block list before, just move the list over and start allocate from it

this->heapBlockList = list;

this->nextAllocableBlockHead = list;

}

else

{

// Find the last block and append the pendingSwpetList

TBlockType \* tail = HeapBlockList::Tail(currentHeapBlockList);

Assert(tail != nullptr);

tail->SetNextBlock(list);

// If we are not currently allocating from the existing heapBlockList,

// that means fill all the exiting one already, we should start with what we just appended.

if (this->nextAllocableBlockHead == nullptr)

{

this->nextAllocableBlockHead = list;

}

}

}

#endif

template <typename TBlockType>

void

HeapBucketT<TBlockType>::EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size))

{

UpdateAllocators();

HeapBucket::EnumerateObjects(fullBlockList, infoBits, CallBackFunction);

HeapBucket::EnumerateObjects(heapBlockList, infoBits, CallBackFunction);

}

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <typename TBlockType>

void

HeapBucketT<TBlockType>::VerifyHeapBlockCount(bool background)

{

// TODO-REFACTOR: GetNonEmptyHeapBlockCount really should be virtual

static\_cast<SmallHeapBlockType<TBlockType::RequiredAttributes, TBlockType::HeapBlockAttributes>::BucketType \*>(this)->GetNonEmptyHeapBlockCount(true);

if (!background)

{

this->GetEmptyHeapBlockCount();

}

}

template <typename TBlockType>

size\_t

HeapBucketT<TBlockType>::Check(bool checkCount)

{

Assert(this->GetRecycler()->recyclerSweep == nullptr);

UpdateAllocators();

size\_t smallHeapBlockCount = HeapInfo::Check(true, false, this->fullBlockList);

smallHeapBlockCount += HeapInfo::Check(true, false, this->heapBlockList, this->nextAllocableBlockHead);

smallHeapBlockCount += HeapInfo::Check(false, false, this->nextAllocableBlockHead);

Assert(!checkCount || this->heapBlockCount == smallHeapBlockCount);

return smallHeapBlockCount;

}

#endif

#ifdef DUMP\_FRAGMENTATION\_STATS

template <typename TBlockType>

void

HeapBucketT<TBlockType>::AggregateBucketStats(HeapBucketStats& stats)

{

auto allocatorHead = &this->allocatorHead;

auto allocatorCurr = allocatorHead;

do

{

TBlockType\* allocatorHeapBlock = allocatorCurr->GetHeapBlock();

if (allocatorHeapBlock)

{

allocatorHeapBlock->AggregateBlockStats(stats, true, allocatorCurr->freeObjectList, allocatorCurr->endAddress != 0);

}

allocatorCurr = allocatorCurr->GetNext();

} while (allocatorCurr != allocatorHead);

auto blockStatsAggregator = [&stats](TBlockType\* heapBlock) {

heapBlock->AggregateBlockStats(stats);

};

HeapBlockList::ForEach(emptyBlockList, blockStatsAggregator);

HeapBlockList::ForEach(fullBlockList, blockStatsAggregator);

HeapBlockList::ForEach(heapBlockList, blockStatsAggregator);

}

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

template <typename TBlockType>

void

HeapBucketT<TBlockType>::Verify()

{

UpdateAllocators();

#if DBG

RecyclerVerifyListConsistencyData recyclerVerifyListConsistencyData;

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

recyclerVerifyListConsistencyData.smallBlockVerifyListConsistencyData.SetupVerifyListConsistencyData((SmallHeapBlock\*) nullptr, true, false);

}

else if (TBlockType::HeapBlockAttributes::IsMediumBlock)

{

recyclerVerifyListConsistencyData.mediumBlockVerifyListConsistencyData.SetupVerifyListConsistencyData((MediumHeapBlock\*) nullptr, true, false);

}

else

{

Assert(false);

}

#endif

HeapBlockList::ForEach(fullBlockList, [DebugOnly(&recyclerVerifyListConsistencyData)](TBlockType \* heapBlock)

{

DebugOnly(VerifyBlockConsistencyInList(heapBlock, recyclerVerifyListConsistencyData));

heapBlock->Verify();

});

#if DBG

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

recyclerVerifyListConsistencyData.smallBlockVerifyListConsistencyData.SetupVerifyListConsistencyData((SmallHeapBlock\*) this->nextAllocableBlockHead, true, false);

}

else if (TBlockType::HeapBlockAttributes::IsMediumBlock)

{

recyclerVerifyListConsistencyData.mediumBlockVerifyListConsistencyData.SetupVerifyListConsistencyData((MediumHeapBlock\*) this->nextAllocableBlockHead, true, false);

}

else

{

Assert(false);

}

#endif

HeapBlockList::ForEach(heapBlockList, [this, DebugOnly(&recyclerVerifyListConsistencyData)](TBlockType \* heapBlock)

{

DebugOnly(VerifyBlockConsistencyInList(heapBlock, recyclerVerifyListConsistencyData));

char \* bumpAllocateAddress = nullptr;

this->ForEachAllocator([heapBlock, &bumpAllocateAddress](TBlockAllocatorType \* allocator)

{

if (allocator->GetHeapBlock() == heapBlock && allocator->GetEndAddress() != nullptr)

{

Assert(bumpAllocateAddress == nullptr);

bumpAllocateAddress = (char \*)allocator->GetFreeObjectList();

}

});

if (bumpAllocateAddress != nullptr)

{

heapBlock->VerifyBumpAllocated(bumpAllocateAddress);

}

else

{

heapBlock->Verify(false);

}

});

}

#endif

#ifdef RECYCLER\_VERIFY\_MARK

template <typename TBlockType>

void

HeapBucketT<TBlockType>::VerifyMark()

{

HeapBlockList::ForEach(this->fullBlockList, [](TBlockType \* heapBlock)

{

heapBlock->VerifyMark();

});

HeapBlockList::ForEach(this->heapBlockList, [](TBlockType \* heapBlock)

{

heapBlock->VerifyMark();

});

}

#endif

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::Initialize(HeapInfo \* heapInfo, uint sizeCat)

{

heapBucket.Initialize(heapInfo, sizeCat);

leafHeapBucket.Initialize(heapInfo, sizeCat);

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.Initialize(heapInfo, sizeCat);

smallFinalizableWithBarrierHeapBucket.Initialize(heapInfo, sizeCat);

#endif

finalizableHeapBucket.Initialize(heapInfo, sizeCat);

}

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::ResetMarks(ResetMarkFlags flags)

{

heapBucket.ResetMarks(flags);

leafHeapBucket.ResetMarks(flags);

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.ResetMarks(flags);

smallFinalizableWithBarrierHeapBucket.ResetMarks(flags);

#endif

// Although we pass in premarkFreeObjects, the finalizable heap bucket ignores

// this parameter and never pre-marks free objects

finalizableHeapBucket.ResetMarks(flags);

}

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::ScanInitialImplicitRoots(Recycler \* recycler)

{

heapBucket.ScanInitialImplicitRoots(recycler);

// Don't need to scan implicit roots on leaf heap bucket

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.ScanInitialImplicitRoots(recycler);

smallFinalizableWithBarrierHeapBucket.ScanInitialImplicitRoots(recycler);

#endif

finalizableHeapBucket.ScanInitialImplicitRoots(recycler);

}

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::ScanNewImplicitRoots(Recycler \* recycler)

{

heapBucket.ScanNewImplicitRoots(recycler);

// Need to scan new implicit roots on leaf heap bucket

leafHeapBucket.ScanNewImplicitRoots(recycler);

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.ScanNewImplicitRoots(recycler);

smallFinalizableWithBarrierHeapBucket.ScanNewImplicitRoots(recycler);

#endif

finalizableHeapBucket.ScanNewImplicitRoots(recycler);

}

// TODO: Fix template args

template void HeapBucketGroup<SmallAllocationBlockAttributes>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void HeapBucketGroup<SmallAllocationBlockAttributes>::Sweep<false>(RecyclerSweep& recyclerSweep);

template void HeapBucketGroup<MediumAllocationBlockAttributes>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void HeapBucketGroup<MediumAllocationBlockAttributes>::Sweep<false>(RecyclerSweep& recyclerSweep);

template <class TBlockAttributes>

template<bool pageheap>

void

HeapBucketGroup<TBlockAttributes>::Sweep(RecyclerSweep& recyclerSweep)

{

heapBucket.Sweep<pageheap>(recyclerSweep);

leafHeapBucket.Sweep<pageheap>(recyclerSweep);

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.Sweep<pageheap>(recyclerSweep);

#endif

}

template void HeapBucketGroup<SmallAllocationBlockAttributes>::SweepFinalizableObjects<true>(RecyclerSweep& recyclerSweep);

template void HeapBucketGroup<SmallAllocationBlockAttributes>::SweepFinalizableObjects<false>(RecyclerSweep& recyclerSweep);

template void HeapBucketGroup<MediumAllocationBlockAttributes>::SweepFinalizableObjects<true>(RecyclerSweep& recyclerSweep);

template void HeapBucketGroup<MediumAllocationBlockAttributes>::SweepFinalizableObjects<false>(RecyclerSweep& recyclerSweep);

// Sweep finalizable objects first to ensure that if they reference any other

// objects in the finalizer - they are valid

template <class TBlockAttributes>

template<bool pageheap>

void

HeapBucketGroup<TBlockAttributes>::SweepFinalizableObjects(RecyclerSweep& recyclerSweep)

{

finalizableHeapBucket.Sweep<pageheap>(recyclerSweep);

#ifdef RECYCLER\_WRITE\_BARRIER

smallFinalizableWithBarrierHeapBucket.Sweep<pageheap>(recyclerSweep);

#endif

}

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::DisposeObjects()

{

finalizableHeapBucket.DisposeObjects();

#ifdef RECYCLER\_WRITE\_BARRIER

smallFinalizableWithBarrierHeapBucket.DisposeObjects();

#endif

}

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::TransferDisposedObjects()

{

finalizableHeapBucket.TransferDisposedObjects();

#ifdef RECYCLER\_WRITE\_BARRIER

smallFinalizableWithBarrierHeapBucket.TransferDisposedObjects();

#endif

}

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size))

{

heapBucket.EnumerateObjects(infoBits, CallBackFunction);

leafHeapBucket.EnumerateObjects(infoBits, CallBackFunction);

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.EnumerateObjects(infoBits, CallBackFunction);

smallFinalizableWithBarrierHeapBucket.EnumerateObjects(infoBits, CallBackFunction);

#endif

finalizableHeapBucket.EnumerateObjects(infoBits, CallBackFunction);

}

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::FinalizeAllObjects()

{

finalizableHeapBucket.FinalizeAllObjects();

#ifdef RECYCLER\_WRITE\_BARRIER

smallFinalizableWithBarrierHeapBucket.FinalizeAllObjects();

#endif

}

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

template <class TBlockAttributes>

uint

HeapBucketGroup<TBlockAttributes>::Rescan(Recycler \* recycler, RescanFlags flags)

{

return heapBucket.Rescan(recycler, flags) +

leafHeapBucket.Rescan(recycler, flags) +

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.Rescan(recycler, flags) +

smallFinalizableWithBarrierHeapBucket.Rescan(recycler, flags) +

#endif

finalizableHeapBucket.Rescan(recycler, flags);

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::PrepareSweep()

{

heapBucket.PrepareSweep();

leafHeapBucket.PrepareSweep();

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.PrepareSweep();

smallFinalizableWithBarrierHeapBucket.PrepareSweep();

#endif

finalizableHeapBucket.PrepareSweep();

}

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::SetupBackgroundSweep(RecyclerSweep& recyclerSweep)

{

heapBucket.SetupBackgroundSweep(recyclerSweep);

leafHeapBucket.SetupBackgroundSweep(recyclerSweep);

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.SetupBackgroundSweep(recyclerSweep);

#endif

}

#endif

#ifdef PARTIAL\_GC\_ENABLED

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::SweepPartialReusePages(RecyclerSweep& recyclerSweep)

{

#if DBG && defined(RECYCLER\_PAGE\_HEAP)

this->heapBucket.PageHeapCheckSweepLists(recyclerSweep);

#ifdef RECYCLER\_WRITE\_BARRIER

this->smallNormalWithBarrierHeapBucket.PageHeapCheckSweepLists(recyclerSweep);

this->smallFinalizableWithBarrierHeapBucket.PageHeapCheckSweepLists(recyclerSweep);

#endif

this->finalizableHeapBucket.PageHeapCheckSweepLists(recyclerSweep);

#endif

// Leaf heap bucket are always reused for allocation and can be done on the concurrent thread

// WriteBarrier-TODO: Do the same for write barrier buckets

heapBucket.SweepPartialReusePages(recyclerSweep);

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.SweepPartialReusePages(recyclerSweep);

smallFinalizableWithBarrierHeapBucket.SweepPartialReusePages(recyclerSweep);

#endif

finalizableHeapBucket.SweepPartialReusePages(recyclerSweep);

}

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::FinishPartialCollect(RecyclerSweep \* recyclerSweep)

{

heapBucket.FinishPartialCollect(recyclerSweep);

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.FinishPartialCollect(recyclerSweep);

smallFinalizableWithBarrierHeapBucket.FinishPartialCollect(recyclerSweep);

#endif

finalizableHeapBucket.FinishPartialCollect(recyclerSweep);

// Leaf heap block always do a full sweep instead of partial sweep

// (since touching the page doesn't affect rescan)

// So just need to verify heap block count (which finishPartialCollect would have done)

// WriteBarrier-TODO: Do that same for write barrier buckets

RECYCLER\_SLOW\_CHECK(leafHeapBucket.VerifyHeapBlockCount(recyclerSweep != nullptr && recyclerSweep->IsBackground()));

}

#endif

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::SweepPendingObjects(RecyclerSweep& recyclerSweep)

{

// For leaf buckets, we can always reuse the page as we don't need to rescan them for partial GC

// It should have been swept immediately during Sweep

// WriteBarrier-TODO: Do the same for write barrier buckets

Assert(recyclerSweep.GetPendingSweepBlockList(&leafHeapBucket) == nullptr);

heapBucket.SweepPendingObjects(recyclerSweep);

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.SweepPendingObjects(recyclerSweep);

smallFinalizableWithBarrierHeapBucket.SweepPendingObjects(recyclerSweep);

#endif

finalizableHeapBucket.SweepPendingObjects(recyclerSweep);

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::TransferPendingEmptyHeapBlocks(RecyclerSweep& recyclerSweep)

{

recyclerSweep.TransferPendingEmptyHeapBlocks(&heapBucket);

recyclerSweep.TransferPendingEmptyHeapBlocks(&leafHeapBucket);

#ifdef RECYCLER\_WRITE\_BARRIER

recyclerSweep.TransferPendingEmptyHeapBlocks(&smallNormalWithBarrierHeapBucket);

recyclerSweep.TransferPendingEmptyHeapBlocks(&smallFinalizableWithBarrierHeapBucket);

#endif

recyclerSweep.TransferPendingEmptyHeapBlocks(&finalizableHeapBucket);

}

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

template <class TBlockAttributes>

size\_t

HeapBucketGroup<TBlockAttributes>::GetNonEmptyHeapBlockCount(bool checkCount) const

{

return heapBucket.GetNonEmptyHeapBlockCount(checkCount) +

finalizableHeapBucket.GetNonEmptyHeapBlockCount(checkCount) +

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.GetNonEmptyHeapBlockCount(checkCount) +

smallFinalizableWithBarrierHeapBucket.GetNonEmptyHeapBlockCount(checkCount) +

#endif

leafHeapBucket.GetNonEmptyHeapBlockCount(checkCount);

}

template <class TBlockAttributes>

size\_t

HeapBucketGroup<TBlockAttributes>::GetEmptyHeapBlockCount() const

{

return heapBucket.GetEmptyHeapBlockCount() +

finalizableHeapBucket.GetEmptyHeapBlockCount() +

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.GetEmptyHeapBlockCount() +

smallFinalizableWithBarrierHeapBucket.GetEmptyHeapBlockCount() +

#endif

leafHeapBucket.GetEmptyHeapBlockCount();

}

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <class TBlockAttributes>

size\_t

HeapBucketGroup<TBlockAttributes>::Check()

{

return heapBucket.Check() + finalizableHeapBucket.Check() + leafHeapBucket.Check()

#ifdef RECYCLER\_WRITE\_BARRIER

+ smallNormalWithBarrierHeapBucket.Check() + smallFinalizableWithBarrierHeapBucket.Check()

#endif

;

}

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::Verify()

{

heapBucket.Verify();

finalizableHeapBucket.Verify();

leafHeapBucket.Verify();

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.Verify();

smallFinalizableWithBarrierHeapBucket.Verify();

#endif

}

#endif

#ifdef RECYCLER\_VERIFY\_MARK

template <class TBlockAttributes>

void

HeapBucketGroup<TBlockAttributes>::VerifyMark()

{

heapBucket.VerifyMark();

finalizableHeapBucket.VerifyMark();

leafHeapBucket.VerifyMark();

#ifdef RECYCLER\_WRITE\_BARRIER

smallNormalWithBarrierHeapBucket.VerifyMark();

smallFinalizableWithBarrierHeapBucket.VerifyMark();

#endif

}

#endif

#if DBG

template <class TBlockAttributes>

bool

HeapBucketGroup<TBlockAttributes>::AllocatorsAreEmpty()

{

return heapBucket.AllocatorsAreEmpty()

&& finalizableHeapBucket.AllocatorsAreEmpty()

&& leafHeapBucket.AllocatorsAreEmpty()

#ifdef RECYCLER\_WRITE\_BARRIER

&& smallNormalWithBarrierHeapBucket.AllocatorsAreEmpty()

&& smallFinalizableWithBarrierHeapBucket.AllocatorsAreEmpty()

#endif

;

}

#endif

template class HeapBucketGroup<SmallAllocationBlockAttributes>;

template class HeapBucketGroup<MediumAllocationBlockAttributes>;

template void HeapBucketT<SmallFinalizableHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<SmallFinalizableHeapBlock>::SweepBucket<false>(RecyclerSweep&);

template void HeapBucketT<SmallNormalHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<SmallNormalHeapBlock>::SweepBucket<false>(RecyclerSweep&);

template void HeapBucketT<SmallLeafHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<SmallLeafHeapBlock>::SweepBucket<false>(RecyclerSweep&);

#ifdef RECYCLER\_WRITE\_BARRIER

template void HeapBucketT<SmallFinalizableWithBarrierHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<SmallFinalizableWithBarrierHeapBlock>::SweepBucket<false>(RecyclerSweep&);

template void HeapBucketT<SmallNormalWithBarrierHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<SmallNormalWithBarrierHeapBlock>::SweepBucket<false>(RecyclerSweep&);

#endif

template void HeapBucketT<MediumFinalizableHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<MediumFinalizableHeapBlock>::SweepBucket<false>(RecyclerSweep&);

template void HeapBucketT<MediumNormalHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<MediumNormalHeapBlock>::SweepBucket<false>(RecyclerSweep&);

template void HeapBucketT<MediumLeafHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<MediumLeafHeapBlock>::SweepBucket<false>(RecyclerSweep&);

#ifdef RECYCLER\_WRITE\_BARRIER

template void HeapBucketT<MediumFinalizableWithBarrierHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<MediumFinalizableWithBarrierHeapBlock>::SweepBucket<false>(RecyclerSweep&);

template void HeapBucketT<MediumNormalWithBarrierHeapBlock>::SweepBucket<true>(RecyclerSweep&);

template void HeapBucketT<MediumNormalWithBarrierHeapBlock>::SweepBucket<false>(RecyclerSweep&);

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

// a size bucket in the heap for small object

class HeapInfo;

class RecyclerSweep;

#if DBG

template <class TBlockAttributes>

class GenericRecyclerVerifyListConsistencyData

{

public:

GenericRecyclerVerifyListConsistencyData() {};

// Temporary data for Sweep list consistency checks

bool expectFull;

bool expectDispose;

SmallHeapBlockT<TBlockAttributes> \* nextAllocableBlockHead;

bool hasSetupVerifyListConsistencyData;

template <typename TBlockAttributes>

void SetupVerifyListConsistencyData(SmallHeapBlockT<TBlockAttributes>\* block, bool expectFull, bool expectDispose)

{

this->nextAllocableBlockHead = block;

this->expectFull = expectFull;

this->expectDispose = expectDispose;

this->hasSetupVerifyListConsistencyData = true;

}

};

class RecyclerVerifyListConsistencyData

{

public:

RecyclerVerifyListConsistencyData() {};

void SetupVerifyListConsistencyDataForSmallBlock(SmallHeapBlock\* block, bool expectFull, bool expectDispose)

{

this->smallBlockVerifyListConsistencyData.nextAllocableBlockHead = block;

this->smallBlockVerifyListConsistencyData.expectFull = expectFull;

this->smallBlockVerifyListConsistencyData.expectDispose = expectDispose;

this->smallBlockVerifyListConsistencyData.hasSetupVerifyListConsistencyData = true;

}

void SetupVerifyListConsistencyDataForMediumBlock(MediumHeapBlock\* block, bool expectFull, bool expectDispose)

{

this->mediumBlockVerifyListConsistencyData.nextAllocableBlockHead = block;

this->mediumBlockVerifyListConsistencyData.expectFull = expectFull;

this->mediumBlockVerifyListConsistencyData.expectDispose = expectDispose;

this->mediumBlockVerifyListConsistencyData.hasSetupVerifyListConsistencyData = true;

}

GenericRecyclerVerifyListConsistencyData<SmallAllocationBlockAttributes> smallBlockVerifyListConsistencyData;

GenericRecyclerVerifyListConsistencyData<MediumAllocationBlockAttributes> mediumBlockVerifyListConsistencyData;

};

#endif

// NOTE: HeapBucket can't have vtable, because we allocate them inline with recycler with custom initializer

class HeapBucket

{

public:

HeapBucket();

uint GetBucketIndex() const;

uint GetMediumBucketIndex() const;

protected:

template <typename TBlockType>

static void EnumerateObjects(TBlockType \* heapBlockList, ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size));

HeapInfo \* heapInfo;

uint sizeCat;

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t heapBlockCount;

size\_t newHeapBlockCount; // count of heap bock that is in the heap info and not in the heap bucket yet

size\_t emptyHeapBlockCount;

#endif

#ifdef RECYCLER\_PAGE\_HEAP

bool isPageHeapEnabled;

\_\_inline bool IsPageHeapEnabled() const { return isPageHeapEnabled; }

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

Recycler \* GetRecycler() const;

#endif

template <typename TBlockType>

friend class SmallHeapBlockAllocator;

template <typename TBlockAttributes>

friend class SmallHeapBlockT;

template <typename TBlockAttributes>

friend class SmallFinalizableHeapBlockT;

friend class LargeHeapBlock;

#ifdef RECYCLER\_WRITE\_BARRIER

template <typename TBlockAttributes>

friend class SmallFinalizableWithBarrierHeapBlockT;

template <typename TBlockAttributes>

friend class SmallNormalWithBarrierHeapBlockT;

#endif

};

template <typename TBlockType>

class HeapBucketT : public HeapBucket

{

typedef typename SmallHeapBlockAllocator<TBlockType> TBlockAllocatorType;

public:

HeapBucketT();

~HeapBucketT();

bool IntegrateBlock(char \* blockAddress, PageSegment \* segment, Recycler \* recycler);

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char \* RealAlloc(Recycler \* recycler, size\_t sizeCat);

#ifdef RECYCLER\_PAGE\_HEAP

char \* PageHeapAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes, PageHeapMode mode, bool nothrow);

void PageHeapCheckSweepLists(RecyclerSweep& recyclerSweep);

#endif

void ExplicitFree(void\* object, size\_t sizeCat);

char \* SnailAlloc(Recycler \* recycler, TBlockAllocatorType \* allocator, size\_t sizeCat, ObjectInfoBits attributes, bool nothrow);

void ResetMarks(ResetMarkFlags flags);

void ScanNewImplicitRoots(Recycler \* recycler);

#ifdef DUMP\_FRAGMENTATION\_STATS

void AggregateBucketStats(HeapBucketStats& stats);

#endif

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

uint Rescan(Recycler \* recycler, RescanFlags flags);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

void MergeNewHeapBlock(TBlockType \* heapBlock);

void PrepareSweep();

void SetupBackgroundSweep(RecyclerSweep& recyclerSweep);

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

friend class ScriptMemoryDumper;

#endif

TBlockAllocatorType \* GetAllocator() { return &allocatorHead;}

protected:

static bool const IsLeafBucket = TBlockType::RequiredAttributes == LeafBit;

static bool const IsFinalizableBucket = TBlockType::RequiredAttributes == FinalizeBit;

static bool const IsNormalBucket = TBlockType::RequiredAttributes == NoBit;

#ifdef RECYCLER\_WRITE\_BARRIER

static bool const IsWriteBarrierBucket = TBlockType::RequiredAttributes == WithBarrierBit;

static bool const IsFinalizableWriteBarrierBucket = TBlockType::RequiredAttributes == FinalizableWithBarrierBit;

#endif

void Initialize(HeapInfo \* heapInfo, uint sizeCat);

void AppendAllocableHeapBlockList(TBlockType \* list);

void DeleteHeapBlockList(TBlockType \* list);

static void DeleteEmptyHeapBlockList(TBlockType \* list);

static void DeleteHeapBlockList(TBlockType \* list, Recycler \* recycler);

// Small allocators

void UpdateAllocators();

void ClearAllocators();

void AddAllocator(TBlockAllocatorType \* allocator);

void RemoveAllocator(TBlockAllocatorType \* allocator);

static void ClearAllocator(TBlockAllocatorType \* allocator);

template <class Fn> void ForEachAllocator(Fn fn);

// Allocations

char \* TryAllocFromNewHeapBlock(Recycler \* recycler, TBlockAllocatorType \* allocator, size\_t sizeCat, ObjectInfoBits attributes);

char \* TryAlloc(Recycler \* recycler, TBlockAllocatorType \* allocator, size\_t sizeCat, ObjectInfoBits attributes);

template<bool pageheap>

TBlockType \* CreateHeapBlock(Recycler \* recycler);

TBlockType \* GetUnusedHeapBlock();

void FreeHeapBlock(TBlockType \* heapBlock);

// GC

template<bool pageheap>

void SweepBucket(RecyclerSweep& recyclerSweep);

template <bool pageheap, typename Fn>

void SweepBucket(RecyclerSweep& recyclerSweep, Fn sweepFn);

void StopAllocationBeforeSweep();

void StartAllocationAfterSweep();

#if DBG

bool IsAllocationStopped() const;

#endif

template<bool pageheap>

void SweepHeapBlockList(RecyclerSweep& recyclerSweep, TBlockType \* heapBlockList, bool allocable);

#if defined(PARTIAL\_GC\_ENABLED)

bool DoQueuePendingSweep(Recycler \* recycler);

bool DoPartialReuseSweep(Recycler \* recycler);

#endif

// Partial/Concurrent GC

void EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size));

#if DBG

bool AllocatorsAreEmpty() const;

bool HasPendingDisposeHeapBlocks() const;

static void VerifyBlockConsistencyInList(TBlockType \* heapBlock, RecyclerVerifyListConsistencyData& recyclerSweep);

static void VerifyBlockConsistencyInList(TBlockType \* heapBlock, RecyclerVerifyListConsistencyData const& recyclerSweep, SweepState state);

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

// TODO-REFACTOR: This really should be virtual

size\_t GetNonEmptyHeapBlockCount(bool checkCount) const;

size\_t GetEmptyHeapBlockCount() const;

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t Check(bool checkCount = true);

void VerifyHeapBlockCount(bool background);

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void Verify();

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void VerifyMark();

#endif

TBlockAllocatorType allocatorHead;

TBlockType \* nextAllocableBlockHead;

TBlockType \* emptyBlockList; // list of blocks that is empty and has it's page freed

TBlockType \* fullBlockList; // list of blocks that are fully allocated

TBlockType \* heapBlockList; // list of blocks that has free objects

FreeObject\* explicitFreeList; // List of objects that have been explicitly freed

TBlockAllocatorType \* lastExplicitFreeListAllocator;

#ifdef RECYCLER\_PAGE\_HEAP

SmallHeapBlock\* explicitFreeLockBlockList; // List of heap blocks which have been locked upon explicit free

#endif

#if DBG

bool isAllocationStopped; // whether the bucket is the middle of sweeping, not including partial sweeping

#endif

template <class TBlockAttributes>

friend class HeapBucketGroup;

friend class HeapInfo;

friend typename TBlockType;

template <class TBucketAttributes>

friend class SmallHeapBlockT;

friend class RecyclerSweep;

};

template <typename TBlockType>

void

HeapBucket::EnumerateObjects(TBlockType \* heapBlockList, ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size))

{

HeapBlockList::ForEach(heapBlockList, [=](TBlockType \* heapBlock)

{

heapBlock->EnumerateObjects(infoBits, CallBackFunction);

});

}

template <typename TBlockType>

template <bool pageheap, typename Fn>

void

HeapBucketT<TBlockType>::SweepBucket(RecyclerSweep& recyclerSweep, Fn sweepFn)

{

this->SweepBucket<pageheap>(recyclerSweep);

// Continue to sweep other list from derived class

sweepFn(recyclerSweep);

#if defined(PARTIAL\_GC\_ENABLED)

if (!this->DoPartialReuseSweep(recyclerSweep.GetRecycler()))

#endif

{

// We should only queue up pending sweep if we are doing partial collect

Assert(recyclerSweep.GetPendingSweepBlockList(this) == nullptr);

// Every thing is swept immediately in non partial collect, so we can allocate

// from the heap block list now

StartAllocationAfterSweep();

}

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount(recyclerSweep.IsBackground()));

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template <typename TBlockType>

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char \*

HeapBucketT<TBlockType>::RealAlloc(Recycler \* recycler, size\_t sizeCat)

{

Assert(sizeCat == this->sizeCat);

char \* memBlock = allocatorHead.InlinedAlloc<(ObjectInfoBits)(attributes & InternalObjectInfoBitMask)>(recycler, sizeCat);

if (memBlock == nullptr)

{

memBlock = SnailAlloc(recycler, &allocatorHead, sizeCat, attributes, nothrow);

Assert(memBlock != nullptr || nothrow);

}

else

{

#ifdef RECYCLER\_PAGE\_HEAP

Assert(allocatorHead.heapBlock == nullptr || !allocatorHead.heapBlock->InPageHeapMode());

#else

Assert(allocatorHead.heapBlock == nullptr);

#endif

}

// If this API is called and throwing is not allowed,

// check if we actually allocated a block before verifying

// its zero fill state. If it is nullptr, return that here.

if (nothrow)

{

if (memBlock == nullptr)

{

return nullptr;

}

}

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

// Do the verify zero fill only if it's not a nothrow alloc

if ((attributes & ObjectInfoBits::LeafBit) == 0

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_THREAD\_PAGE

&& ((attributes & ObjectInfoBits::WithBarrierBit) == 0)

#endif

)

{

// Skip the first and the last pointer objects- the first may have next pointer for the free list

// the last might have the old size of the object if this was allocated from an explicit free list

recycler->VerifyZeroFill(memBlock + sizeof(FreeObject), sizeCat - (2 \* sizeof(FreeObject)));

}

#endif

return memBlock;

}

template <typename TBlockType>

void

HeapBucketT<TBlockType>::ExplicitFree(void\* object, size\_t sizeCat)

{

FreeObject\* explicitFreeObject = (FreeObject\*) object;

if (lastExplicitFreeListAllocator->IsExplicitFreeObjectListAllocMode())

{

explicitFreeObject->SetNext(lastExplicitFreeListAllocator->GetFreeObjectList());

lastExplicitFreeListAllocator->SetFreeObjectList(explicitFreeObject);

}

else

{

explicitFreeObject->SetNext(this->explicitFreeList);

this->explicitFreeList = explicitFreeObject;

}

// Don't fill memory fill pattern here since we're still pretending like the object

// is allocated to other parts of the GC.

}

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

inline

Recycler \*

HeapBucket::GetRecycler() const

{

return this->heapInfo->recycler;

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

class HeapConstants

{

public:

#if defined(\_M\_IX86\_OR\_ARM32)

static const uint MaxSmallObjectSize = 512;

#else

static const uint MaxSmallObjectSize = 768;

#endif

#if SMALLBLOCK\_MEDIUM\_ALLOC

static const uint MaxMediumObjectSize = 8 \* 1024; // Maximum medium object size is 8K

#else

static const uint MaxMediumObjectSize = 9216;

#endif

static const uint ObjectAllocationShift = 4; // 16

static const uint ObjectGranularity = 1 << ObjectAllocationShift;

static const uint BucketCount = (MaxSmallObjectSize >> ObjectAllocationShift);

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

static const uint MediumObjectGranularity = 256;

static const uint MediumBucketCount = (MaxMediumObjectSize - MaxSmallObjectSize) / MediumObjectGranularity;

#endif

};

///

/// BlockAttributes are used to determine the allocation characteristics of a heap block

/// These include the number of pages to allocate, the object capacity of the block

/// and the shape of the object's bit vectors

/// Since the constants here are used while generating the ValidPointerMap constants

/// please remember to regenerate the ValidPointersMap by first switching to a dynamic VPM

/// as controlled by the USE\_STATIC\_VPM in HeapInfo.h, and then running GenValidPointers.cmd

///

class SmallAllocationBlockAttributes

{

public:

static const size\_t MinObjectSize = HeapConstants::ObjectGranularity;

#if defined(\_M\_IX86\_OR\_ARM32)

static const size\_t PageCount = 2;

#else

static const size\_t PageCount = 4;

#endif

static const size\_t BitVectorCount = ((PageCount \* AutoSystemInfo::PageSize) / HeapConstants::ObjectGranularity);

static const ushort MaxAddressBit = BitVectorCount - 1;

static const uint BucketCount = HeapConstants::BucketCount;

static const size\_t BucketGranularity = HeapConstants::ObjectGranularity;

static const uint MaxObjectSize = HeapConstants::MaxSmallObjectSize;

static const uint MaxObjectCount = PageCount \* AutoSystemInfo::PageSize / HeapConstants::ObjectGranularity;

static const uint MaxSmallObjectCount = MaxObjectCount;

static const uint ObjectCountPerPage = AutoSystemInfo::PageSize / HeapConstants::ObjectGranularity;

// This is there for RecyclerSweep to distinguish which bucket index to use

static const bool IsSmallBlock = true;

static const bool IsMediumBlock = false;

static const bool IsLargeBlock = false;

static BOOL IsAlignedObjectSize(size\_t sizeCat);

};

class MediumAllocationBlockAttributes

{

public:

static const size\_t PageCount = 8;

static const size\_t MinObjectSize = HeapConstants::MaxSmallObjectSize;

static const ushort BitVectorCount = ((PageCount \* AutoSystemInfo::PageSize) / HeapConstants::ObjectGranularity);

static const size\_t MaxAddressBit = (BitVectorCount - 1);

static const uint MaxObjectSize = HeapConstants::MaxMediumObjectSize;

static const uint BucketCount = HeapConstants::MediumBucketCount;

static const size\_t BucketGranularity = HeapConstants::MediumObjectGranularity;

static const uint MaxObjectCount = PageCount \* AutoSystemInfo::PageSize / (MinObjectSize + BucketGranularity);

static const uint MaxSmallObjectCount = PageCount \* AutoSystemInfo::PageSize / HeapConstants::ObjectGranularity;

static const uint ObjectCountPerPage = AutoSystemInfo::PageSize / MinObjectSize;

// This is there for RecyclerSweep to distinguish which bucket index to use

static const bool IsSmallBlock = false;

static const bool IsMediumBlock = true;

static const bool IsLargeBlock = false;

static BOOL IsAlignedObjectSize(size\_t sizeCat);

};

class LargeAllocationBlockAttributes

{

public:

// This is there for RecyclerSweep to distinguish which bucket index to use

static const bool IsSmallBlock = false;

static const bool IsMediumBlock = false;

static const bool IsLargeBlock = true;

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#include "Memory\PageHeapBlockTypeFilter.h"

#if defined(\_M\_IX86\_OR\_ARM32)

#include "ValidPointersMap\vpm.32b.h"

#elif defined(\_M\_X64\_OR\_ARM64)

#include "ValidPointersMap\vpm.64b.h"

#else

#error "Platform is not handled"

#endif

template \_\_forceinline char\* HeapInfo::RealAlloc<NoBit, false>(Recycler \* recycler, size\_t sizeCat);

HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes> HeapInfo::smallAllocValidPointersMap;

HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes> HeapInfo::mediumAllocValidPointersMap;

template <class TBlockAttributes>

ValidPointers<TBlockAttributes>::ValidPointers(ushort const \* validPointers)

: validPointers(validPointers)

{

}

template <class TBlockAttributes>

ushort ValidPointers<TBlockAttributes>::GetAddressIndex(uint index) const

{

Assert(index < TBlockAttributes::MaxSmallObjectCount);

return validPointers[index];

}

template <class TBlockAttributes>

ushort ValidPointers<TBlockAttributes>::GetInteriorAddressIndex(uint index) const

{

Assert(index < TBlockAttributes::MaxSmallObjectCount);

return validPointers[index + TBlockAttributes::MaxSmallObjectCount];

}

template <class TBlockAttributes>

const ValidPointers<TBlockAttributes>

HeapInfo::ValidPointersMap<TBlockAttributes>::GetValidPointersForIndex(uint index) const

{

Assert(index < TBlockAttributes::BucketCount);

\_\_analysis\_assume(index < TBlockAttributes::BucketCount);

return validPointersBuffer[index];

}

template <class TBlockAttributes>

const typename SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector \*

HeapInfo::ValidPointersMap<TBlockAttributes>::GetInvalidBitVector(uint index) const

{

Assert(index < TBlockAttributes::BucketCount);

\_\_analysis\_assume(index < TBlockAttributes::BucketCount);

#if USE\_STATIC\_VPM

return &(\*invalidBitsBuffers)[index];

#else

return &invalidBitsBuffers[index];

#endif

}

template <class TBlockAttributes>

const typename SmallHeapBlockT<TBlockAttributes>::BlockInfo \*

HeapInfo::ValidPointersMap<TBlockAttributes>::GetBlockInfo (uint index) const

{

Assert(index < TBlockAttributes::BucketCount);

\_\_analysis\_assume(index < TBlockAttributes::BucketCount);

return blockInfoBuffer[index];

}

template <class TBlockAttributes>

void HeapInfo::ValidPointersMap<TBlockAttributes>::GenerateValidPointersMap(ValidPointersMapTable& validTable, InvalidBitsTable& invalidTable, BlockInfoMapTable& blockInfoTable)

{

// Create the valid pointer map to be shared by the buckets.

// Also create the invalid objects bit vector.

ushort \* buffer = &validTable[0][0];

memset(buffer, -1, sizeof(ushort)\* 2 \* TBlockAttributes::MaxSmallObjectCount \* TBlockAttributes::BucketCount);

for (uint i = 0; i < TBlockAttributes::BucketCount; i++)

{

// Non-interior first

ushort \* validPointers = buffer;

buffer += TBlockAttributes::MaxSmallObjectCount;

SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector \* invalidBitVector = &invalidTable[i];

invalidBitVector->SetAll();

uint bucketSize;

if (TBlockAttributes::IsSmallBlock)

{

bucketSize = TBlockAttributes::MinObjectSize + HeapConstants::ObjectGranularity \* i;

}

else

{

bucketSize = TBlockAttributes::MinObjectSize + HeapConstants::MediumObjectGranularity \* (i + 1);

}

uint stride = bucketSize / HeapConstants::ObjectGranularity;

uint maxObjectCountForBucket = ((TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / bucketSize);

BlockInfoMapRow\* blockInfoRow = &blockInfoTable[i];

memset(blockInfoRow, 0, sizeof(BlockInfoMapRow));

for (ushort j = 0; j < maxObjectCountForBucket; j++)

{

validPointers[j \* stride] = j;

uintptr\_t objectAddress = j \* bucketSize;

Assert(objectAddress / AutoSystemInfo::PageSize < USHRT\_MAX);

ushort pageIndex = (ushort)(objectAddress / AutoSystemInfo::PageSize);

(\*blockInfoRow)[pageIndex].pageObjectCount++;

(\*blockInfoRow)[pageIndex].lastObjectIndexOnPage = max(j, (\*blockInfoRow)[pageIndex].lastObjectIndexOnPage);

invalidBitVector->Clear(j \* stride);

}

// interior pointer

ushort \* validInteriorPointers = buffer;

buffer += TBlockAttributes::MaxSmallObjectCount;

for (ushort j = 0; j < maxObjectCountForBucket; j++)

{

uint start = j \* stride;

uint end = min(start + stride, TBlockAttributes::MaxSmallObjectCount);

for (uint k = start; k < end; k++)

{

validInteriorPointers[k] = j;

}

}

}

}

template <>

HRESULT HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::GenerateValidPointersMapForBlockType(FILE\* file)

{

#define IfErrorGotoCleanup(result) if ((result) < 0) { hr = E\_FAIL; goto cleanup; }

Assert(file != nullptr);

HRESULT hr = S\_OK;

// Use heap to allocate the table so we don't bloat the stack (~64k). We only use this function

// to generate headers as part of testing.

ValidPointersMapTable \*valid = (ValidPointersMapTable \*)malloc(sizeof(ValidPointersMapTable));

InvalidBitsTable \*invalid = (InvalidBitsTable \*)malloc(sizeof(InvalidBitsTable));

BlockInfoMapTable \*blockMap = (BlockInfoMapTable\*)malloc(sizeof(BlockInfoMapTable));

if (valid == nullptr || invalid == nullptr || blockMap == nullptr)

{

hr = E\_FAIL;

goto cleanup;

}

GenerateValidPointersMap(\*valid, \*invalid, \*blockMap);

IfErrorGotoCleanup(fwprintf(file, L"const ushort HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::validPointersBuffer[HeapConstants::BucketCount][HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::rowSize] = \n{\n"));

// Generate the full buffer.

for (unsigned i = 0; i < HeapConstants::BucketCount; ++i)

{

IfErrorGotoCleanup(fwprintf(file, L" {\n "));

for (unsigned j = 0; j < rowSize; ++j)

{

IfErrorGotoCleanup(fwprintf(

file,

(j < rowSize - 1) ? L"0x%04hX, " : L"0x%04hX",

(\*valid)[i][j]));

}

IfErrorGotoCleanup(fwprintf(file, (i < HeapConstants::BucketCount - 1 ? L"\n },\n" : L"\n }\n")));

}

IfErrorGotoCleanup(fwprintf(file, L"};\n"));

// Generate the invalid bitvectors.

IfErrorGotoCleanup(fwprintf(

file,

L"const BVUnit HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::invalidBitsData[HeapConstants::BucketCount][SmallHeapBlockT<SmallAllocationBlockAttributes>::SmallHeapBlockBitVector::wordCount] = {\n"));

for (unsigned i = 0; i < HeapConstants::BucketCount; ++i)

{

IfErrorGotoCleanup(fwprintf(file, L" {\n "));

for (unsigned j = 0; j < (\*invalid)[i].wordCount; ++j)

{

const wchar\_t \*format = (j < (\*invalid)[i].wordCount - 1) ?

#if defined(\_M\_IX86\_OR\_ARM32)

L"0x%08X, " : L"0x%08X"

#elif defined(\_M\_X64\_OR\_ARM64)

L"0x%016I64X, " : L"0x%016I64X"

#else

#error "Platform is not handled"

#endif

;

IfErrorGotoCleanup(fwprintf(file, format, (\*invalid)[i].GetRawData()[j]));

}

IfErrorGotoCleanup(fwprintf(file, (i < HeapConstants::BucketCount - 1 ? L"\n },\n" : L"\n }\n")));

}

IfErrorGotoCleanup(fwprintf(

file,

L"};\n"

L"// The following is used to construct the InvalidBitsTable statically without forcing BVStatic to be an aggregate\n"

L"const HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::InvalidBitsTable \* const HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::invalidBitsBuffers =\n"

L" reinterpret\_cast<const HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::InvalidBitsTable \*>(&HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::invalidBitsData);\n"));

// Generate the block map table

IfErrorGotoCleanup(fwprintf(

file,

L"const SmallHeapBlockT<SmallAllocationBlockAttributes>::BlockInfo HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::blockInfoBuffer[SmallAllocationBlockAttributes::BucketCount][SmallAllocationBlockAttributes::PageCount] = {\n"));

for (unsigned i = 0; i < HeapConstants::BucketCount; ++i)

{

IfErrorGotoCleanup(fwprintf(file, L" // Bucket: %u, Size: %d\n", i, (int) (HeapConstants::ObjectGranularity + (i \* SmallAllocationBlockAttributes::BucketGranularity))));

IfErrorGotoCleanup(fwprintf(file, L" {\n"));

for (unsigned j = 0; j < SmallAllocationBlockAttributes::PageCount; ++j)

{

IfErrorGotoCleanup(fwprintf(file, L" { "));

const wchar\_t \*format = L"0x%04hX, 0x%04hX";

IfErrorGotoCleanup(fwprintf(file, format, (\*blockMap)[i][j].lastObjectIndexOnPage, (\*blockMap)[i][j].pageObjectCount));

IfErrorGotoCleanup(fwprintf(file, (j < SmallAllocationBlockAttributes::PageCount - 1 ? L" },\n" : L" }\n")));

}

IfErrorGotoCleanup(fwprintf(file, (i < HeapConstants::BucketCount - 1 ? L"\n },\n" : L"\n }\n")));

}

IfErrorGotoCleanup(fwprintf(file, L"};\n"));

cleanup:

#undef IfErrorGotoCleanup

free(valid);

free(invalid);

return hr;

}

template <>

HRESULT HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::GenerateValidPointersMapForBlockType(FILE\* file)

{

#define IfErrorGotoCleanup(result) if ((result) < 0) { hr = E\_FAIL; goto cleanup; }

Assert(file != nullptr);

HRESULT hr = S\_OK;

// Use heap to allocate the table so we don't bloat the stack (~64k). We only use this function

// to generate headers as part of testing.

ValidPointersMapTable \*valid = (ValidPointersMapTable \*)malloc(sizeof(ValidPointersMapTable));

InvalidBitsTable \*invalid = (InvalidBitsTable \*)malloc(sizeof(InvalidBitsTable));

BlockInfoMapTable \*blockMap = (BlockInfoMapTable \*)malloc(sizeof(BlockInfoMapTable));

if (valid == nullptr || invalid == nullptr || blockMap == nullptr)

{

hr = E\_FAIL;

goto cleanup;

}

GenerateValidPointersMap(\*valid, \*invalid, \*blockMap);

IfErrorGotoCleanup(fwprintf(file, L"const ushort HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::validPointersBuffer[MediumAllocationBlockAttributes::BucketCount][HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::rowSize] = \n{\n"));

// Generate the full buffer.

for (unsigned i = 0; i < HeapConstants::MediumBucketCount; ++i)

{

IfErrorGotoCleanup(fwprintf(file, L" {\n "));

for (unsigned j = 0; j < rowSize; ++j)

{

IfErrorGotoCleanup(fwprintf(

file,

(j < rowSize - 1) ? L"0x%04hX, " : L"0x%04hX",

(\*valid)[i][j]));

}

IfErrorGotoCleanup(fwprintf(file, (i < HeapConstants::MediumBucketCount - 1 ? L"\n },\n" : L"\n }\n")));

}

IfErrorGotoCleanup(fwprintf(file, L"};\n"));

// Generate the invalid bitvectors.

IfErrorGotoCleanup(fwprintf(

file,

L"const BVUnit HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::invalidBitsData[MediumAllocationBlockAttributes::BucketCount][SmallHeapBlockT<MediumAllocationBlockAttributes>::SmallHeapBlockBitVector::wordCount] = {\n"));

for (unsigned i = 0; i < HeapConstants::MediumBucketCount; ++i)

{

IfErrorGotoCleanup(fwprintf(file, L" {\n "));

for (unsigned j = 0; j < (\*invalid)[i].wordCount; ++j)

{

const wchar\_t \*format = (j < (\*invalid)[i].wordCount - 1) ?

#if defined(\_M\_IX86\_OR\_ARM32)

L"0x%08X, " : L"0x%08X"

#elif defined(\_M\_X64\_OR\_ARM64)

L"0x%016I64X, " : L"0x%016I64X"

#else

#error "Platform is not handled"

#endif

;

IfErrorGotoCleanup(fwprintf(file, format, (\*invalid)[i].GetRawData()[j]));

}

IfErrorGotoCleanup(fwprintf(file, (i < HeapConstants::MediumBucketCount - 1 ? L"\n },\n" : L"\n }\n")));

}

IfErrorGotoCleanup(fwprintf(

file,

L"};\n"

L"// The following is used to construct the InvalidBitsTable statically without forcing BVStatic to be an aggregate\n"

L"const HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::InvalidBitsTable \* const HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::invalidBitsBuffers =\n"

L" reinterpret\_cast<const HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::InvalidBitsTable \*>(&HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::invalidBitsData);\n"));

// Generate the block map table

IfErrorGotoCleanup(fwprintf(

file,

L"const SmallHeapBlockT<MediumAllocationBlockAttributes>::BlockInfo HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::blockInfoBuffer[MediumAllocationBlockAttributes::BucketCount][MediumAllocationBlockAttributes::PageCount] = {\n"));

for (unsigned i = 0; i < HeapConstants::MediumBucketCount; ++i)

{

IfErrorGotoCleanup(fwprintf(file, L" // Bucket: %u, Size: %d\n", i, (int)(HeapConstants::MaxSmallObjectSize + ((i + 1) \* MediumAllocationBlockAttributes::BucketGranularity))));

IfErrorGotoCleanup(fwprintf(file, L" {\n"));

for (unsigned j = 0; j < MediumAllocationBlockAttributes::PageCount; ++j)

{

IfErrorGotoCleanup(fwprintf(file, L" { "));

const wchar\_t \*format = L"0x%04hX, 0x%04hX";

IfErrorGotoCleanup(fwprintf(file, format, (\*blockMap)[i][j].lastObjectIndexOnPage, (\*blockMap)[i][j].pageObjectCount));

IfErrorGotoCleanup(fwprintf(file, (j < MediumAllocationBlockAttributes::PageCount - 1 ? L" },\n" : L" }\n")));

}

IfErrorGotoCleanup(fwprintf(file, (i < HeapConstants::MediumBucketCount - 1 ? L"\n },\n" : L"\n }\n")));

}

IfErrorGotoCleanup(fwprintf(file, L"};\n"));

cleanup:

#undef IfErrorGotoCleanup

free(valid);

free(invalid);

return hr;

}

template <class TBlockAttributes>

HRESULT HeapInfo::ValidPointersMap<TBlockAttributes>::GenerateValidPointersMapHeader(LPCWSTR vpmFullPath)

{

Assert(vpmFullPath != nullptr);

HRESULT hr = E\_FAIL;

FILE \* file = nullptr;

if (\_wfopen\_s(&file, vpmFullPath, L"w") == 0 && file != nullptr)

{

const wchar\_t \* header =

L"//-------------------------------------------------------------------------------------------------------\n"

L"// Copyright (C) Microsoft. All rights reserved.\n"

L"// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.\n"

L"//-------------------------------------------------------------------------------------------------------\n"

L"// Generated via jshost -GenerateValidPointersMapHeader\n"

#if defined(\_M\_IX86\_OR\_ARM32)

L"// Target platforms: 32bit - x86 & arm\n"

#elif defined(\_M\_X64\_OR\_ARM64)

L"// Target platform: 64bit - amd64 & arm64\n"

#else

#error "Platform is not handled"

#endif

L"#if USE\_STATIC\_VPM\n"

L"\n";

if (fwprintf(file, header) >= 0)

{

hr = ValidPointersMap<SmallAllocationBlockAttributes>::GenerateValidPointersMapForBlockType(file);

if (SUCCEEDED(hr))

{

hr = ValidPointersMap<MediumAllocationBlockAttributes>::GenerateValidPointersMapForBlockType(file);

}

fwprintf(file, L"#endif // USE\_STATIC\_VPM\n");

}

fclose(file);

}

return hr;

}

HeapInfo::HeapInfo() :

recycler(nullptr),

#ifdef CONCURRENT\_GC\_ENABLED

newLeafHeapBlockList(nullptr),

newNormalHeapBlockList(nullptr),

#ifdef RECYCLER\_WRITE\_BARRIER

newNormalWithBarrierHeapBlockList(nullptr),

newFinalizableWithBarrierHeapBlockList(nullptr),

#endif

newFinalizableHeapBlockList(nullptr),

newMediumLeafHeapBlockList(nullptr),

newMediumNormalHeapBlockList(nullptr),

#ifdef RECYCLER\_WRITE\_BARRIER

newMediumNormalWithBarrierHeapBlockList(nullptr),

newMediumFinalizableWithBarrierHeapBlockList(nullptr),

#endif

newMediumFinalizableHeapBlockList(nullptr),

#endif

#ifdef RECYCLER\_FINALIZE\_CHECK

liveFinalizableObjectCount(0),

pendingDisposableObjectCount(0),

newFinalizableObjectCount(0),

#endif

#ifdef PARTIAL\_GC\_ENABLED

uncollectedNewPageCount(0),

unusedPartialCollectFreeBytes(0),

#endif

uncollectedAllocBytes(0),

lastUncollectedAllocBytes(0),

pendingZeroPageCount(0)

#ifdef RECYCLER\_PAGE\_HEAP

, pageHeapMode(PageHeapMode::PageHeapModeOff)

, isPageHeapEnabled(false)

, pageHeapBlockType(PageHeapBlockTypeFilter::PageHeapBlockTypeFilterAll)

, captureAllocCallStack(false)

, captureFreeCallStack(false)

#endif

{

}

HeapInfo::~HeapInfo()

{

RECYCLER\_SLOW\_CHECK(this->VerifySmallHeapBlockCount());

// Finalize all finalizable object first

for (uint i=0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].FinalizeAllObjects();

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i=0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].FinalizeAllObjects();

}

#endif

largeObjectBucket.FinalizeAllObjects();

SmallFinalizableHeapBucket::FinalizeHeapBlockList(this->newFinalizableHeapBlockList);

MediumFinalizableHeapBucket::FinalizeHeapBlockList(this->newMediumFinalizableHeapBlockList);

#ifdef RECYCLER\_WRITE\_BARRIER

SmallFinalizableWithBarrierHeapBucket::FinalizeHeapBlockList(this->newFinalizableWithBarrierHeapBlockList);

MediumFinalizableWithBarrierHeapBucket::FinalizeHeapBlockList(this->newMediumFinalizableWithBarrierHeapBlockList);

#endif

#ifdef RECYCLER\_FINALIZE\_CHECK

Assert(liveFinalizableObjectCount == 0);

Assert(pendingDisposableObjectCount == 0);

#endif

// Delete the heap blocks

Recycler \* recycler = this->recycler;

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t largeBlockCount = this->largeObjectBucket.GetLargeHeapBlockCount(false);

uint mediumBlockCount = 0;

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && !SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumBlockCount += mediumHeapBuckets[i].GetLargeHeapBlockCount(false);

}

#endif

#endif

RECYCLER\_SLOW\_CHECK(Assert(this->heapBlockCount[HeapBlock::HeapBlockType::LargeBlockType] - largeBlockCount - mediumBlockCount == 0));

SmallLeafHeapBucket::DeleteHeapBlockList(this->newLeafHeapBlockList, recycler);

SmallNormalHeapBucket::DeleteHeapBlockList(this->newNormalHeapBlockList, recycler);

#ifdef RECYCLER\_WRITE\_BARRIER

SmallNormalWithBarrierHeapBucket::DeleteHeapBlockList(this->newNormalWithBarrierHeapBlockList, recycler);

SmallFinalizableWithBarrierHeapBucket::DeleteHeapBlockList(this->newFinalizableWithBarrierHeapBlockList, recycler);

#endif

SmallFinalizableHeapBucket::DeleteHeapBlockList(this->newFinalizableHeapBlockList, recycler);

MediumLeafHeapBucket::DeleteHeapBlockList(this->newMediumLeafHeapBlockList, recycler);

MediumNormalHeapBucket::DeleteHeapBlockList(this->newMediumNormalHeapBlockList, recycler);

#ifdef RECYCLER\_WRITE\_BARRIER

MediumNormalWithBarrierHeapBucket::DeleteHeapBlockList(this->newMediumNormalWithBarrierHeapBlockList, recycler);

MediumFinalizableWithBarrierHeapBucket::DeleteHeapBlockList(this->newMediumFinalizableWithBarrierHeapBlockList, recycler);

#endif

MediumFinalizableHeapBucket::DeleteHeapBlockList(this->newMediumFinalizableHeapBlockList, recycler);

// We do this here, instead of in the Recycler destructor, because the above stuff may

// generate additional tracking events, particularly ReportUnallocated.

// Arguably we shouldn't report these things as ReportUnallocated...

RecyclerMemoryTracking::ReportRecyclerDestroy(recycler);

}

void

HeapInfo::Initialize(Recycler \* recycler

#ifdef RECYCLER\_PAGE\_HEAP

, PageHeapMode pageheapmode

, bool captureAllocCallStack

, bool captureFreeCallStack

#endif

)

{

this->recycler = recycler;

#ifdef DUMP\_FRAGMENTATION\_STATS

if (recycler->GetRecyclerFlagsTable().flags.DumpFragmentationStats)

{

printf("[FRAG %d] Start", ::GetTickCount());

}

#endif

#ifdef RECYCLER\_PAGE\_HEAP

isPageHeapEnabled = false;

PageHeapBlockTypeFilter blockTypeFilter = PageHeapBlockTypeFilter::PageHeapBlockTypeFilterAll;

Js::NumberRange bucketNumberRange;

Js::NumberRange\* pBucketNumberRange = &bucketNumberRange;

if (pageheapmode == PageHeapMode::PageHeapModeOff)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

isPageHeapEnabled = Js::Configuration::Global.flags.PageHeap != PageHeapMode::PageHeapModeOff;

pageheapmode = (PageHeapMode)Js::Configuration::Global.flags.PageHeap;

blockTypeFilter = (PageHeapBlockTypeFilter)Js::Configuration::Global.flags.PageHeapBlockType;

pBucketNumberRange = &Js::Configuration::Global.flags.PageHeapBucketNumber;

#else

// @TODO in free build, use environment var or other way to enable page heap

// currently page heap build is enable in free build but has not implemented a way to input the page heap flags.

// if we only need page heap in free test build, just move RECYCLER\_PAGE\_HEAP definition into ENABLE\_DEBUG\_CONFIG\_OPTIONS

// in CommonDefines.h it should work

#endif

}

else

{

isPageHeapEnabled = true;

}

#ifdef RECYCLER\_PAGE\_HEAP

if (isPageHeapEnabled)

{

this->captureAllocCallStack = captureAllocCallStack;

this->captureFreeCallStack = captureFreeCallStack;

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

this->captureAllocCallStack = captureAllocCallStack || Js::Configuration::Global.flags.PageHeapAllocStack;

this->captureFreeCallStack = captureFreeCallStack || Js::Configuration::Global.flags.PageHeapFreeStack;

#endif

}

#endif

if (IsPageHeapEnabled())

{

this->pageHeapMode = pageheapmode;

// Use one of the two modes with -PageHeap flag

Assert(this->pageHeapMode == PageHeapMode::PageHeapModeBlockStart || this->pageHeapMode == PageHeapMode::PageHeapModeBlockEnd);

this->pageHeapBlockType = blockTypeFilter;

for (int i = 0; i < HeapConstants::BucketCount + HeapConstants::MediumBucketCount; i++)

{

if (pBucketNumberRange->InRange(i))

{

if (i < HeapConstants::BucketCount)

{

this->smallBlockPageHeapBucketFilter.Set(i);

}

else

{

this->mediumBlockPageHeapBucketFilter.Set(i - HeapConstants::BucketCount);

}

}

}

}

else

{

// These should not be set if we're not in page heap mode

Assert(!(captureAllocCallStack || captureFreeCallStack));

}

#endif

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].Initialize(this, (i + 1) << HeapConstants::ObjectAllocationShift);

}

RECYCLER\_SLOW\_CHECK(memset(this->heapBlockCount, 0, sizeof(this->heapBlockCount)));

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

#if SMALLBLOCK\_MEDIUM\_ALLOC

mediumHeapBuckets[i].Initialize(this, HeapConstants::MaxSmallObjectSize + ((i + 1) \* HeapConstants::MediumObjectGranularity));

#else

mediumHeapBuckets[i].Initialize(this, HeapConstants::MaxSmallObjectSize + ((i + 1) \* HeapConstants::MediumObjectGranularity), true);

#endif

}

#endif

largeObjectBucket.Initialize(this, HeapConstants::MaxMediumObjectSize);

}

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(RECYCLER\_MEMORY\_VERIFY) || defined(MEMSPECT\_TRACKING) || defined(ETW\_MEMORY\_TRACKING)

void

HeapInfo::Initialize(Recycler \* recycler, void(\*trackNativeAllocCallBack)(Recycler \*, void \*, size\_t)

#ifdef RECYCLER\_PAGE\_HEAP

, PageHeapMode pageheapmode

, bool captureAllocCallStack

, bool captureFreeCallStack

#endif

)

{

Initialize(recycler

#ifdef RECYCLER\_PAGE\_HEAP

, pageheapmode

, captureAllocCallStack

, captureFreeCallStack

#endif

);

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].GetBucket<NoBit>().GetAllocator()->SetTrackNativeAllocatedObjectCallBack(trackNativeAllocCallBack);

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].GetBucket<NoBit>().GetAllocator()->SetTrackNativeAllocatedObjectCallBack(trackNativeAllocCallBack);

}

#endif

}

#endif

#ifdef RECYCLER\_PAGE\_HEAP

template bool HeapInfo::IsPageHeapEnabledForBlock<MediumAllocationBlockAttributes>(const size\_t objectSize);

template bool HeapInfo::IsPageHeapEnabledForBlock<SmallAllocationBlockAttributes>(const size\_t objectSize);

template bool HeapInfo::IsPageHeapEnabledForBlock<LargeAllocationBlockAttributes>(const size\_t objectSize);

template <typename TBlockAttributes>

bool HeapInfo::IsPageHeapEnabledForBlock(const size\_t objectSize)

{

if (IsPageHeapEnabled())

{

if (TBlockAttributes::IsSmallBlock)

{

return smallBlockPageHeapBucketFilter.Test(GetBucketIndex(objectSize)) != 0;

}

else if (TBlockAttributes::IsMediumBlock)

{

return mediumBlockPageHeapBucketFilter.Test(GetMediumBucketIndex(objectSize)) != 0;

}

else

{

// Page heap is enabled for large heap by default if page heap mode is on

return true;

}

}

return false;

}

#endif

void

HeapInfo::ResetMarks(ResetMarkFlags flags)

{

for (uint i=0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].ResetMarks(flags);

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i=0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].ResetMarks(flags);

}

#endif

largeObjectBucket.ResetMarks(flags);

#ifdef CONCURRENT\_GC\_ENABLED

if ((flags & ResetMarkFlags\_ScanImplicitRoot) != 0)

{

HeapBlockList::ForEach(newLeafHeapBlockList, [flags](SmallLeafHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

HeapBlockList::ForEach(newNormalHeapBlockList, [flags](SmallNormalHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newNormalWithBarrierHeapBlockList, [flags](SmallNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

HeapBlockList::ForEach(newFinalizableWithBarrierHeapBlockList, [flags](SmallFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

#endif

HeapBlockList::ForEach(newFinalizableHeapBlockList, [flags](SmallNormalHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

HeapBlockList::ForEach(newMediumLeafHeapBlockList, [flags](MediumLeafHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

HeapBlockList::ForEach(newMediumNormalHeapBlockList, [flags](MediumNormalHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newMediumNormalWithBarrierHeapBlockList, [flags](MediumNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

HeapBlockList::ForEach(newMediumFinalizableWithBarrierHeapBlockList, [flags](MediumFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

#endif

HeapBlockList::ForEach(newMediumFinalizableHeapBlockList, [flags](MediumNormalHeapBlock \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

}

#endif

}

void

HeapInfo::ScanInitialImplicitRoots()

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].ScanInitialImplicitRoots(recycler);

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].ScanInitialImplicitRoots(recycler);

}

#endif

largeObjectBucket.ScanInitialImplicitRoots(recycler);

#ifdef CONCURRENT\_GC\_ENABLED

// NOTE: Don't need to do newLeafHeapBlockList

HeapBlockList::ForEach(newNormalHeapBlockList, [this](SmallNormalHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newNormalWithBarrierHeapBlockList, [this](SmallNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

HeapBlockList::ForEach(newFinalizableWithBarrierHeapBlockList, [this](SmallFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#endif

HeapBlockList::ForEach(newFinalizableHeapBlockList, [this](SmallNormalHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#endif

#ifdef CONCURRENT\_GC\_ENABLED

// NOTE: Don't need to do newLeafHeapBlockList

HeapBlockList::ForEach(newMediumNormalHeapBlockList, [this](MediumNormalHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newMediumNormalWithBarrierHeapBlockList, [this](MediumNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

HeapBlockList::ForEach(newMediumFinalizableWithBarrierHeapBlockList, [this](MediumFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#endif

HeapBlockList::ForEach(newMediumFinalizableHeapBlockList, [this](MediumNormalHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#endif

}

void

HeapInfo::ScanNewImplicitRoots()

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].ScanNewImplicitRoots(recycler);

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].ScanNewImplicitRoots(recycler);

}

#endif

largeObjectBucket.ScanNewImplicitRoots(recycler);

#ifdef CONCURRENT\_GC\_ENABLED

// NOTE: need to do newLeafHeapBlockList to find new memory

HeapBlockList::ForEach(newLeafHeapBlockList, [this](SmallLeafHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

HeapBlockList::ForEach(newNormalHeapBlockList, [this](SmallNormalHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newNormalWithBarrierHeapBlockList, [this](SmallNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

HeapBlockList::ForEach(newFinalizableWithBarrierHeapBlockList, [this](SmallFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

#endif

HeapBlockList::ForEach(newFinalizableHeapBlockList, [this](SmallNormalHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

// NOTE: need to do newLeafHeapBlockList to find new memory

HeapBlockList::ForEach(newMediumLeafHeapBlockList, [this](MediumLeafHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

HeapBlockList::ForEach(newMediumNormalHeapBlockList, [this](MediumNormalHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newMediumNormalWithBarrierHeapBlockList, [this](MediumNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

HeapBlockList::ForEach(newMediumFinalizableWithBarrierHeapBlockList, [this](MediumFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

#endif

HeapBlockList::ForEach(newMediumFinalizableHeapBlockList, [this](MediumNormalHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

#endif

}

LargeHeapBlock \*

HeapInfo::AddLargeHeapBlock(size\_t size)

{

// Do a no-throwing allocation here

return largeObjectBucket.AddLargeHeapBlock(size, /\* nothrow = \*/ true);

}

template<bool pageheap>

void HeapInfo::Sweep(RecyclerSweep& recyclerSweep, bool concurrent)

{

Recycler \* recycler = recyclerSweep.GetRecycler();

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].SweepFinalizableObjects<pageheap>(recyclerSweep);

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

// CONCURRENT-TODO: Allow this in the background as well

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].SweepFinalizableObjects<pageheap>(recyclerSweep);

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

if (concurrent)

{

RECYCLER\_SLOW\_CHECK(VerifySmallHeapBlockCount());

RECYCLER\_SLOW\_CHECK(VerifyLargeHeapBlockCount());

}

if (concurrent)

{

this->SetupBackgroundSweep(recyclerSweep);

}

else

#endif

{

this->SweepSmallNonFinalizable<pageheap>(recyclerSweep);

}

RECYCLER\_PROFILE\_EXEC\_CHANGE(recycler, Js::SweepSmallPhase, Js::SweepLargePhase);

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && !(SMALLBLOCK\_MEDIUM\_ALLOC)

// CONCURRENT-TODO: Allow this in the background as well

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].Sweep<pageheap>(recyclerSweep);

}

#endif

largeObjectBucket.Sweep<pageheap>(recyclerSweep);

}

void

HeapInfo::Sweep(RecyclerSweep& recyclerSweep, bool concurrent)

{

#ifdef RECYCLER\_FINALIZE\_CHECK

this->newFinalizableObjectCount = 0;

#endif

// Initialize this to false. Individual heap buckets can set it to true

Recycler \* recycler = recyclerSweep.GetRecycler();

RECYCLER\_PROFILE\_EXEC\_BEGIN(recycler, Js::SweepSmallPhase);

#ifdef RECYCLER\_STATS

memset(&recycler->collectionStats.numEmptySmallBlocks, 0, sizeof(recycler->collectionStats.numEmptySmallBlocks));

recycler->collectionStats.numZeroedOutSmallBlocks = 0;

#endif

RECYCLER\_SLOW\_CHECK(VerifySmallHeapBlockCount());

// Call finalize before sweeping so that the finalizer can still access object it referenced

largeObjectBucket.Finalize();

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && !(SMALLBLOCK\_MEDIUM\_ALLOC)

for (uint i=0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].Finalize();

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

// Merge the new blocks before we sweep the finalizable object in thread

recyclerSweep.MergePendingNewHeapBlockList<SmallFinalizableHeapBlock>();

recyclerSweep.MergePendingNewMediumHeapBlockList<MediumFinalizableHeapBlock>();

#ifdef RECYCLER\_WRITE\_BARRIER

recyclerSweep.MergePendingNewHeapBlockList<SmallFinalizableWithBarrierHeapBlock>();

recyclerSweep.MergePendingNewMediumHeapBlockList<MediumFinalizableWithBarrierHeapBlock>();

#endif

#endif

if (IsPageHeapEnabled())

{

Sweep<true>(recyclerSweep, concurrent);

}

else

{

Sweep<false>(recyclerSweep, concurrent);

}

RECYCLER\_PROFILE\_EXEC\_END(recycler, Js::SweepLargePhase);

RECYCLER\_SLOW\_CHECK(VerifyLargeHeapBlockCount());

RECYCLER\_SLOW\_CHECK(Assert(this->newFinalizableObjectCount == 0));

}

#ifdef CONCURRENT\_GC\_ENABLED

void

HeapInfo::SetupBackgroundSweep(RecyclerSweep& recyclerSweep)

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

this->heapBuckets[i].SetupBackgroundSweep(recyclerSweep);

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

this->mediumHeapBuckets[i].SetupBackgroundSweep(recyclerSweep);

}

#endif

}

#endif

template<bool pageheap>

void

HeapInfo::SweepSmallNonFinalizable(RecyclerSweep& recyclerSweep)

{

#ifdef CONCURRENT\_GC\_ENABLED

recyclerSweep.MergePendingNewHeapBlockList<SmallLeafHeapBlock>();

recyclerSweep.MergePendingNewHeapBlockList<SmallNormalHeapBlock>();

recyclerSweep.MergePendingNewMediumHeapBlockList<MediumLeafHeapBlock>();

recyclerSweep.MergePendingNewMediumHeapBlockList<MediumNormalHeapBlock>();

#ifdef RECYCLER\_WRITE\_BARRIER

recyclerSweep.MergePendingNewHeapBlockList<SmallNormalWithBarrierHeapBlock>();

recyclerSweep.MergePendingNewMediumHeapBlockList<MediumNormalWithBarrierHeapBlock>();

#endif

// Finalizable are already merge before in SweepHeap

Assert(!recyclerSweep.HasPendingNewHeapBlocks());

#endif

if (!recyclerSweep.IsBackground())

{

// finalizer may trigger arena allocations, do don't suspend the leaf (thread) page allocator

// until we are going to sweep leaf pages.

recycler->GetRecyclerLeafPageAllocator()->SuspendIdleDecommit();

}

for (uint i=0; i<HeapConstants::BucketCount; i++)

{

heapBuckets[i].Sweep<pageheap>(recyclerSweep);

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].Sweep<pageheap>(recyclerSweep);

}

#endif

if (!recyclerSweep.IsBackground())

{

// large block don't use the leaf page allocator, we can resume idle decommit now

recycler->GetRecyclerLeafPageAllocator()->ResumeIdleDecommit();

RECYCLER\_SLOW\_CHECK(VerifySmallHeapBlockCount());

RECYCLER\_SLOW\_CHECK(VerifyLargeHeapBlockCount());

}

}

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t

HeapInfo::Rescan(RescanFlags flags)

{

size\_t scannedPageCount = 0;

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

scannedPageCount += heapBuckets[i].Rescan(recycler, flags);

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

#if SMALLBLOCK\_MEDIUM\_ALLOC

scannedPageCount += mediumHeapBuckets[i].Rescan(recycler, flags);

#else

scannedPageCount += mediumHeapBuckets[i].Rescan(flags);

#endif

}

#endif

scannedPageCount += largeObjectBucket.Rescan(flags);

return scannedPageCount;

}

#endif

template <ObjectInfoBits TBucketType, class TBlockAttributes>

void DumpBucket(uint bucketIndex, typename SmallHeapBlockType<TBucketType, TBlockAttributes>::BucketType& bucket)

{

HeapBucketStats stats = { 0 };

bucket.AggregateBucketStats(stats);

Output::Print(L"%d,%d,", bucketIndex, (bucketIndex + 1) << HeapConstants::ObjectAllocationShift);

Output::Print(L"%d,%d,%d,%d,%d,%d,%d\n", stats.totalBlockCount, stats.finalizeBlockCount, stats.emptyBlockCount, stats.objectCount, stats.finalizeCount, stats.objectByteCount, stats.totalByteCount);

}

#ifdef DUMP\_FRAGMENTATION\_STATS

void

HeapInfo::DumpFragmentationStats()

{

Output::Print(L"[FRAG %d] Post-Collection State\n", ::GetTickCount());

Output::Print(L"Bucket,SizeCat,Block Count,Finalizable Block Count,Empty Block Count, Object Count, Finalizable Object Count, Object size, Block Size\n");

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

DumpBucket<NoBit, SmallAllocationBlockAttributes>(i, heapBuckets[i].GetBucket<NoBit>());

DumpBucket<FinalizeBit, SmallAllocationBlockAttributes>(i, heapBuckets[i].GetBucket<FinalizeBit>());

DumpBucket<LeafBit, SmallAllocationBlockAttributes>(i, heapBuckets[i].GetBucket<LeafBit>());

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

DumpBucket<NoBit, MediumAllocationBlockAttributes>(i, mediumHeapBuckets[i].GetBucket<NoBit>());

DumpBucket<FinalizeBit, MediumAllocationBlockAttributes>(i, mediumHeapBuckets[i].GetBucket<FinalizeBit>());

DumpBucket<LeafBit, MediumAllocationBlockAttributes>(i, mediumHeapBuckets[i].GetBucket<LeafBit>());

}

#endif

}

#endif

#ifdef PARTIAL\_GC\_ENABLED

void

HeapInfo::SweepPartialReusePages(RecyclerSweep& recyclerSweep)

{

RECYCLER\_PROFILE\_EXEC\_THREAD\_BEGIN(recyclerSweep.IsBackground(), recyclerSweep.GetRecycler(), Js::SweepPartialReusePhase);

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].SweepPartialReusePages(recyclerSweep);

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].SweepPartialReusePages(recyclerSweep);

}

#endif

RECYCLER\_PROFILE\_EXEC\_THREAD\_END(recyclerSweep.IsBackground(), recyclerSweep.GetRecycler(), Js::SweepPartialReusePhase);

// GC-TODO: LargeHeapBlock don't reuse object, so we don't need to keep

// pages with low free space from being reused.

// Only count the byte that we would have freed but we are not reusing it if we are doing a partial GC

// This will increase the GC pressure and make partial less and less likely.

if (recyclerSweep.InPartialCollect())

{

this->unusedPartialCollectFreeBytes += recyclerSweep.GetPartialUnusedFreeByteCount();

}

}

void HeapInfo::FinishPartialCollect(RecyclerSweep \* recyclerSweep)

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].FinishPartialCollect(recyclerSweep);

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].FinishPartialCollect(recyclerSweep);

}

#endif

largeObjectBucket.FinishPartialCollect(recyclerSweep);

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

void

HeapInfo::PrepareSweep()

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].PrepareSweep();

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].PrepareSweep();

}

#endif

}

#endif

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

void

HeapInfo::SweepPendingObjects(RecyclerSweep& recyclerSweep)

{

if (recyclerSweep.HasPendingSweepSmallHeapBlocks())

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].SweepPendingObjects(recyclerSweep);

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].SweepPendingObjects(recyclerSweep);

}

#endif

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && !SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].SweepPendingObjects(recyclerSweep);

}

#endif

largeObjectBucket.SweepPendingObjects(recyclerSweep);

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

void

HeapInfo::TransferPendingHeapBlocks(RecyclerSweep& recyclerSweep)

{

Assert(!recyclerSweep.IsBackground());

RECYCLER\_SLOW\_CHECK(VerifySmallHeapBlockCount());

if (recyclerSweep.HasPendingEmptyBlocks())

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].TransferPendingEmptyHeapBlocks(recyclerSweep);

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].TransferPendingEmptyHeapBlocks(recyclerSweep);

}

#endif

RECYCLER\_SLOW\_CHECK(VerifySmallHeapBlockCount());

}

// We might still have block that has been disposed but not made allocable

// which happens if we finish disposing object during concurrent sweep

// and can't modify the block lists

recyclerSweep.FlushPendingTransferDisposedObjects();

}

void

HeapInfo::ConcurrentTransferSweptObjects(RecyclerSweep& recyclerSweep)

{

Assert(!recyclerSweep.InPartialCollectMode());

Assert(!recyclerSweep.IsBackground());

TransferPendingHeapBlocks(recyclerSweep);

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && !SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].ConcurrentTransferSweptObjects(recyclerSweep);

}

#endif

largeObjectBucket.ConcurrentTransferSweptObjects(recyclerSweep);

}

#ifdef PARTIAL\_GC\_ENABLED

void

HeapInfo::ConcurrentPartialTransferSweptObjects(RecyclerSweep& recyclerSweep)

{

Assert(recyclerSweep.InPartialCollectMode());

Assert(!recyclerSweep.IsBackground());

TransferPendingHeapBlocks(recyclerSweep);

RECYCLER\_SLOW\_CHECK(this->VerifyLargeHeapBlockCount());

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && !SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].ConcurrentPartialTransferSweptObjects(recyclerSweep);

}

#endif

largeObjectBucket.ConcurrentPartialTransferSweptObjects(recyclerSweep);

RECYCLER\_SLOW\_CHECK(this->VerifyLargeHeapBlockCount());

}

#endif

#endif

void

HeapInfo::DisposeObjects()

{

Recycler \* recycler = this->recycler;

do

{

recycler->hasDisposableObject = false;

// finalizing the objects

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].DisposeObjects();

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].DisposeObjects();

}

#endif

largeObjectBucket.DisposeObjects();

}

// Calling dispose may enter the GC again and dispose more objects, loop until we don't have any more

while (recycler->hasDisposableObject);

recycler->hasPendingTransferDisposedObjects = true;

if (!recycler->IsConcurrentExecutingState())

{

// Can't transfer disposed object when the background thread is walking the heap block list

// That includes reset mark, background rescan and concurrent sweep. Delay the transfer later.

// NOTE1: During concurrent sweep, we can't do this only if the bucket has "stopped" allocation

// After it resume allocation, we don't walk the list in the background thread any more

// (except for checking heap block count). But this is easier to detect via the collection state

// without walking all buckets.

// NOTE2: During transitive closure mark, we don't walk the heap block list, but we can continue

// to do background rescan. Since we don't have synchronization for that, we can't really enable

// able this just for the transitive closure, so just do all the background executing state.

TransferDisposedObjects();

}

}

void

HeapInfo::TransferDisposedObjects()

{

Recycler \* recycler = this->recycler;

Assert(recycler->hasPendingTransferDisposedObjects);

Assert(!recycler->IsConcurrentExecutingState());

recycler->hasPendingTransferDisposedObjects = false;

// move the disposed object back to the free lists

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].TransferDisposedObjects();

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].TransferDisposedObjects();

}

#endif

largeObjectBucket.TransferDisposedObjects();

}

void

HeapInfo::EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size))

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].EnumerateObjects(infoBits, CallBackFunction);

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].EnumerateObjects(infoBits, CallBackFunction);

}

#endif

largeObjectBucket.EnumerateObjects(infoBits, CallBackFunction);

#ifdef CONCURRENT\_GC\_ENABLED

HeapBucket::EnumerateObjects(newLeafHeapBlockList, infoBits, CallBackFunction);

HeapBucket::EnumerateObjects(newNormalHeapBlockList, infoBits, CallBackFunction);

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBucket::EnumerateObjects(newNormalWithBarrierHeapBlockList, infoBits, CallBackFunction);

HeapBucket::EnumerateObjects(newFinalizableWithBarrierHeapBlockList, infoBits, CallBackFunction);

#endif

HeapBucket::EnumerateObjects(newFinalizableHeapBlockList, infoBits, CallBackFunction);

HeapBucket::EnumerateObjects(newMediumLeafHeapBlockList, infoBits, CallBackFunction);

HeapBucket::EnumerateObjects(newMediumNormalHeapBlockList, infoBits, CallBackFunction);

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBucket::EnumerateObjects(newMediumNormalWithBarrierHeapBlockList, infoBits, CallBackFunction);

HeapBucket::EnumerateObjects(newMediumFinalizableWithBarrierHeapBlockList, infoBits, CallBackFunction);

#endif

HeapBucket::EnumerateObjects(newMediumFinalizableHeapBlockList, infoBits, CallBackFunction);

#endif

}

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t

HeapInfo::GetSmallHeapBlockCount(bool checkCount) const

{

size\_t currentSmallHeapBlockCount = 0;

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

currentSmallHeapBlockCount += heapBuckets[i].GetNonEmptyHeapBlockCount(checkCount);

currentSmallHeapBlockCount += heapBuckets[i].GetEmptyHeapBlockCount();

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

currentSmallHeapBlockCount += mediumHeapBuckets[i].GetNonEmptyHeapBlockCount(checkCount);

currentSmallHeapBlockCount += mediumHeapBuckets[i].GetEmptyHeapBlockCount();

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

currentSmallHeapBlockCount += HeapBlockList::Count(this->newLeafHeapBlockList);

currentSmallHeapBlockCount += HeapBlockList::Count(this->newNormalHeapBlockList);

currentSmallHeapBlockCount += HeapBlockList::Count(this->newFinalizableHeapBlockList);

#ifdef RECYCLER\_WRITE\_BARRIER

currentSmallHeapBlockCount += HeapBlockList::Count(this->newNormalWithBarrierHeapBlockList);

currentSmallHeapBlockCount += HeapBlockList::Count(this->newFinalizableWithBarrierHeapBlockList);

#endif

currentSmallHeapBlockCount += HeapBlockList::Count(this->newMediumLeafHeapBlockList);

currentSmallHeapBlockCount += HeapBlockList::Count(this->newMediumNormalHeapBlockList);

currentSmallHeapBlockCount += HeapBlockList::Count(this->newMediumFinalizableHeapBlockList);

#ifdef RECYCLER\_WRITE\_BARRIER

currentSmallHeapBlockCount += HeapBlockList::Count(this->newMediumNormalWithBarrierHeapBlockList);

currentSmallHeapBlockCount += HeapBlockList::Count(this->newMediumFinalizableWithBarrierHeapBlockList);

#endif

// TODO: Update recycler sweep

// Recycler can be null if we have OOM in the ctor

if (this->recycler && this->recycler->recyclerSweep != nullptr)

{

// This function can't be called in the background

Assert(!this->recycler->recyclerSweep->IsBackground());

currentSmallHeapBlockCount += this->recycler->recyclerSweep->SetPendingMergeNewHeapBlockCount();

}

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t expectedHeapBlockCount =

this->heapBlockCount[HeapBlock::HeapBlockType::SmallNormalBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::SmallLeafBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::SmallFinalizableBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumNormalBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumLeafBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumFinalizableBlockType];

#ifdef RECYCLER\_WRITE\_BARRIER

expectedHeapBlockCount +=

this->heapBlockCount[HeapBlock::HeapBlockType::SmallNormalBlockWithBarrierType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::SmallFinalizableBlockWithBarrierType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumNormalBlockWithBarrierType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumFinalizableBlockWithBarrierType];

#endif

Assert(!checkCount || currentSmallHeapBlockCount == expectedHeapBlockCount);

#endif

return currentSmallHeapBlockCount;

}

size\_t

HeapInfo::GetLargeHeapBlockCount(bool checkCount) const

{

size\_t currentLargeHeapBlockCount = 0;

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && !SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

currentLargeHeapBlockCount += mediumHeapBuckets[i].GetLargeHeapBlockCount(checkCount);

}

#endif

currentLargeHeapBlockCount += largeObjectBucket.GetLargeHeapBlockCount(checkCount);

RECYCLER\_SLOW\_CHECK(Assert(!checkCount || currentLargeHeapBlockCount == this->heapBlockCount[HeapBlock::HeapBlockType::LargeBlockType]));

return currentLargeHeapBlockCount;

}

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

void

HeapInfo::Check()

{

Assert(!this->recycler->CollectionInProgress());

size\_t currentSmallHeapBlockCount = 0;

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

currentSmallHeapBlockCount += heapBuckets[i].Check();

currentSmallHeapBlockCount += heapBuckets[i].GetEmptyHeapBlockCount();

}

size\_t currentLargeHeapBlockCount = 0;

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

#if SMALLBLOCK\_MEDIUM\_ALLOC

currentSmallHeapBlockCount += mediumHeapBuckets[i].Check();

currentSmallHeapBlockCount += mediumHeapBuckets[i].GetEmptyHeapBlockCount();

#else

currentLargeHeapBlockCount += mediumHeapBuckets[i].Check();

#endif

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

currentSmallHeapBlockCount += Check(true, false, this->newLeafHeapBlockList);

currentSmallHeapBlockCount += Check(true, false, this->newNormalHeapBlockList);

#ifdef RECYCLER\_WRITE\_BARRIER

currentSmallHeapBlockCount += Check(true, false, this->newNormalWithBarrierHeapBlockList);

currentSmallHeapBlockCount += Check(true, false, this->newFinalizableWithBarrierHeapBlockList);

#endif

currentSmallHeapBlockCount += Check(true, false, this->newFinalizableHeapBlockList);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

currentSmallHeapBlockCount += Check(true, false, this->newMediumLeafHeapBlockList);

currentSmallHeapBlockCount += Check(true, false, this->newMediumNormalHeapBlockList);

#ifdef RECYCLER\_WRITE\_BARRIER

currentSmallHeapBlockCount += Check(true, false, this->newMediumNormalWithBarrierHeapBlockList);

currentSmallHeapBlockCount += Check(true, false, this->newMediumFinalizableWithBarrierHeapBlockList);

#endif

currentSmallHeapBlockCount += Check(true, false, this->newMediumFinalizableHeapBlockList);

#endif

size\_t expectedHeapBlockCount =

this->heapBlockCount[HeapBlock::HeapBlockType::SmallNormalBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::SmallLeafBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::SmallFinalizableBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumNormalBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumLeafBlockType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumFinalizableBlockType];

#ifdef RECYCLER\_WRITE\_BARRIER

expectedHeapBlockCount +=

this->heapBlockCount[HeapBlock::HeapBlockType::SmallNormalBlockWithBarrierType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::SmallFinalizableBlockWithBarrierType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumNormalBlockWithBarrierType]

+ this->heapBlockCount[HeapBlock::HeapBlockType::MediumFinalizableBlockWithBarrierType];

#endif

Assert(currentSmallHeapBlockCount == expectedHeapBlockCount);

currentLargeHeapBlockCount += largeObjectBucket.Check();

Assert(currentLargeHeapBlockCount == this->heapBlockCount[HeapBlock::HeapBlockType::LargeBlockType]);

}

template <typename TBlockType>

size\_t

HeapInfo::Check(bool expectFull, bool expectPending, TBlockType \* list, TBlockType \* tail)

{

size\_t heapBlockCount = 0;

HeapBlockList::ForEach(list, tail, [&heapBlockCount, expectFull, expectPending](TBlockType \* heapBlock)

{

heapBlock->Check(expectFull, expectPending);

heapBlockCount++;

});

return heapBlockCount;

}

template size\_t HeapInfo::Check<SmallNormalHeapBlock>(bool expectFull, bool expectPending, SmallNormalHeapBlock \* list, SmallNormalHeapBlock \* tail);

template size\_t HeapInfo::Check<SmallLeafHeapBlock>(bool expectFull, bool expectPending, SmallLeafHeapBlock \* list, SmallLeafHeapBlock \* tail);

template size\_t HeapInfo::Check<SmallFinalizableHeapBlock>(bool expectFull, bool expectPending, SmallFinalizableHeapBlock \* list, SmallFinalizableHeapBlock \* tail);

template size\_t HeapInfo::Check<LargeHeapBlock>(bool expectFull, bool expectPending, LargeHeapBlock \* list, LargeHeapBlock \* tail);

#ifdef RECYCLER\_WRITE\_BARRIER

template size\_t HeapInfo::Check<SmallNormalWithBarrierHeapBlock>(bool expectFull, bool expectPending, SmallNormalWithBarrierHeapBlock \* list, SmallNormalWithBarrierHeapBlock \* tail);

template size\_t HeapInfo::Check<SmallFinalizableWithBarrierHeapBlock>(bool expectFull, bool expectPending, SmallFinalizableWithBarrierHeapBlock \* list, SmallFinalizableWithBarrierHeapBlock \* tail);

#endif

template size\_t HeapInfo::Check<MediumNormalHeapBlock>(bool expectFull, bool expectPending, MediumNormalHeapBlock \* list, MediumNormalHeapBlock \* tail);

template size\_t HeapInfo::Check<MediumLeafHeapBlock>(bool expectFull, bool expectPending, MediumLeafHeapBlock \* list, MediumLeafHeapBlock \* tail);

template size\_t HeapInfo::Check<MediumFinalizableHeapBlock>(bool expectFull, bool expectPending, MediumFinalizableHeapBlock \* list, MediumFinalizableHeapBlock \* tail);

template size\_t HeapInfo::Check<LargeHeapBlock>(bool expectFull, bool expectPending, LargeHeapBlock \* list, LargeHeapBlock \* tail);

#ifdef RECYCLER\_WRITE\_BARRIER

template size\_t HeapInfo::Check<MediumNormalWithBarrierHeapBlock>(bool expectFull, bool expectPending, MediumNormalWithBarrierHeapBlock \* list, MediumNormalWithBarrierHeapBlock \* tail);

template size\_t HeapInfo::Check<MediumFinalizableWithBarrierHeapBlock>(bool expectFull, bool expectPending, MediumFinalizableWithBarrierHeapBlock \* list, MediumFinalizableWithBarrierHeapBlock \* tail);

#endif

void

HeapInfo::VerifySmallHeapBlockCount()

{

GetSmallHeapBlockCount(true);

}

void

HeapInfo::VerifyLargeHeapBlockCount()

{

GetLargeHeapBlockCount(true);

}

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void

HeapInfo::Verify()

{

Assert(!this->recycler->CollectionInProgress());

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].Verify();

}

Recycler \* recycler = this->recycler;

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].Verify();

}

#endif

largeObjectBucket.Verify();

#ifdef CONCURRENT\_GC\_ENABLED

HeapBlockList::ForEach(newLeafHeapBlockList, [recycler](SmallLeafHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

HeapBlockList::ForEach(newNormalHeapBlockList, [recycler](SmallNormalHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newNormalWithBarrierHeapBlockList, [recycler](SmallNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

HeapBlockList::ForEach(newFinalizableWithBarrierHeapBlockList, [recycler](SmallFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

#endif

HeapBlockList::ForEach(newFinalizableHeapBlockList, [recycler](SmallFinalizableHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

#endif

#ifdef CONCURRENT\_GC\_ENABLED

HeapBlockList::ForEach(newMediumLeafHeapBlockList, [recycler](MediumLeafHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

HeapBlockList::ForEach(newMediumNormalHeapBlockList, [recycler](MediumNormalHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newMediumNormalWithBarrierHeapBlockList, [recycler](MediumNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

HeapBlockList::ForEach(newMediumFinalizableWithBarrierHeapBlockList, [recycler](MediumFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

#endif

HeapBlockList::ForEach(newMediumFinalizableHeapBlockList, [recycler](MediumFinalizableHeapBlock \* heapBlock)

{

heapBlock->Verify();

});

#endif

}

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void

HeapInfo::VerifyMark()

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

heapBuckets[i].VerifyMark();

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

mediumHeapBuckets[i].VerifyMark();

}

#endif

largeObjectBucket.VerifyMark();

#ifdef CONCURRENT\_GC\_ENABLED

HeapBlockList::ForEach(newLeafHeapBlockList, [](SmallLeafHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

HeapBlockList::ForEach(newNormalHeapBlockList, [](SmallNormalHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newNormalWithBarrierHeapBlockList, [](SmallNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

HeapBlockList::ForEach(newFinalizableWithBarrierHeapBlockList, [](SmallFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

#endif

HeapBlockList::ForEach(newFinalizableHeapBlockList, [](SmallFinalizableHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

#endif

#ifdef CONCURRENT\_GC\_ENABLED

HeapBlockList::ForEach(newMediumLeafHeapBlockList, [](MediumLeafHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

HeapBlockList::ForEach(newMediumNormalHeapBlockList, [](MediumNormalHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

#ifdef RECYCLER\_WRITE\_BARRIER

HeapBlockList::ForEach(newMediumNormalWithBarrierHeapBlockList, [](MediumNormalWithBarrierHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

HeapBlockList::ForEach(newMediumFinalizableWithBarrierHeapBlockList, [](MediumFinalizableWithBarrierHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

#endif

HeapBlockList::ForEach(newMediumFinalizableHeapBlockList, [](MediumFinalizableHeapBlock \* heapBlock)

{

heapBlock->VerifyMark();

});

#endif

}

#endif

#ifdef RECYCLER\_FINALIZE\_CHECK

void

HeapInfo::VerifyFinalize()

{

// We can't check this if we are marking

Assert(!this->recycler->IsMarkState());

size\_t currentFinalizableObjectCount = this->liveFinalizableObjectCount - this->newFinalizableObjectCount - this->pendingDisposableObjectCount;

#if DBG

Assert(currentFinalizableObjectCount == this->recycler->collectionStats.finalizeCount);

#else

if (currentFinalizableObjectCount != this->recycler->collectionStats.finalizeCount)

{

Output::Print(L"ERROR: Recycler dropped some finalizable objects");

DebugBreak();

}

#endif

}

#endif

#if DBG

bool

HeapInfo::AllocatorsAreEmpty()

{

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

if (!heapBuckets[i].AllocatorsAreEmpty())

{

return false;

}

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS) && SMALLBLOCK\_MEDIUM\_ALLOC

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

if (!mediumHeapBuckets[i].AllocatorsAreEmpty())

{

return false;

}

}

#endif

return true;

}

#endif

// Block attribute functions

/\* static \*/

BOOL SmallAllocationBlockAttributes::IsAlignedObjectSize(size\_t sizeCat)

{

return HeapInfo::IsAlignedSmallObjectSize(sizeCat);

}

/\* static \*/

BOOL MediumAllocationBlockAttributes::IsAlignedObjectSize(size\_t sizeCat)

{

return HeapInfo::IsAlignedMediumObjectSize(sizeCat);

}

template class HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>;

template class ValidPointers<SmallAllocationBlockAttributes>;

template class HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>;

template class ValidPointers<MediumAllocationBlockAttributes>;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

namespace Memory

{

class HeapInfo

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

friend class ScriptMemoryDumper;

#endif

public:

HeapInfo();

~HeapInfo();

void Initialize(Recycler \* recycler

#ifdef RECYCLER\_PAGE\_HEAP

, PageHeapMode pageheapmode = PageHeapMode::PageHeapModeOff

, bool captureAllocCallStack = false

, bool captureFreeCallStack = false

#endif

);

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(RECYCLER\_MEMORY\_VERIFY) || defined(MEMSPECT\_TRACKING) || defined(ETW\_MEMORY\_TRACKING)

void Initialize(Recycler \* recycler, void(\*trackNativeAllocCallBack)(Recycler \*, void \*, size\_t)

#ifdef RECYCLER\_PAGE\_HEAP

, PageHeapMode pageheapmode = PageHeapMode::PageHeapModeOff

, bool captureAllocCallStack = false

, bool captureFreeCallStack = false

#endif

);

#endif

void ResetMarks(ResetMarkFlags flags);

void EnumerateObjects(ObjectInfoBits infoBits, void(\*CallBackFunction)(void \* address, size\_t size));

#ifdef RECYCLER\_PAGE\_HEAP

bool IsPageHeapEnabled() const{ return isPageHeapEnabled; }

template <typename TBlockAttributes>

bool IsPageHeapEnabledForBlock(const size\_t objectSize);

#else

const bool IsPageHeapEnabled() const{ return false; }

#endif

#ifdef DUMP\_FRAGMENTATION\_STATS

void DumpFragmentationStats();

#endif

template <ObjectInfoBits attributes, bool nothrow>

char \* MediumAlloc(Recycler \* recycler, size\_t sizeCat);

// Small allocator

template <ObjectInfoBits attributes, bool nothrow>

char \* RealAlloc(Recycler \* recycler, size\_t sizeCat);

template <ObjectInfoBits attributes>

bool IntegrateBlock(char \* blockAddress, PageSegment \* segment, Recycler \* recycler, size\_t sizeCat);

template <typename SmallHeapBlockAllocatorType>

void AddSmallAllocator(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat);

template <typename SmallHeapBlockAllocatorType>

void RemoveSmallAllocator(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat);

template <ObjectInfoBits attributes, typename SmallHeapBlockAllocatorType>

char \* SmallAllocatorAlloc(Recycler \* recycler, SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat);

// collection functions

void ScanInitialImplicitRoots();

void ScanNewImplicitRoots();

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t Rescan(RescanFlags flags);

void SweepPendingObjects(RecyclerSweep& recyclerSweep);

#endif

void Sweep(RecyclerSweep& recyclerSweep, bool concurrent);

template <ObjectInfoBits attributes>

void FreeSmallObject(void\* object, size\_t bytes);

template <ObjectInfoBits attributes>

void FreeMediumObject(void\* object, size\_t bytes);

#ifdef PARTIAL\_GC\_ENABLED

void SweepPartialReusePages(RecyclerSweep& recyclerSweep);

void FinishPartialCollect(RecyclerSweep \* recyclerSweep);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

void PrepareSweep();

void TransferPendingHeapBlocks(RecyclerSweep& recyclerSweep);

void ConcurrentTransferSweptObjects(RecyclerSweep& recyclerSweep);

#ifdef PARTIAL\_GC\_ENABLED

void ConcurrentPartialTransferSweptObjects(RecyclerSweep& recyclerSweep);

#endif

#endif

void DisposeObjects();

void TransferDisposedObjects();

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

void Check();

template <typename TBlockType>

static size\_t Check(bool expectFull, bool expectPending, TBlockType \* list, TBlockType \* tail = nullptr);

void VerifySmallHeapBlockCount();

void VerifyLargeHeapBlockCount();

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void Verify();

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void VerifyMark();

#endif

public:

static bool IsSmallObject(size\_t nBytes) { return nBytes <= HeapConstants::MaxSmallObjectSize; }

static bool IsMediumObject(size\_t nBytes)

{

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

return nBytes > HeapConstants::MaxSmallObjectSize && nBytes <= HeapConstants::MaxMediumObjectSize;

#else

return false;

#endif

}

static bool IsSmallBlockAllocation(size\_t nBytes)

{

#if SMALLBLOCK\_MEDIUM\_ALLOC

return HeapInfo::IsSmallObject(nBytes) || HeapInfo::IsMediumObject(nBytes);

#else

return HeapInfo::IsSmallObject(nBytes);

#endif

}

static bool IsLargeObject(size\_t nBytes)

{

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

return nBytes > HeapConstants::MaxMediumObjectSize;

#else

return nBytes > HeapConstants::MaxSmallObjectSize;

#endif

}

static BOOL IsAlignedSize(size\_t sizeCat) { return (sizeCat != 0) && (0 == (sizeCat & HeapInfo::ObjectAlignmentMask)); }

static BOOL IsAlignedSmallObjectSize(size\_t sizeCat) { return (sizeCat != 0) && (HeapInfo::IsSmallObject(sizeCat) && (0 == (sizeCat & HeapInfo::ObjectAlignmentMask))); }

static BOOL IsAlignedMediumObjectSize(size\_t sizeCat) { return (sizeCat != 0) && (HeapInfo::IsMediumObject(sizeCat) && (0 == (sizeCat & HeapInfo::ObjectAlignmentMask))); }

static size\_t GetAlignedSize(size\_t size) { return AllocSizeMath::Align(size, HeapConstants::ObjectGranularity); }

static size\_t GetAlignedSizeNoCheck(size\_t size) { return Math::Align<size\_t>(size, HeapConstants::ObjectGranularity); }

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

static size\_t GetMediumObjectAlignedSize(size\_t size) { return AllocSizeMath::Align(size, HeapConstants::MediumObjectGranularity); }

static size\_t GetMediumObjectAlignedSizeNoCheck(size\_t size) { return Math::Align<size\_t>(size, HeapConstants::MediumObjectGranularity); }

#endif

static uint GetBucketIndex(size\_t sizeCat) { Assert(IsAlignedSmallObjectSize(sizeCat)); return (uint)(sizeCat >> HeapConstants::ObjectAllocationShift) - 1; }

template <typename TBlockAttributes>

static uint GetObjectSizeForBucketIndex(uint bucketIndex) { return (bucketIndex + 1) << HeapConstants::ObjectAllocationShift; }

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

static uint GetMediumBucketIndex(size\_t sizeCat) { Assert(IsAlignedMediumObjectSize(sizeCat)); return (uint)((sizeCat - HeapConstants::MaxSmallObjectSize - 1) / HeapConstants::MediumObjectGranularity); }

template <>

static uint GetObjectSizeForBucketIndex<MediumAllocationBlockAttributes>(uint bucketIndex)

{

Assert(IsMediumObject(HeapConstants::MaxSmallObjectSize + ((bucketIndex + 1) \* HeapConstants::MediumObjectGranularity)));

return HeapConstants::MaxSmallObjectSize + ((bucketIndex + 1) \* HeapConstants::MediumObjectGranularity);

}

#endif

static BOOL IsAlignedAddress(void \* address) { return (0 == (((size\_t)address) & HeapInfo::ObjectAlignmentMask)); }

private:

template <ObjectInfoBits attributes>

typename SmallHeapBlockType<attributes, SmallAllocationBlockAttributes>::BucketType& GetBucket(size\_t sizeCat);

template<bool pageheap>

void Sweep(RecyclerSweep& recyclerSweep, bool concurrent);

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

#if SMALLBLOCK\_MEDIUM\_ALLOC

template <ObjectInfoBits attributes>

typename SmallHeapBlockType<attributes, MediumAllocationBlockAttributes>::BucketType& GetMediumBucket(size\_t sizeCat);

#else

LargeHeapBucket& GetMediumBucket(size\_t sizeCat);

#endif

#endif

LargeHeapBlock \* AddLargeHeapBlock(size\_t pageCount);

template <typename TBlockType>

void AppendNewHeapBlock(TBlockType \* heapBlock, HeapBucketT<TBlockType> \* heapBucket)

{

TBlockType \*& list = this->GetNewHeapBlockList<TBlockType>(heapBucket);

heapBlock->SetNextBlock(list);

list = heapBlock;

}

#ifdef CONCURRENT\_GC\_ENABLED

template <typename TBlockType> TBlockType \*& GetNewHeapBlockList(HeapBucketT<TBlockType> \* heapBucket);

template <>

SmallLeafHeapBlock \*& GetNewHeapBlockList<SmallLeafHeapBlock>(HeapBucketT<SmallLeafHeapBlock> \* heapBucket)

{

return this->newLeafHeapBlockList;

}

template <>

SmallNormalHeapBlock \*& GetNewHeapBlockList<SmallNormalHeapBlock>(HeapBucketT<SmallNormalHeapBlock> \* heapBucket)

{

return this->newNormalHeapBlockList;

}

template <>

SmallFinalizableHeapBlock \*& GetNewHeapBlockList<SmallFinalizableHeapBlock>(HeapBucketT<SmallFinalizableHeapBlock> \* heapBucket)

{

// Even though we don't concurrent sweep finalizable heap block, the background thread may

// find some partial swept block to be reused, thus modifying the heapBlockList in the background

// so new block can't go into heapBlockList

return this->newFinalizableHeapBlockList;

}

#ifdef RECYCLER\_WRITE\_BARRIER

template <>

SmallNormalWithBarrierHeapBlock \*& GetNewHeapBlockList<SmallNormalWithBarrierHeapBlock>(HeapBucketT<SmallNormalWithBarrierHeapBlock> \* heapBucket)

{

return this->newNormalWithBarrierHeapBlockList;

}

template <>

SmallFinalizableWithBarrierHeapBlock \*& GetNewHeapBlockList<SmallFinalizableWithBarrierHeapBlock>(HeapBucketT<SmallFinalizableWithBarrierHeapBlock> \* heapBucket)

{

return this->newFinalizableWithBarrierHeapBlockList;

}

#endif

template <>

MediumLeafHeapBlock \*& GetNewHeapBlockList<MediumLeafHeapBlock>(HeapBucketT<MediumLeafHeapBlock> \* heapBucket)

{

return this->newMediumLeafHeapBlockList;

}

template <>

MediumNormalHeapBlock \*& GetNewHeapBlockList<MediumNormalHeapBlock>(HeapBucketT<MediumNormalHeapBlock> \* heapBucket)

{

return this->newMediumNormalHeapBlockList;

}

template <>

MediumFinalizableHeapBlock \*& GetNewHeapBlockList<MediumFinalizableHeapBlock>(HeapBucketT<MediumFinalizableHeapBlock> \* heapBucket)

{

// Even though we don't concurrent sweep finalizable heap block, the background thread may

// find some partial swept block to be reused, thus modifying the heapBlockList in the background

// so new block can't go into heapBlockList

return this->newMediumFinalizableHeapBlockList;

}

#ifdef RECYCLER\_WRITE\_BARRIER

template <>

MediumNormalWithBarrierHeapBlock \*& GetNewHeapBlockList<MediumNormalWithBarrierHeapBlock>(HeapBucketT<MediumNormalWithBarrierHeapBlock> \* heapBucket)

{

return this->newMediumNormalWithBarrierHeapBlockList;

}

template <>

MediumFinalizableWithBarrierHeapBlock \*& GetNewHeapBlockList<MediumFinalizableWithBarrierHeapBlock>(HeapBucketT<MediumFinalizableWithBarrierHeapBlock> \* heapBucket)

{

return this->newMediumFinalizableWithBarrierHeapBlockList;

}

#endif

void SetupBackgroundSweep(RecyclerSweep& recyclerSweep);

template<bool pageheap>

void SweepSmallNonFinalizable(RecyclerSweep& recyclerSweep);

void SweepLargeNonFinalizable(RecyclerSweep& recyclerSweep);

#else

template <typename TBlockType> TBlockType \*& GetNewHeapBlockList(HeapBucketT<TBlockType> \* heapBucket)

{

return heapBucket->heapBlockList;

}

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t GetSmallHeapBlockCount(bool checkCount = false) const;

size\_t GetLargeHeapBlockCount(bool checkCount = false) const;

#endif

#if DBG

bool AllocatorsAreEmpty();

#endif

private:

template <typename TBlockAttributes>

class ValidPointersMap

{

#define USE\_STATIC\_VPM 1 // Disable to force generation at runtime

private:

static const uint rowSize = TBlockAttributes::MaxSmallObjectCount \* 2;

typedef ushort ValidPointersMapRow[rowSize];

typedef ValidPointersMapRow ValidPointersMapTable[TBlockAttributes::BucketCount];

typedef typename SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector InvalidBitsTable[TBlockAttributes::BucketCount];

typedef typename SmallHeapBlockT<TBlockAttributes>::BlockInfo BlockInfoMapRow[TBlockAttributes::PageCount];

typedef BlockInfoMapRow BlockInfoMapTable[TBlockAttributes::BucketCount];

// Architecture-dependent initialization done in ValidPointersMap/vpm.(32b|64b).h

#if USE\_STATIC\_VPM

static const

#endif

ValidPointersMapTable validPointersBuffer;

#if USE\_STATIC\_VPM

static const

#endif

BlockInfoMapTable blockInfoBuffer;

#if USE\_STATIC\_VPM

static const BVUnit invalidBitsData[TBlockAttributes::BucketCount][SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector::wordCount];

static const InvalidBitsTable \* const invalidBitsBuffers;

#endif

public:

#if !USE\_STATIC\_VPM

InvalidBitsTable invalidBitsBuffers;

ValidPointersMap() { GenerateValidPointersMap(validPointersBuffer, invalidBitsBuffers, blockInfoBuffer); }

#endif

static void GenerateValidPointersMap(ValidPointersMapTable& validTable, InvalidBitsTable& invalidTable, BlockInfoMapTable& blockInfoTable);

const ValidPointers<TBlockAttributes> GetValidPointersForIndex(uint index) const;

const typename SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector \* GetInvalidBitVector(uint index) const;

const typename SmallHeapBlockT<TBlockAttributes>::BlockInfo \* GetBlockInfo(uint index) const;

static HRESULT GenerateValidPointersMapHeader(LPCWSTR vpmFullPath);

static HRESULT GenerateValidPointersMapForBlockType(FILE\* file);

};

static ValidPointersMap<SmallAllocationBlockAttributes> smallAllocValidPointersMap;

static ValidPointersMap<MediumAllocationBlockAttributes> mediumAllocValidPointersMap;

public:

static HRESULT GenerateValidPointersMapHeader(LPCWSTR vpmFullPath)

{

return smallAllocValidPointersMap.GenerateValidPointersMapHeader(vpmFullPath);

}

template <typename TBlockAttributes>

static typename SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector const \* GetInvalidBitVector(uint objectSize)

{

return smallAllocValidPointersMap.GetInvalidBitVector(GetBucketIndex(objectSize));

}

template <typename TBlockAttributes>

static typename SmallHeapBlockT<TBlockAttributes>::SmallHeapBlockBitVector const \* GetInvalidBitVectorForBucket(uint bucketIndex)

{

return smallAllocValidPointersMap.GetInvalidBitVector(bucketIndex);

}

template <typename TBlockAttributes>

static typename ValidPointers<TBlockAttributes> const GetValidPointersMapForBucket(uint bucketIndex)

{

return smallAllocValidPointersMap.GetValidPointersForIndex(bucketIndex);

}

template <>

static typename SmallHeapBlockT<MediumAllocationBlockAttributes>::SmallHeapBlockBitVector const \* GetInvalidBitVector<MediumAllocationBlockAttributes>(uint objectSize)

{

return mediumAllocValidPointersMap.GetInvalidBitVector(GetMediumBucketIndex(objectSize));

}

template <>

static typename SmallHeapBlockT<MediumAllocationBlockAttributes>::SmallHeapBlockBitVector const \* GetInvalidBitVectorForBucket<MediumAllocationBlockAttributes>(uint bucketIndex)

{

return mediumAllocValidPointersMap.GetInvalidBitVector(bucketIndex);

}

template <>

static typename ValidPointers<MediumAllocationBlockAttributes> const GetValidPointersMapForBucket<MediumAllocationBlockAttributes>(uint bucketIndex)

{

return mediumAllocValidPointersMap.GetValidPointersForIndex(bucketIndex);

}

Recycler\* GetRecycler(){ return recycler; }

template <typename TBlockAttributes>

static typename SmallHeapBlockT<TBlockAttributes>::BlockInfo const \* GetBlockInfo(uint objectSize)

{

return smallAllocValidPointersMap.GetBlockInfo(GetBucketIndex(objectSize));

}

template <>

static typename SmallHeapBlockT<MediumAllocationBlockAttributes>::BlockInfo const \* GetBlockInfo<MediumAllocationBlockAttributes>(uint objectSize)

{

return mediumAllocValidPointersMap.GetBlockInfo(GetMediumBucketIndex(objectSize));

}

private:

size\_t uncollectedAllocBytes;

size\_t lastUncollectedAllocBytes;

size\_t uncollectedExternalBytes;

uint pendingZeroPageCount;

#ifdef PARTIAL\_GC\_ENABLED

size\_t uncollectedNewPageCount;

size\_t unusedPartialCollectFreeBytes;

#endif

Recycler \* recycler;

#ifdef CONCURRENT\_GC\_ENABLED

SmallLeafHeapBlock \* newLeafHeapBlockList;

SmallNormalHeapBlock \* newNormalHeapBlockList;

SmallFinalizableHeapBlock \* newFinalizableHeapBlockList;

#ifdef RECYCLER\_WRITE\_BARRIER

SmallNormalWithBarrierHeapBlock \* newNormalWithBarrierHeapBlockList;

SmallFinalizableWithBarrierHeapBlock \* newFinalizableWithBarrierHeapBlockList;

#endif

#endif

#ifdef CONCURRENT\_GC\_ENABLED

MediumLeafHeapBlock \* newMediumLeafHeapBlockList;

MediumNormalHeapBlock \* newMediumNormalHeapBlockList;

MediumFinalizableHeapBlock \* newMediumFinalizableHeapBlockList;

#ifdef RECYCLER\_WRITE\_BARRIER

MediumNormalWithBarrierHeapBlock \* newMediumNormalWithBarrierHeapBlockList;

MediumFinalizableWithBarrierHeapBlock \* newMediumFinalizableWithBarrierHeapBlockList;

#endif

#endif

#ifdef RECYCLER\_PAGE\_HEAP

PageHeapMode pageHeapMode;

bool isPageHeapEnabled;

BVStatic<HeapConstants::BucketCount> smallBlockPageHeapBucketFilter;

BVStatic<HeapConstants::MediumBucketCount> mediumBlockPageHeapBucketFilter;

bool captureAllocCallStack;

bool captureFreeCallStack;

PageHeapBlockTypeFilter pageHeapBlockType;

#endif

HeapBucketGroup<SmallAllocationBlockAttributes> heapBuckets[HeapConstants::BucketCount];

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

#if SMALLBLOCK\_MEDIUM\_ALLOC

HeapBucketGroup<MediumAllocationBlockAttributes> mediumHeapBuckets[HeapConstants::MediumBucketCount];

#else

LargeHeapBucket mediumHeapBuckets[HeapConstants::MediumBucketCount];

#endif

#endif

LargeHeapBucket largeObjectBucket;

static const size\_t ObjectAlignmentMask = HeapConstants::ObjectGranularity - 1; // 0xF

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t heapBlockCount[HeapBlock::BlockTypeCount];

#endif

#ifdef RECYCLER\_FINALIZE\_CHECK

size\_t liveFinalizableObjectCount;

size\_t newFinalizableObjectCount;

size\_t pendingDisposableObjectCount;

void VerifyFinalize();

#endif

friend class Recycler;

friend class HeapBucket;

friend class HeapBlockMap32;

friend class LargeHeapBucket;

template <typename TBlockType>

friend class HeapBucketT;

template <typename TBlockType>

friend class SmallHeapBlockAllocator;

template <typename TBlockAttributes>

friend class SmallFinalizableHeapBucketT;

template <typename TBlockAttributes>

friend class SmallLeafHeapBucketBaseT;

template <typename TBlockAttributes>

friend class SmallHeapBlockT;

template <typename TBlockAttributes>

friend class SmallLeafHeapBlockT;

template <typename TBlockAttributes>

friend class SmallFinalizableHeapBlockT;

friend class LargeHeapBlock;

friend class RecyclerSweep;

};

template <ObjectInfoBits attributes>

typename SmallHeapBlockType<attributes, SmallAllocationBlockAttributes>::BucketType&

HeapInfo::GetBucket(size\_t sizeCat)

{

uint bucket = HeapInfo::GetBucketIndex(sizeCat);

return this->heapBuckets[bucket].GetBucket<attributes>();

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

#if SMALLBLOCK\_MEDIUM\_ALLOC

template <ObjectInfoBits attributes>

typename SmallHeapBlockType<attributes, MediumAllocationBlockAttributes>::BucketType&

HeapInfo::GetMediumBucket(size\_t sizeCat)

{

uint bucket = HeapInfo::GetMediumBucketIndex(sizeCat);

return this->mediumHeapBuckets[bucket].GetBucket<attributes>();

}

#else

LargeHeapBucket&

HeapInfo::GetMediumBucket(size\_t sizeCat)

{

uint bucket = HeapInfo::GetMediumBucketIndex(sizeCat);

return this->mediumHeapBuckets[bucket];

}

#endif

#endif

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char \*

HeapInfo::RealAlloc(Recycler \* recycler, size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedSmallObjectSize(sizeCat));

auto& bucket = this->GetBucket<(ObjectInfoBits)(attributes & GetBlockTypeBitMask)>(sizeCat);

return bucket.RealAlloc<attributes, nothrow>(recycler, sizeCat);

}

#if defined(BUCKETIZE\_MEDIUM\_ALLOCATIONS)

#if SMALLBLOCK\_MEDIUM\_ALLOC

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char \*

HeapInfo::MediumAlloc(Recycler \* recycler, size\_t sizeCat)

{

auto& bucket = this->GetMediumBucket<(ObjectInfoBits)(attributes & GetBlockTypeBitMask)>(sizeCat);

return bucket.RealAlloc<attributes, nothrow>(recycler, sizeCat);

}

#else

template <ObjectInfoBits attributes, bool nothrow>

\_\_forceinline char \*

HeapInfo::MediumAlloc(Recycler \* recycler, size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedMediumObjectSize(sizeCat));

return this->GetMediumBucket<attributes>(sizeCat).Alloc<attributes, nothrow>(recycler, sizeCat);

}

#endif

#endif

template <ObjectInfoBits attributes>

\_\_inline void

HeapInfo::FreeSmallObject(void\* object, size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedSmallObjectSize(sizeCat));

return this->GetBucket<(ObjectInfoBits)(attributes & GetBlockTypeBitMask)>(sizeCat).ExplicitFree(object, sizeCat);

}

template <ObjectInfoBits attributes>

\_\_inline void

HeapInfo::FreeMediumObject(void\* object, size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedMediumObjectSize(sizeCat));

return this->GetMediumBucket<(ObjectInfoBits)(attributes & GetBlockTypeBitMask)>(sizeCat).ExplicitFree(object, sizeCat);

}

template <ObjectInfoBits attributes>

bool

HeapInfo::IntegrateBlock(char \* blockAddress, PageSegment \* segment, Recycler \* recycler, size\_t sizeCat)

{

// We only support no bit and leaf bit right now, where we don't need to set the object info in either case

CompileAssert(attributes == NoBit || attributes == LeafBit);

Assert(HeapInfo::IsAlignedSmallObjectSize(sizeCat));

return this->GetBucket<(ObjectInfoBits)(attributes & GetBlockTypeBitMask)>(sizeCat).IntegrateBlock(blockAddress, segment, recycler);

}

template <typename SmallHeapBlockAllocatorType>

void

HeapInfo::AddSmallAllocator(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedSmallObjectSize(sizeCat));

this->GetBucket<SmallHeapBlockAllocatorType::BlockType::RequiredAttributes>(sizeCat).AddAllocator(allocator);

}

template <typename SmallHeapBlockAllocatorType>

void

HeapInfo::RemoveSmallAllocator(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedSmallObjectSize(sizeCat));

this->GetBucket<SmallHeapBlockAllocatorType::BlockType::RequiredAttributes>(sizeCat).RemoveAllocator(allocator);

}

template <ObjectInfoBits attributes, typename SmallHeapBlockAllocatorType>

char \*

HeapInfo::SmallAllocatorAlloc(Recycler \* recycler, SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedSmallObjectSize(sizeCat));

CompileAssert((attributes & SmallHeapBlockAllocatorType::BlockType::RequiredAttributes) == SmallHeapBlockAllocatorType::BlockType::RequiredAttributes);

auto& bucket = this->GetBucket<SmallHeapBlockAllocatorType::BlockType::RequiredAttributes>(sizeCat);

// For now, SmallAllocatorAlloc is always throwing- but it's pretty easy to switch it if it's needed

return bucket.SnailAlloc(recycler, allocator, sizeCat, attributes, /\* nothrow = \*/ false);

}

extern template class HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>;

extern template class HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

IdleDecommitPageAllocator::IdleDecommitPageAllocator(AllocationPolicyManager \* policyManager, PageAllocatorType type,

#ifndef JD\_PRIVATE

Js::ConfigFlagsTable& flagTable,

#endif

uint maxFreePageCount, uint maxIdleFreePageCount,

bool zeroPages, BackgroundPageQueue \* backgroundPageQueue, uint maxAllocPageCount) :

#ifdef IDLE\_DECOMMIT\_ENABLED

idleDecommitTryEnterWaitFactor(0),

hasDecommitTimer(false),

hadDecommitTimer(false),

#endif

PageAllocator(policyManager,

#ifndef JD\_PRIVATE

flagTable,

#endif

type, maxFreePageCount, zeroPages, backgroundPageQueue, maxAllocPageCount),

maxIdleDecommitFreePageCount(maxIdleFreePageCount),

maxNonIdleDecommitFreePageCount(maxFreePageCount)

{

// if maxIdle is the same as max free, disable idleDecommit but setting the entry count to 1

this->idleDecommitEnterCount = (maxIdleFreePageCount == maxFreePageCount);

#ifdef IDLE\_DECOMMIT\_ENABLED

#if DBG\_DUMP

idleDecommitCount = 0;

#endif

#endif

}

void

IdleDecommitPageAllocator::EnterIdleDecommit()

{

this->idleDecommitEnterCount++;

if (this->idleDecommitEnterCount != 1)

{

return;

}

#ifdef IDLE\_DECOMMIT\_ENABLED

cs.Enter();

this->isUsed = false;

this->hadDecommitTimer = hasDecommitTimer;

PAGE\_ALLOC\_VERBOSE\_TRACE(L"EnterIdleDecommit");

if (hasDecommitTimer)

{

// Cancel the decommit timer

Assert(this->maxFreePageCount == maxIdleDecommitFreePageCount);

hasDecommitTimer = false;

PAGE\_ALLOC\_TRACE(L"Cancel Decommit Timer");

}

else

{

// Switch to maxIdleDecommitFreePageCount

Assert(this->maxFreePageCount == maxNonIdleDecommitFreePageCount);

Assert(minFreePageCount == 0);

this->maxFreePageCount = maxIdleDecommitFreePageCount;

}

cs.Leave();

Assert(!hasDecommitTimer);

#else

Assert(this->maxFreePageCount == maxNonIdleDecommitFreePageCount);

this->maxFreePageCount = maxIdleDecommitFreePageCount;

#endif

}

IdleDecommitSignal

IdleDecommitPageAllocator::LeaveIdleDecommit(bool allowTimer)

{

Assert(this->idleDecommitEnterCount > 0);

Assert(this->maxFreePageCount == maxIdleDecommitFreePageCount);

#ifdef IDLE\_DECOMMIT\_ENABLED

Assert(!hasDecommitTimer);

#endif

this->idleDecommitEnterCount--;

if (this->idleDecommitEnterCount != 0)

{

return IdleDecommitSignal\_None;

}

#ifdef IDLE\_DECOMMIT\_ENABLED

if (allowTimer)

{

cs.Enter();

PAGE\_ALLOC\_VERBOSE\_TRACE(L"LeaveIdleDecommit");

Assert(maxIdleDecommitFreePageCount != maxNonIdleDecommitFreePageCount);

IdleDecommitSignal idleDecommitSignal = IdleDecommitSignal\_None;

if (freePageCount == 0 && !isUsed && !hadDecommitTimer)

{

Assert(minFreePageCount == 0);

Assert(minFreePageCount == debugMinFreePageCount);

// Nothing to decommit, it isn't used, and there was no timer before.

// Just switch it back to non idle decommit mode

this->maxFreePageCount = maxNonIdleDecommitFreePageCount;

}

else

{

UpdateMinFreePageCount();

hasDecommitTimer = true;

idleDecommitSignal = IdleDecommitSignal\_NeedTimer;

if (isUsed)

{

// Reschedule the timer

decommitTime = ::GetTickCount() + IdleDecommitTimeout;

PAGE\_ALLOC\_TRACE( L"Schedule idle decommit at %d (%d)", decommitTime, IdleDecommitTimeout);

}

else

{

int timeDiff = (int)decommitTime - ::GetTickCount();

if (timeDiff < 20)

{

idleDecommitSignal = IdleDecommitSignal\_NeedSignal;

}

PAGE\_ALLOC\_TRACE(L"Reschedule idle decommit at %d (%d)", decommitTime, decommitTime - ::GetTickCount());

}

}

cs.Leave();

return idleDecommitSignal;

}

#endif

this->maxFreePageCount = maxNonIdleDecommitFreePageCount;

\_\_super::DecommitNow();

ClearMinFreePageCount();

return IdleDecommitSignal\_None;

}

#ifdef IDLE\_DECOMMIT\_ENABLED

void

IdleDecommitPageAllocator::DecommitNow(bool all)

{

SuspendIdleDecommit();

// If we are in non-idle-decommit mode, then always decommit all.

// Otherwise, we will end up with some un-decommitted pages and get confused later.

if (maxFreePageCount == maxNonIdleDecommitFreePageCount)

all = true;

\_\_super::DecommitNow(all);

if (all)

{

if (this->hasDecommitTimer)

{

Assert(idleDecommitEnterCount == 0);

Assert(this->maxFreePageCount == maxIdleDecommitFreePageCount);

this->hasDecommitTimer = false;

this->maxFreePageCount = maxNonIdleDecommitFreePageCount;

}

else

{

Assert((idleDecommitEnterCount > 0? maxIdleDecommitFreePageCount : maxNonIdleDecommitFreePageCount)

== this->maxFreePageCount);

}

ClearMinFreePageCount();

}

else

{

ResetMinFreePageCount();

}

ResumeIdleDecommit();

}

DWORD

IdleDecommitPageAllocator::IdleDecommit()

{

// We can check hasDecommitTimer outside of the lock because when it change to true

// the Recycler::concurrentIdleDecommitEvent will signal and we try to IdleDecommit again

// If it change to false, we check again when we acquired the lock

if (!hasDecommitTimer)

{

return INFINITE;

}

if (!cs.TryEnter())

{

// Failed to acquire the lock, wait for a variable time.

PAGE\_ALLOC\_TRACE(L"IdleDecommit Retry");

// Varies the wait time between 11 - 99

idleDecommitTryEnterWaitFactor++;

if (idleDecommitTryEnterWaitFactor >= 10)

{

idleDecommitTryEnterWaitFactor = 1;

}

DWORD waitTime = 11 \* idleDecommitTryEnterWaitFactor;

return waitTime; // Retry time

}

idleDecommitTryEnterWaitFactor = 0;

DWORD waitTime = INFINITE;

if (hasDecommitTimer)

{

Assert(this->maxFreePageCount == maxIdleDecommitFreePageCount);

int timediff = (int)(decommitTime - ::GetTickCount());

if (timediff >= 20) // Ignore time diff is it is < 20 since the system timer doesn't have that high of precision anyways

{

waitTime = (DWORD)timediff;

}

else

{

// Do the decommit in normal priority so that we don't block the main thread for too long

PAGE\_ALLOC\_TRACE(L"IdleDecommit");

#if DBG\_DUMP

idleDecommitCount++;

#endif

\_\_super::DecommitNow();

hasDecommitTimer = false;

ClearMinFreePageCount();

this->maxFreePageCount = maxNonIdleDecommitFreePageCount;

}

}

cs.Leave();

return waitTime;

}

#endif

void

IdleDecommitPageAllocator::Prime(uint primePageCount)

{

while (this->freePageCount < primePageCount)

{

PageSegment \* segment = AddPageSegment(emptySegments);

if (segment == nullptr)

{

return;

}

segment->Prime();

}

}

#if DBG

bool

IdleDecommitPageAllocator::HasMultiThreadAccess() const

{

#ifdef IDLE\_DECOMMIT\_ENABLED

return this->hasDecommitTimer && !cs.IsLocked();

#else

return false;

#endif

}

void

IdleDecommitPageAllocator::ShutdownIdleDecommit()

{

// The recycler thread should have died already

// Just set the state

idleDecommitEnterCount = 1;

#ifdef IDLE\_DECOMMIT\_ENABLED

hasDecommitTimer = false;

#endif

}

#endif

#ifdef IDLE\_DECOMMIT\_ENABLED

#if DBG\_DUMP

void

IdleDecommitPageAllocator::DumpStats() const

{

\_\_super::DumpStats();

Output::Print(L" Idle Decommit Count : %4d\n",

this->idleDecommitCount);

}

#endif

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

namespace Memory

{

enum IdleDecommitSignal

{

IdleDecommitSignal\_None,

IdleDecommitSignal\_NeedTimer,

IdleDecommitSignal\_NeedSignal

};

class IdleDecommitPageAllocator : public PageAllocator

{

public:

IdleDecommitPageAllocator(AllocationPolicyManager \* policyManager, PageAllocatorType type,

#ifndef JD\_PRIVATE

Js::ConfigFlagsTable& flagTable,

#endif

uint maxFreePageCount = 0,

uint maxIdleFreePageCount = DefaultMaxFreePageCount,

bool zeroPages = false, BackgroundPageQueue \* backgroundPageQueue = nullptr,

uint maxAllocPageCount = PageAllocator::DefaultMaxAllocPageCount);

void EnterIdleDecommit();

IdleDecommitSignal LeaveIdleDecommit(bool allowTimer);

void Prime(uint primePageCount);

#ifdef IDLE\_DECOMMIT\_ENABLED

DWORD IdleDecommit();

void DecommitNow(bool all = true);

#endif

#if DBG

virtual bool IsIdleDecommitPageAllocator() const override { return true; }

virtual bool HasMultiThreadAccess() const override;

void ShutdownIdleDecommit();

#endif

private:

#ifdef IDLE\_DECOMMIT\_ENABLED

#if DBG\_DUMP

virtual void DumpStats() const override sealed;

size\_t idleDecommitCount;

#endif

#endif

uint maxIdleDecommitFreePageCount;

uint maxNonIdleDecommitFreePageCount;

#ifdef IDLE\_DECOMMIT\_ENABLED

bool hasDecommitTimer;

bool hadDecommitTimer;

DWORD decommitTime;

uint idleDecommitTryEnterWaitFactor;

CriticalSection cs;

static const uint IdleDecommitTimeout = 1000;

#endif

friend class PageAllocatorBase<VirtualAllocWrapper>;

friend class PageAllocatorBase<PreReservedVirtualAllocWrapper>;

#if IDLE\_DECOMMIT\_ENABLED && DBG

public:

virtual void UpdateThreadContextHandle(ThreadContextId threadContextHandle) override sealed

{

PageAllocator::UpdateThreadContextHandle(threadContextHandle);

}

virtual void SetDisableThreadAccessCheck() override

{

PageAllocator::SetDisableThreadAccessCheck();

}

virtual void SetEnableThreadAccessCheck() override

{

// Can't re-enable thread access check for idle decommit page allocator

Assert(false);

}

#endif

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

CompileAssert(

sizeof(LargeObjectHeader) == HeapConstants::ObjectGranularity ||

sizeof(LargeObjectHeader) == HeapConstants::ObjectGranularity \* 2);

void \*

LargeObjectHeader::GetAddress() { return ((char \*)this) + sizeof(LargeObjectHeader); }

#ifdef LARGEHEAPBLOCK\_ENCODING

// decodedNext = decoded next field

// decodedAttributes = decoded attributes part of attributesAndChecksum

// Decode 'next' and 'attributes' using \_cookie

// If next=B1B2B3B4, checksum = (B1^B2^B3^B4^attributes)

// Encode 'next' and 'attributes' using \_cookie

unsigned char

LargeObjectHeader::CalculateCheckSum(LargeObjectHeader\* decodedNext, unsigned char decodedAttributes)

{

unsigned char checksum = 0;

byte \*nextField = (byte \*)&decodedNext;

checksum = nextField[0] ^ nextField[1] ^ nextField[2] ^ nextField[3] ^ decodedAttributes;

return checksum;

}

LargeObjectHeader\*

LargeObjectHeader::EncodeNext(uint cookie, LargeObjectHeader\* next)

{

return (LargeObjectHeader \*)((uintptr)next ^ cookie);

}

ushort

LargeObjectHeader::EncodeAttributesAndChecksum(uint cookie, ushort attributesAndChecksum)

{

return attributesAndChecksum ^ (ushort)cookie;

}

LargeObjectHeader\*

LargeObjectHeader::DecodeNext(uint cookie, LargeObjectHeader\* next) { return EncodeNext(cookie, next); }

ushort

LargeObjectHeader::DecodeAttributesAndChecksum(uint cookie) { return EncodeAttributesAndChecksum(cookie, this->attributesAndChecksum); }

#else

// If heap block encoding is disabled then have an API to expose

// pointer to attributes so that can be passed to RecyclerHeapObjectInfo()

// which updates the attributes field.

unsigned char \*

LargeObjectHeader::GetAttributesPtr()

{

return &this->attributes;

}

#endif

void

LargeObjectHeader::SetNext(uint cookie, LargeObjectHeader\* next)

{

#ifdef LARGEHEAPBLOCK\_ENCODING

ushort decodedAttributesAndChecksum = this->DecodeAttributesAndChecksum(cookie);

// Calculate the checksum value with new next

unsigned char newCheckSumValue = this->CalculateCheckSum(next, (unsigned char)(decodedAttributesAndChecksum >> 8));

// pack the (attribute + checksum)

ushort newAttributeWithCheckSum = (decodedAttributesAndChecksum & 0xFF00) | newCheckSumValue;

// encode the packed (attribute + checksum), next and set them

this->attributesAndChecksum = this->EncodeAttributesAndChecksum(cookie, newAttributeWithCheckSum);

this->next = this->EncodeNext(cookie, next);

#else

this->next = next;

#endif

}

LargeObjectHeader \*

LargeObjectHeader::GetNext(uint cookie)

{

#ifdef LARGEHEAPBLOCK\_ENCODING

LargeObjectHeader \*decodedNext = this->DecodeNext(cookie, this->next);

ushort decodedAttributesAndChecksum = this->DecodeAttributesAndChecksum(cookie);

unsigned char checkSum = (unsigned char)(decodedAttributesAndChecksum & 0xFF);

unsigned char calculatedCheckSumField = this->CalculateCheckSum(decodedNext, (unsigned char)(decodedAttributesAndChecksum >> 8));

if (checkSum != calculatedCheckSumField)

{

LargeHeapBlock\_Metadata\_Corrupted((ULONG\_PTR)this, calculatedCheckSumField);

}

// If checksum matches return the up-to-date next (in case other thread changed it from last time

// we read it in this method.

return this->DecodeNext(cookie, this->next);

#else

return this->next;

#endif

}

void

LargeObjectHeader::SetAttributes(uint cookie, unsigned char attributes)

{

#ifdef LARGEHEAPBLOCK\_ENCODING

LargeObjectHeader \*decodedNext = this->DecodeNext(cookie, this->next);

// Calculate the checksum value with new attribute

unsigned char newCheckSumValue = this->CalculateCheckSum(decodedNext, attributes);

// pack the (attribute + checksum)

ushort newAttributeWithCheckSum = ((ushort)attributes << 8) | newCheckSumValue;

// encode the packed (attribute + checksum) and set it

this->attributesAndChecksum = this->EncodeAttributesAndChecksum(cookie, newAttributeWithCheckSum);

#else

this->attributes = attributes;

#endif

}

unsigned char

LargeObjectHeader::GetAttributes(uint cookie)

{

#ifdef LARGEHEAPBLOCK\_ENCODING

LargeObjectHeader \*decodedNext = this->DecodeNext(cookie, this->next);

ushort decodedAttributesAndChecksum = this->DecodeAttributesAndChecksum(cookie);

unsigned char checkSum = (unsigned char)(decodedAttributesAndChecksum & 0xFF);

unsigned char calculatedCheckSumField = this->CalculateCheckSum(decodedNext, (unsigned char)(decodedAttributesAndChecksum >> 8));

if (checkSum != calculatedCheckSumField)

{

LargeHeapBlock\_Metadata\_Corrupted((ULONG\_PTR)this, calculatedCheckSumField);

}

// If checksum matches return the up-to-date attributes (in case other thread changed it from last time

// we read it in this method.

return this->DecodeAttributesAndChecksum(cookie) >> 8;

#else

return this->attributes;

#endif

}

size\_t

LargeHeapBlock::GetAllocPlusSize(uint objectCount)

{

// Large Heap Block Layout:

// LargeHeapBlock

// LargeObjectHeader \* [objectCount]

// TrackerData \* [objectCount] (Optional)

size\_t allocPlusSize = objectCount \* (sizeof(LargeObjectHeader \*));

#ifdef PROFILE\_RECYCLER\_ALLOC

if (Recycler::DoProfileAllocTracker())

{

allocPlusSize += objectCount \* sizeof(void \*);

}

#endif

return allocPlusSize;

}

LargeHeapBlock \*

LargeHeapBlock::New(\_\_in char \* address, size\_t pageCount, Segment \* segment, uint objectCount, LargeHeapBucket\* bucket)

{

return NoMemProtectHeapNewNoThrowPlusZ(GetAllocPlusSize(objectCount), LargeHeapBlock, address, pageCount, segment, objectCount, bucket);

}

void

LargeHeapBlock::Delete(LargeHeapBlock \* heapBlock)

{

NoMemProtectHeapDeletePlus(GetAllocPlusSize(heapBlock->objectCount), heapBlock);

}

LargeHeapBlock::LargeHeapBlock(\_\_in char \* address, size\_t pageCount, Segment \* segment, uint objectCount, LargeHeapBucket\* bucket)

: HeapBlock(LargeBlockType), pageCount(pageCount), allocAddressEnd(address), objectCount(objectCount), bucket(bucket), freeList(this)

{

Assert(address != nullptr);

Assert(pageCount != 0);

Assert(objectCount != 0);

Assert(lastCollectAllocCount == 0);

Assert(finalizeCount == 0);

Assert(next == nullptr);

Assert(!hasPartialFreeObjects);

this->address = address;

this->segment = segment;

this->isPendingConcurrentSweep = false;

this->addressEnd = this->address + this->pageCount \* AutoSystemInfo::PageSize;

RECYCLER\_PERF\_COUNTER\_INC(LargeHeapBlockCount);

RECYCLER\_PERF\_COUNTER\_ADD(LargeHeapBlockPageSize, pageCount \* AutoSystemInfo::PageSize);

}

LargeHeapBlock::~LargeHeapBlock()

{

AssertMsg(this->segment == nullptr || this->heapInfo->recycler->recyclerLargeBlockPageAllocator.IsClosed(),

"ReleasePages needs to be called before delete");

RECYCLER\_PERF\_COUNTER\_DEC(LargeHeapBlockCount);

#ifdef RECYCLER\_PAGE\_HEAP

if (this->pageHeapAllocStack != nullptr)

{

this->pageHeapAllocStack->Delete(&NoCheckHeapAllocator::Instance);

this->pageHeapAllocStack = nullptr;

}

// REVIEW: This means that the old free stack is lost when we get free the heap block

// Is this okay? Should we delay freeing heap blocks till process/thread shutdown time?

if (this->pageHeapFreeStack != nullptr)

{

this->pageHeapFreeStack->Delete(&NoCheckHeapAllocator::Instance);

this->pageHeapFreeStack = nullptr;

}

#endif

}

Recycler \*

LargeHeapBlock::GetRecycler() const

{

return this->bucket->heapInfo->recycler;

}

LargeObjectHeader \*\*

LargeHeapBlock::HeaderList()

{

// See LargeHeapBlock::GetAllocPlusSize for layout description

return (LargeObjectHeader \*\*)(((byte \*)this) + sizeof(LargeHeapBlock));

}

void

LargeHeapBlock::FinalizeAllObjects()

{

if (this->finalizeCount != 0)

{

DebugOnly(uint processedCount = 0);

for (uint i = 0; i < allocCount; i++)

{

LargeObjectHeader \* header = this->GetHeader(i);

if (header == nullptr || ((header->GetAttributes(this->heapInfo->recycler->Cookie) & FinalizeBit) == 0))

{

continue;

}

FinalizableObject \* finalizableObject = ((FinalizableObject \*)header->GetAddress());

finalizableObject->Finalize(true);

finalizableObject->Dispose(true);

#ifdef RECYCLER\_FINALIZE\_CHECK

this->heapInfo->liveFinalizableObjectCount--;

#endif

DebugOnly(processedCount++);

}

while (pendingDisposeObject != nullptr)

{

LargeObjectHeader \* header = pendingDisposeObject;

pendingDisposeObject = header->GetNext(this->heapInfo->recycler->Cookie);

Assert(header->GetAttributes(this->heapInfo->recycler->Cookie) & FinalizeBit);

Assert(this->HeaderList()[header->objectIndex] == nullptr);

void \* objectAddress = header->GetAddress();

((FinalizableObject \*)objectAddress)->Dispose(true);

#ifdef RECYCLER\_FINALIZE\_CHECK

this->heapInfo->liveFinalizableObjectCount--;

this->heapInfo->pendingDisposableObjectCount--;

#endif

DebugOnly(processedCount++);

}

Assert(this->finalizeCount == processedCount);

}

}

void

LargeHeapBlock::ReleasePagesShutdown(Recycler \* recycler)

{

#if DBG

recycler->heapBlockMap.ClearHeapBlock(this->address, this->pageCount);

// Don't release the page in shut down, the page allocator will release them faster

Assert(recycler->recyclerLargeBlockPageAllocator.IsClosed());

#endif

}

template void LargeHeapBlock::ReleasePagesSweep<true>(Recycler \* recycler);

template void LargeHeapBlock::ReleasePagesSweep<false>(Recycler \* recycler);

template<bool pageheap>

void

LargeHeapBlock::ReleasePagesSweep(Recycler \* recycler)

{

recycler->heapBlockMap.ClearHeapBlock(this->address, this->pageCount);

ReleasePages<pageheap>(recycler);

}

template void LargeHeapBlock::ReleasePages<true>(Recycler \* recycler);

template void LargeHeapBlock::ReleasePages<false>(Recycler \* recycler);

template<bool pageheap>

void

LargeHeapBlock::ReleasePages(Recycler \* recycler)

{

Assert(segment != nullptr);

char\* pageAddress = address;

size\_t realPageCount = pageCount;

#ifdef RECYCLER\_PAGE\_HEAP

if (pageheap)

{

if (InPageHeapMode())

{

if (guardPageAddress != nullptr)

{

DWORD noAccess;

if (::VirtualProtect(static\_cast<LPVOID>(guardPageAddress), AutoSystemInfo::PageSize, guardPageOldProtectFlags, &noAccess) == FALSE)

{

AssertMsg(false, "Unable to set permission for guard page.");

return;

}

AssertMsg(noAccess == PAGE\_NOACCESS, "Guard page should be PAGE\_NOACCESS");

if (this->pageHeapMode == PageHeapMode::PageHeapModeBlockStart)

{

pageAddress = guardPageAddress;

}

realPageCount = actualPageCount;

}

}

}

#endif

#ifdef RECYCLER\_FREE\_MEM\_FILL

memset(this->address, DbgMemFill, AutoSystemInfo::PageSize \* pageCount);

#endif

recycler->recyclerLargeBlockPageAllocator.Release(pageAddress, realPageCount, segment);

RECYCLER\_PERF\_COUNTER\_SUB(LargeHeapBlockPageSize, pageCount \* AutoSystemInfo::PageSize);

this->segment = nullptr;

}

BOOL

LargeHeapBlock::IsValidObject(void\* objectAddress)

{

LargeObjectHeader \* header = GetHeader(objectAddress);

return ((char \*)header >= this->address && header->objectIndex < this->allocCount && this->HeaderList()[header->objectIndex] == header);

}

#if DBG

BOOL

LargeHeapBlock::IsFreeObject(void \* objectAddress)

{

LargeObjectHeader \* header = GetHeader(objectAddress);

return ((char \*)header >= this->address && header->objectIndex < this->allocCount && this->GetHeader(header->objectIndex) == nullptr);

}

#endif

size\_t

LargeHeapBlock::GetPagesNeeded(size\_t size, bool multiplyRequest)

{

if (multiplyRequest)

{

size = AllocSizeMath::Mul(size, 4);

}

uint pageSize = AutoSystemInfo::PageSize;

size = AllocSizeMath::Add(size, sizeof(LargeObjectHeader) + (pageSize - 1));

if (size == (size\_t)-1)

{

return 0;

}

size\_t pageCount = size / pageSize;

return pageCount;

}

char\*

LargeHeapBlock::TryAllocFromFreeList(size\_t size, ObjectInfoBits attributes)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

LargeHeapBlockFreeListEntry\*\* prev = &this->freeList.entries;

LargeHeapBlockFreeListEntry\* freeListEntry = this->freeList.entries;

char\* memBlock = nullptr;

// Walk through the free list, find the first entry that can fit our desired size

while (freeListEntry)

{

LargeHeapBlockFreeListEntry\* next = freeListEntry->next;

LargeHeapBlock\* heapBlock = freeListEntry->heapBlock;

if (freeListEntry->objectSize >= size)

{

memBlock = heapBlock->AllocFreeListEntry(size, attributes, freeListEntry);

if (memBlock)

{

(\*prev) = next;

break;

}

}

prev = &freeListEntry->next;

freeListEntry = freeListEntry->next;

}

if (this->freeList.entries == nullptr)

{

this->bucket->UnregisterFreeList(&this->freeList);

}

return memBlock;

}

char\*

LargeHeapBlock::AllocFreeListEntry(size\_t size, ObjectInfoBits attributes, LargeHeapBlockFreeListEntry\* entry)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

Assert(HeapInfo::IsAlignedSize(size));

AssertMsg((attributes & TrackBit) == 0, "Large tracked object collection not implemented");

Assert(entry->heapBlock == this);

Assert(entry->headerIndex < this->objectCount);

Assert(this->HeaderList()[entry->headerIndex] == nullptr);

uint headerIndex = entry->headerIndex;

size\_t originalSize = entry->objectSize;

LargeObjectHeader \* header = (LargeObjectHeader \*) entry;

char \* allocObject = ((char\*) entry) + sizeof(LargeObjectHeader); // shouldn't overflow

char \* newAllocAddressEnd = allocObject + size;

char \* originalAllocEnd = allocObject + originalSize;

if (newAllocAddressEnd > addressEnd || newAllocAddressEnd < allocObject || (originalAllocEnd < newAllocAddressEnd))

{

return nullptr;

}

#ifdef RECYCLER\_MEMORY\_VERIFY

if (this->heapInfo->recycler->VerifyEnabled())

{

this->heapInfo->recycler->VerifyCheckFill(allocObject , originalSize);

}

#endif

memset(entry, 0, sizeof(LargeObjectHeader) + originalSize);

#ifdef RECYCLER\_MEMORY\_VERIFY

// If we're in recyclerVerify mode, fill the non-header part of the allocation

// with the verification pattern

if (this->heapInfo->recycler->VerifyEnabled())

{

memset(allocObject, Recycler::VerifyMemFill, originalSize);

}

#endif

#if DBG

LargeAllocationVerboseTrace(this->heapInfo->recycler->GetRecyclerFlagsTable(), L"Allocated object of size 0x%x in from free list entry at address 0x%p\n", size, allocObject);

#endif

Assert(allocCount <= objectCount);

header->objectIndex = headerIndex;

header->objectSize = originalSize;

header->SetAttributes(this->heapInfo->recycler->Cookie, (attributes & StoredObjectInfoBitMask));

header->markOnOOMRescan = nullptr;

header->SetNext(this->heapInfo->recycler->Cookie, nullptr);

HeaderList()[headerIndex] = header;

finalizeCount += ((attributes & FinalizeBit) != 0);

#ifdef RECYCLER\_FINALIZE\_CHECK

if (attributes & FinalizeBit)

{

HeapInfo \* heapInfo = this->heapInfo;

heapInfo->liveFinalizableObjectCount++;

heapInfo->newFinalizableObjectCount++;

}

#endif

return allocObject;

}

char\*

LargeHeapBlock::Alloc(size\_t size, ObjectInfoBits attributes)

{

Assert(HeapInfo::IsAlignedSize(size));

Assert((attributes & InternalObjectInfoBitMask) == attributes);

AssertMsg((attributes & TrackBit) == 0, "Large tracked object collection not implemented");

LargeObjectHeader \* header = (LargeObjectHeader \*)allocAddressEnd;

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

Assert(!IsPartialSweptHeader(header));

#endif

char \* allocObject = allocAddressEnd + sizeof(LargeObjectHeader); // shouldn't overflow

char \* newAllocAddressEnd = allocObject + size;

if (newAllocAddressEnd > addressEnd || newAllocAddressEnd < allocObject)

{

return nullptr;

}

Recycler\* recycler = this->heapInfo->recycler;

#if DBG

LargeAllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Allocated object of size 0x%x in existing heap block at address 0x%p\n", size, allocObject);

#endif

Assert(allocCount < objectCount);

allocAddressEnd = newAllocAddressEnd;

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

recycler->VerifyZeroFill(header, sizeof(LargeObjectHeader));

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

if (recycler->VerifyEnabled())

{

memset(header, 0, sizeof(LargeObjectHeader));

}

#endif

header->objectIndex = allocCount;

header->objectSize = size;

header->SetAttributes(recycler->Cookie, (attributes & StoredObjectInfoBitMask));

HeaderList()[allocCount++] = header;

finalizeCount += ((attributes & FinalizeBit) != 0);

#ifdef RECYCLER\_FINALIZE\_CHECK

if (attributes & FinalizeBit)

{

HeapInfo \* heapInfo = this->heapInfo;

heapInfo->liveFinalizableObjectCount++;

heapInfo->newFinalizableObjectCount++;

}

#endif

return allocObject;

}

\_\_declspec(noinline)

void

LargeHeapBlock::Mark(void\* objectAddress, MarkContext \* markContext)

{

LargeObjectHeader \* header = GetHeader(objectAddress);

if ((char \*)header < this->address)

{

return;

}

uint index = header->objectIndex;

if (index >= this->allocCount)

{

// Not allocated yet.

return;

}

if (this->HeaderList()[index] != header)

{

// header doesn't match, not a real object

return;

}

DUMP\_OBJECT\_REFERENCE(markContext->GetRecycler(), objectAddress);

if (!UpdateAttributesOfMarkedObjects(markContext, objectAddress, header->objectSize, header->GetAttributes(this->heapInfo->recycler->Cookie),

[&](unsigned char attributes) { header->SetAttributes(this->heapInfo->recycler->Cookie, attributes); }))

{

// Couldn't mark children- bail out and come back later

this->SetNeedOOMRescan(markContext->GetRecycler());

// Single page large heap block rescan all marked object on oom rescan

if (this->GetPageCount() != 1)

{

// Failed to mark the objects referenced by this object, so we'll

// revisit this on rescan

header->markOnOOMRescan = true;

}

}

}

bool

LargeHeapBlock::TestObjectMarkedBit(void\* objectAddress)

{

Assert(IsValidObject(objectAddress));

LargeObjectHeader\* pHeader = nullptr;

if (GetObjectHeader(objectAddress, &pHeader))

{

Recycler\* recycler = this->heapInfo->recycler;

return recycler->heapBlockMap.IsMarked(objectAddress);

}

return FALSE;

}

void

LargeHeapBlock::SetObjectMarkedBit(void\* objectAddress)

{

Assert(IsValidObject(objectAddress));

LargeObjectHeader\* pHeader = nullptr;

if (GetObjectHeader(objectAddress, &pHeader))

{

Recycler\* recycler = this->heapInfo->recycler;

recycler->heapBlockMap.SetMark(objectAddress);

}

}

bool

LargeHeapBlock::FindImplicitRootObject(void\* objectAddress, Recycler \* recycler, RecyclerHeapObjectInfo& heapObject)

{

if (!IsValidObject(objectAddress))

{

return false;

}

LargeObjectHeader\* pHeader = nullptr;

if (!GetObjectHeader(objectAddress, &pHeader))

{

return false;

}

#ifdef LARGEHEAPBLOCK\_ENCODING

heapObject = RecyclerHeapObjectInfo(objectAddress, recycler, this, nullptr);

heapObject.SetLargeHeapBlockHeader(pHeader);

#else

heapObject = RecyclerHeapObjectInfo(objectAddress, recycler, this, pHeader->GetAttributesPtr());

#endif

return true;

}

bool

LargeHeapBlock::FindHeapObject(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags, RecyclerHeapObjectInfo& heapObject)

{

// Currently the same actual implementation (flags is ignored)

return FindImplicitRootObject(objectAddress, recycler, heapObject);

}

bool

LargeHeapBlock::GetObjectHeader(void\* objectAddress, LargeObjectHeader\*\* ppHeader)

{

(\*ppHeader) = nullptr;

LargeObjectHeader \* header = GetHeader(objectAddress);

if ((char \*)header < this->address)

{

return false;

}

uint index = header->objectIndex;

if (this->HeaderList()[index] != header)

{

// header doesn't match, not a real object

return false;

}

Assert(index < this->allocCount);

(\*ppHeader) = header;

return true;

}

void

LargeHeapBlock::ResetMarks(ResetMarkFlags flags, Recycler\* recycler)

{

Assert(!this->needOOMRescan);

// Update the lastCollectAllocCount for sweep

this->lastCollectAllocCount = this->allocCount;

Assert(this->GetMarkCount() == 0);

Assert(!this->isPendingConcurrentSweep);

if (flags & ResetMarkFlags\_ScanImplicitRoot)

{

for (uint objectIndex = 0; objectIndex < allocCount; objectIndex++)

{

// object is allocated during the concurrent mark or it is marked, do rescan

LargeObjectHeader \* header = this->GetHeader(objectIndex);

// check if the object index is not allocated

if (header == nullptr)

{

continue;

}

// check whether the object is a leaf and doesn't need to be scanned

if ((header->GetAttributes(this->heapInfo->recycler->Cookie) & ImplicitRootBit) != 0)

{

recycler->heapBlockMap.SetMark(header->GetAddress());

}

}

}

}

LargeObjectHeader \*

LargeHeapBlock::GetHeader(void \* objectAddress)

{

Assert(objectAddress >= this->address && objectAddress < this->addressEnd);

return GetHeaderFromAddress(objectAddress);

}

LargeObjectHeader \*

LargeHeapBlock::GetHeaderFromAddress(void \* objectAddress)

{

return (LargeObjectHeader\*)(((char \*)objectAddress) - sizeof(LargeObjectHeader));

}

byte \*

LargeHeapBlock::GetRealAddressFromInterior(void \* interiorAddress)

{

for (uint i = 0; i < allocCount; i++)

{

LargeObjectHeader \* header = this->HeaderList()[i];

if (header != nullptr && !IsPartialSweptHeader(header))

{

Assert(header->objectIndex == i);

byte \* startAddress = (byte \*)header->GetAddress();

if (startAddress <= interiorAddress && (startAddress + header->objectSize > interiorAddress))

{

return startAddress;

}

}

}

return nullptr;

}

#ifdef RECYCLER\_VERIFY\_MARK

void

LargeHeapBlock::VerifyMark()

{

Assert(!this->needOOMRescan);

Recycler\* recycler = this->heapInfo->recycler;

for (uint i = 0; i < allocCount; i++)

{

LargeObjectHeader \* header = this->GetHeader(i);

if (header == nullptr)

{

continue;

}

char \* objectAddress = (char \*)header->GetAddress();

if (!recycler->heapBlockMap.IsMarked(objectAddress))

{

continue;

}

unsigned char attributes = header->GetAttributes(this->heapInfo->recycler->Cookie);

Assert((attributes & NewFinalizeBit) == 0);

if ((attributes & LeafBit) != 0)

{

continue;

}

Assert(!header->markOnOOMRescan);

char \* objectAddressEnd = objectAddress + header->objectSize;

while (objectAddress + sizeof(void \*) <= objectAddressEnd)

{

void\* target = \*(void \*\*)objectAddress;

recycler->VerifyMark(target);

objectAddress += sizeof(void \*);

}

}

}

void

LargeHeapBlock::VerifyMark(void \* objectAddress)

{

LargeObjectHeader \* header = GetHeader(objectAddress);

if ((char \*)header < this->address)

{

return;

}

uint index = header->objectIndex;

if (index >= this->allocCount)

{

// object not allocated

return;

}

if (this->HeaderList()[index] != header)

{

// header doesn't match, not a real object

return;

}

bool isMarked = this->heapInfo->recycler->heapBlockMap.IsMarked(objectAddress);

#if DBG

Assert(isMarked);

#else

if (!isMarked)

{

DebugBreak();

}

#endif

}

#endif

void

LargeHeapBlock::ScanInitialImplicitRoots(Recycler \* recycler)

{

Assert(recycler->enableScanImplicitRoots);

const HeapBlockMap& heapBlockMap = recycler->heapBlockMap;

for (uint objectIndex = 0; objectIndex < allocCount; objectIndex++)

{

// object is allocated during the concurrent mark or it is marked, do rescan

LargeObjectHeader \* header = this->GetHeader(objectIndex);

// check if the object index is not allocated

if (header == nullptr)

{

continue;

}

// check whether the object is a leaf and doesn't need to be scanned

if ((header->GetAttributes(this->heapInfo->recycler->Cookie) & LeafBit) != 0)

{

continue;

}

char \* objectAddress = (char \*)header->GetAddress();

// it is not marked, don't scan implicit root

if (!heapBlockMap.IsMarked(objectAddress))

{

continue;

}

// TODO: Assume scan interior?

DUMP\_IMPLICIT\_ROOT(recycler, objectAddress);

recycler->ScanObjectInlineInterior((void \*\*)objectAddress, header->objectSize);

}

}

void

LargeHeapBlock::ScanNewImplicitRoots(Recycler \* recycler)

{

Assert(recycler->enableScanImplicitRoots);

uint objectIndex = 0;

HeapBlockMap& heapBlockMap = recycler->heapBlockMap;

while (objectIndex < allocCount)

{

// object is allocated during the concurrent mark or it is marked, do rescan

LargeObjectHeader \* header = this->GetHeader(objectIndex);

objectIndex++;

// check if the object index is not allocated

if (header == nullptr)

{

continue;

}

// check whether the object is an implicit root

if ((header->GetAttributes(this->heapInfo->recycler->Cookie) & ImplicitRootBit) == 0)

{

continue;

}

char \* objectAddress = (char \*)header->GetAddress();

bool marked = heapBlockMap.TestAndSetMark(objectAddress);

if (!marked)

{

DUMP\_IMPLICIT\_ROOT(recycler, objectAddress);

// check whether the object is a leaf and doesn't need to be scanned

if ((header->GetAttributes(this->heapInfo->recycler->Cookie) & LeafBit) != 0)

{

continue;

}

// TODO: Assume scan interior

recycler->ScanObjectInlineInterior((void \*\*)objectAddress, header->objectSize);

}

}

}

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

bool

LargeHeapBlock::RescanOnePage(Recycler \* recycler, DWORD const writeWatchFlags)

{

Assert(this->GetPageCount() == 1);

bool const oldNeedOOMRescan = this->needOOMRescan;

// Reset this, we'll increment this if we OOM again

this->needOOMRescan = false;

// don't need to get the write watch bit if we already need to oom rescan

if (!oldNeedOOMRescan)

{

if (recycler->inEndMarkOnLowMemory)

{

// we only do oom rescan if we are on low memory mark

return false;

}

// Check the write watch bit to see if we need to rescan

ULONG\_PTR count = 1;

DWORD pageSize = AutoSystemInfo::PageSize;

void \* written;

if (GetWriteWatch(writeWatchFlags, this->GetBeginAddress(), AutoSystemInfo::PageSize, &written, &count, &pageSize) == 0 && (count != 1))

{

return false;

}

}

RECYCLER\_STATS\_INC(recycler, markData.rescanLargePageCount);

for (uint objectIndex = 0; objectIndex < allocCount; objectIndex++)

{

// object is allocated during the concurrent mark or it is marked, do rescan

LargeObjectHeader \* header = this->GetHeader(objectIndex);

// check if the object index is not allocated

if (header == nullptr)

{

continue;

}

char \* objectAddress = (char \*)header->GetAddress();

// it is not marked, don't rescan

if (!recycler->heapBlockMap.IsMarked(objectAddress))

{

continue;

}

unsigned char attributes = header->GetAttributes(this->heapInfo->recycler->Cookie);

#ifdef RECYCLER\_STATS

if (((attributes & FinalizeBit) != 0) && ((attributes & NewFinalizeBit) != 0))

{

// The concurrent thread saw a false reference to this object and marked it before the attribute was set.

// As such, our finalizeCount is not correct. Update it now.

RECYCLER\_STATS\_INC(recycler, finalizeCount);

header->SetAttributes(this->heapInfo->recycler->Cookie, (attributes & ~NewFinalizeBit));

}

#endif

// check whether the object is a leaf and doesn't need to be scanned

if ((attributes & LeafBit) != 0)

{

continue;

}

RECYCLER\_STATS\_INC(recycler, markData.rescanLargeObjectCount);

RECYCLER\_STATS\_ADD(recycler, markData.rescanLargeByteCount, header->objectSize);

if (!recycler->AddMark(objectAddress, header->objectSize))

{

this->SetNeedOOMRescan(recycler);

}

}

return true;

}

size\_t

LargeHeapBlock::Rescan(Recycler \* recycler, bool isPartialSwept, RescanFlags flags)

{

// Update the lastCollectAllocCount for sweep

this->lastCollectAllocCount = this->allocCount;

Assert(recycler->collectionState != CollectionStateConcurrentFinishMark || (flags & RescanFlags\_ResetWriteWatch));

DWORD const writeWatchFlags = (flags & RescanFlags\_ResetWriteWatch? WRITE\_WATCH\_FLAG\_RESET : 0);

if (this->GetPageCount() == 1)

{

return RescanOnePage(recycler, writeWatchFlags);

}

// Need to rescan for finish mark even if it is done on the background thread

if (recycler->collectionState != CollectionStateConcurrentFinishMark && recycler->IsConcurrentMarkState())

{

// CONCURRENT-TODO: Don't do background rescan for pages with multiple pages because

// we don't track which page we have queued up

return 0;

}

return RescanMultiPage(recycler, writeWatchFlags);

}

size\_t

LargeHeapBlock::RescanMultiPage(Recycler \* recycler, DWORD const writeWatchFlags)

{

Assert(this->GetPageCount() != 1);

DebugOnly(bool oldNeedOOMRescan = this->needOOMRescan);

// Reset this, we'll increment this if we OOM again

this->needOOMRescan = false;

size\_t rescanCount = 0;

DWORD pageSize = AutoSystemInfo::PageSize;

uint objectIndex = 0;

char \* lastPageCheckedForWriteWatch = nullptr;

bool isLastPageCheckedForWriteWatchDirty = false;

const HeapBlockMap& heapBlockMap = recycler->heapBlockMap;

while (objectIndex < allocCount)

{

// object is allocated during the concurrent mark or it is marked, do rescan

LargeObjectHeader \* header = this->GetHeader(objectIndex);

objectIndex++;

// check if the object index is not allocated

if (header == nullptr)

{

continue;

}

char \* objectAddress = (char \*)header->GetAddress();

// it is not marked, don't rescan

if (!heapBlockMap.IsMarked(objectAddress))

{

continue;

}

unsigned char attributes = header->GetAttributes(this->heapInfo->recycler->Cookie);

#ifdef RECYCLER\_STATS

if (((attributes & FinalizeBit) != 0) && ((attributes & NewFinalizeBit) != 0))

{

// The concurrent thread saw a false reference to this object and marked it before the attribute was set.

// As such, our finalizeCount is not correct. Update it now.

RECYCLER\_STATS\_INC(recycler, finalizeCount);

header->SetAttributes(this->heapInfo->recycler->Cookie, (attributes & ~NewFinalizeBit));

}

#endif

// check whether the object is a leaf and doesn't need to be scanned

if ((attributes & LeafBit) != 0)

{

continue;

}

#ifdef RECYCLER\_STATS

bool objectScanned = false;

#endif

Assert(oldNeedOOMRescan || !header->markOnOOMRescan);

// Avoid writing to the page unnecessary by checking first

if (header->markOnOOMRescan)

{

if (!recycler->AddMark(objectAddress, header->objectSize))

{

this->SetNeedOOMRescan(recycler);

header->markOnOOMRescan = true;

// We need to bail out of rescan early only if the recycler is

// trying to finish marking because of low memory. If this is

// a regular rescan, we want to try and rescan all the objects

// on the page. It's possible that the rescan OOMs but if the

// object rescan does OOM, we'll set the right bit on the

// object header. When we later rescan it in a low memory

// situation, when the bit is set, we don't need to check for

// write-watch etc. since we'd have already done that before

// setting the bit in the non-low-memory rescan case.

if (!recycler->inEndMarkOnLowMemory)

{

continue;

}

return rescanCount;

}

header->markOnOOMRescan = false;

#ifdef RECYCLER\_STATS

objectScanned = true;

#endif

}

else if (!recycler->inEndMarkOnLowMemory)

{

char \* objectAddressEnd = objectAddress + header->objectSize;

// Walk through the object, checking if any of its pages have been written to

// If it has, then queue up this object for marking

do

{

char \* pageStart = (char \*)(((size\_t)objectAddress) & ~(size\_t)(AutoSystemInfo::PageSize - 1));

/\*

\* The rescan logic for large object is as follows:

\* - We rescan the object if it was marked during concurrent mark

\* - If it was marked, since the large object has multiple pages, we'll rescan only the parts that were changed

\* - So for each page in the large object, check if it's been written to, and if it hasn't, skip looking at that region

\* - If we can't get the write watch, rescan that region

\* - However, this logic applies only if we're not rescanning because of an OOM

\* - If we are rescanning this object because of OOM (i.e !rescanBecauseOfOOM = false), rescan the whole object

\*

\* We cache the result of the write watch and the page that it was checked on so that we don't call GetWriteWatch on the same

\* page twice and inadvertently reset the write watch on a page where we've already scanned an object

\*/

if (lastPageCheckedForWriteWatch != pageStart)

{

void \* written = nullptr;

ULONG\_PTR count = 1;

lastPageCheckedForWriteWatch = pageStart;

isLastPageCheckedForWriteWatchDirty = true;

if (GetWriteWatch(writeWatchFlags, pageStart, AutoSystemInfo::PageSize, &written, &count, &pageSize) == 0 && (count != 1))

{

// Fall through to the case below where we'll update objectAddress and continue

isLastPageCheckedForWriteWatchDirty = false;

}

}

if (!isLastPageCheckedForWriteWatchDirty)

{

objectAddress = pageStart + AutoSystemInfo::PageSize;

continue;

}

// We're interested in only rescanning the parts of the object that have changed, not the whole

// object. So just queue that up for marking

char \* checkEnd = min(pageStart + AutoSystemInfo::PageSize, objectAddressEnd);

if (!recycler->AddMark(objectAddress, (checkEnd - objectAddress)))

{

this->SetNeedOOMRescan(recycler);

header->markOnOOMRescan = true;

}

#ifdef RECYCLER\_STATS

objectScanned = true;

recycler->collectionStats.markData.rescanLargePageCount++;

recycler->collectionStats.markData.rescanLargeByteCount += (checkEnd - objectAddress);

#endif

objectAddress = checkEnd;

rescanCount++;

}

while (objectAddress < objectAddressEnd);

}

RECYCLER\_STATS\_ADD(recycler, markData.rescanLargeObjectCount, objectScanned);

}

return rescanCount;

}

#endif

/\*

\* Sweep the large heap block

\*

\* If there are no finalizable or weak referenced objects, and if nothing is marked

\* that means that everything in this heap block is considered free. So the heap block

\* can be released.

\* In that case, return SweepStateEmpty

\* If there are objects to be freed, first see if they are any finalizable objects. If there

\* aren't any in this heap block, then this heap block can be swept concurrently. So return SweepStatePendingSweep

\* If there are finalizable objects, sweep them in thread. They would have been added to the pendingDispose list

\* during the finalize phase, so we return SweepStatePendingDispose.

\* In any case, if the pendingDispose list is not empty, we return SweepStatePendingDispose.

\* If the allocCount equals the max object count, or if there's no more space to allocate a large object,

\* we return SweepStateFull, so that the HeapInfo can move this to the full block list. Otherwise,

\* we return SweepStateSwept.

\*/

template SweepState LargeHeapBlock::Sweep<true>(RecyclerSweep& recyclerSweep, bool queuePendingSweep);

template SweepState LargeHeapBlock::Sweep<false>(RecyclerSweep& recyclerSweep, bool queuePendingSweep);

template<bool pageheap>

SweepState

LargeHeapBlock::Sweep(RecyclerSweep& recyclerSweep, bool queuePendingSweep)

{

Recycler \* recycler = recyclerSweep.GetRecycler();

uint markCount = GetMarkCount();

#if DBG

Assert(this->lastCollectAllocCount == this->allocCount);

Assert(markCount <= allocCount);

#endif

RECYCLER\_STATS\_INC(recycler, heapBlockCount[HeapBlock::LargeBlockType]);

#if DBG

this->expectedSweepCount = allocCount - markCount;

#endif

Assert(!this->isPendingConcurrentSweep);

bool isAllFreed = (finalizeCount == 0 && markCount == 0);

if (isAllFreed)

{

recycler->NotifyFree<pageheap>(this);

Assert(this->pendingDisposeObject == nullptr);

return SweepStateEmpty;

}

RECYCLER\_STATS\_ADD(recycler, largeHeapBlockTotalByteCount, this->pageCount \* AutoSystemInfo::PageSize);

RECYCLER\_STATS\_ADD(recycler, heapBlockFreeByteCount[HeapBlock::LargeBlockType],

addressEnd - allocAddressEnd <= HeapConstants::MaxSmallObjectSize? 0 : (size\_t)(addressEnd - allocAddressEnd));

// If the number of objects marked is not equal to the number of objects

// that have been allocated by this large heap block, that means that there

// could be some objects that need to be swept

if (markCount != allocCount)

{

Assert(this->expectedSweepCount != 0);

// We need to sweep in thread if there are any finalizable objects so

// that the PrepareFinalize() can be called before concurrent sweep

// and other finalizers. This gives the object an opportunity before any

// other script can be ran to clean up its references/states that are not

// valid since we've determined that the object is not live any more.

//

// An example is the ITrackable's tracking alias. The reference to the alias

// object needs to be clear so that the reference will not be given out again

// in other script during concurrent sweep or finalizer called before.

Assert(!recyclerSweep.IsBackground());

#ifdef CONCURRENT\_GC\_ENABLED

if (queuePendingSweep && finalizeCount == 0)

{

this->isPendingConcurrentSweep = true;

return SweepStatePendingSweep;

}

#else

Assert(!queuePendingSweep);

#endif

SweepObjects<pageheap, SweepMode\_InThread>(recycler);

if (TransferSweptObjects())

{

return SweepStatePendingDispose;

}

}

#ifdef RECYCLER\_STATS

else

{

Assert(expectedSweepCount == 0);

isForceSweeping = true;

SweepObjects<pageheap, SweepMode\_InThread>(recycler);

isForceSweeping = false;

}

#endif

if (this->pendingDisposeObject != nullptr)

{

return SweepStatePendingDispose;

}

return (allocCount == objectCount || addressEnd - allocAddressEnd <= HeapConstants::MaxSmallObjectSize) && this->freeList.entries == nullptr ?

SweepStateFull : SweepStateSwept;

}

bool

LargeHeapBlock::TrimObject(Recycler\* recycler, LargeObjectHeader\* header, size\_t sizeOfObject, bool inDispose)

{

IdleDecommitPageAllocator\* pageAllocator = recycler->GetRecyclerLargeBlockPageAllocator();

uint pageSize = AutoSystemInfo::PageSize ;

// If we have to trim an object, either we need to have more than one object in the

// heap block or we're being called as a part of force-sweep or dispose

Assert(this->allocCount > 1 || this->isForceSweeping || inDispose);

// If we have more than 1 page of bytes to free

// make sure that the number of bytes doesn't exceed the cap for a PageSegment

// since this optimization can only be applied to heap blocks using page segments.

// We also skip this optimization if the allocCount is 1 since that means

// the heap block is empty and we've been called only because we're force sweeping.

// So, skip the opt since we're going to be marking the heap block as empty soon

if (sizeOfObject > pageSize &&

this->segment->GetPageCount() <= pageAllocator->GetMaxAllocPageCount() &&

this->allocCount > 1)

{

Assert(!this->hadTrimmed);

// We want to decommit the free pages beyond 4K (the page size)

// The way large allocations work is that at most we can have 4 objects in a large heap block

// The first object can span multiple pages, the remaining 3 objects must all fit within a page

// So if the object being freed is greater than 1 page, then it must be the first object

// The objectIndex must be 0 and the header must be same as this->address

// The end address is (baseAddress + objectSize) & ~(4k - 1)

// The number of pages to free is (freePageEnd - freePageStart) / pageSize

char\* objectAddress = (char\*) header;

char\* objectEndAddress = objectAddress + sizeof(LargeObjectHeader) + header->objectSize;

uintptr alignmentMask = ~((uintptr) (AutoSystemInfo::PageSize - 1));

uintptr objectFreeAddress = (uintptr) objectAddress;

uintptr objectFreeEndAddress = ((uintptr) objectEndAddress) & alignmentMask;

size\_t bytesToFree = (objectFreeEndAddress - objectFreeAddress);

// Verify assumptions

// Make sure that the object being freed is the first object since

// the expectation in a large heap block is that the first object is the largest

// object.

// The amount of bytes to free is always less than the size of the object being freed including its header

// The exception is if the original object's size + header size is a multiple of the page size

Assert(objectAddress == this->address);

Assert(header->objectIndex == 0);

Assert(objectFreeEndAddress <= (uintptr) objectEndAddress);

Assert(objectFreeAddress <= objectFreeEndAddress);

Assert(bytesToFree < sizeOfObject + sizeof(LargeObjectHeader) || (uintptr) objectEndAddress == objectFreeEndAddress);

// If we actually have something to free, release those pages

// Move the heap block to start from the new start address

// Change the heap block map to contain an entry for only the pages that haven't been freed

// Fill up the old object's unreleased memory if we have to

Assert(bytesToFree > 0);

Assert((bytesToFree & (AutoSystemInfo::PageSize - 1)) == 0);

size\_t freePageCount = bytesToFree / AutoSystemInfo::PageSize;

Assert(freePageCount > 0);

Assert(freePageCount < this->pageCount);

// If this call to trim needs idle decommit to be suspended (e.g. dispose case)

// check if IdleDecommit has been suspended already. If it hasn't, suspend it

// This is to prevent reentrant idle decommits (e.g. sometimes dispose is called with

if (inDispose)

{

pageAllocator->SuspendIdleDecommit();

}

pageAllocator->Release((char\*) objectFreeAddress, freePageCount, this->GetSegment());

if (inDispose)

{

pageAllocator->ResumeIdleDecommit();

}

// Remove the freed pages from the heap block map

// and move the heap block to start from after the pages that were freed

// and update the page count

recycler->heapBlockMap.ClearHeapBlock(this->address, freePageCount);

this->address = (char\*) objectFreeEndAddress;

this->pageCount -= freePageCount;

FillFreeMemory(recycler, (void\*) objectFreeEndAddress, (size\_t) (objectEndAddress - objectFreeEndAddress));

#if DBG

this->hadTrimmed = true;

#endif

return true;

}

return false;

}

template <>

void

LargeHeapBlock::SweepObject<SweepMode\_InThread>(Recycler \* recycler, LargeObjectHeader \* header)

{

Assert(this->HeaderList()[header->objectIndex] == header);

// Set the header and object to null only if this is not a finalizable object

// If it's finalizable, it'll be zeroed out during dispose

if ((header->GetAttributes(this->heapInfo->recycler->Cookie) & FinalizeBit) != FinalizeBit)

{

this->HeaderList()[header->objectIndex] = nullptr;

size\_t sizeOfObject = header->objectSize;

bool objectTrimmed = false;

if (this->bucket == nullptr)

{

objectTrimmed = TrimObject(recycler, header, sizeOfObject);

}

if (!objectTrimmed)

{

FillFreeMemory(recycler, header, sizeof(LargeObjectHeader) + sizeOfObject);

}

}

}

//

// Call the finalizer on the heapblock object and add it to the pending dispose list

//

void

LargeHeapBlock::FinalizeObject(Recycler\* recycler, LargeObjectHeader\* header)

{

// The header count can also be null if this object has already been finalized

// but this method should never be called if the header list header is null

Assert(this->HeaderList()[header->objectIndex] == header);

Assert(header->GetAttributes(this->heapInfo->recycler->Cookie) & FinalizeBit);

// Call finalize to do clean up that needs to be done immediately

// (e.g. Clear the ITrackable alias reference, so it can't be revived during

// other finalizers or concurrent sweep)

// Call it only if it hasn't already been finalized

((FinalizableObject \*)header->GetAddress())->Finalize(false);

header->SetNext(this->heapInfo->recycler->Cookie, this->pendingDisposeObject);

this->pendingDisposeObject = header;

// Null out the header in the header list- this means that this object has already

// been finalized and is just pending dispose

this->HeaderList()[header->objectIndex] = nullptr;

#ifdef RECYCLER\_FINALIZE\_CHECK

recycler->autoHeap.pendingDisposableObjectCount++;

#endif

}

// Explicitly instantiate all the sweep modes

template void LargeHeapBlock::SweepObjects<false, SweepMode\_InThread>(Recycler \* recycler);

template void LargeHeapBlock::SweepObjects<true, SweepMode\_InThread>(Recycler \* recycler);

#ifdef CONCURRENT\_GC\_ENABLED

template <>

void

LargeHeapBlock::SweepObject<SweepMode\_Concurrent>(Recycler \* recycler, LargeObjectHeader \* header)

{

Assert(!(header->GetAttributes(this->heapInfo->recycler->Cookie) & FinalizeBit));

Assert(this->HeaderList()[header->objectIndex] == header);

this->HeaderList()[header->objectIndex] = nullptr;

FillFreeMemory(recycler, header, sizeof(LargeObjectHeader) + header->objectSize);

}

// Explicitly instantiate all the sweep modes

template void LargeHeapBlock::SweepObjects<false, SweepMode\_Concurrent>(Recycler \* recycler);

#ifdef PARTIAL\_GC\_ENABLED

template <>

void

LargeHeapBlock::SweepObject<SweepMode\_ConcurrentPartial>(Recycler \* recycler, LargeObjectHeader \* header)

{

Assert(!(header->GetAttributes(this->heapInfo->recycler->Cookie) & FinalizeBit));

Assert(this->HeaderList()[header->objectIndex] == header);

this->HeaderList()[header->objectIndex] = (LargeObjectHeader \*)((size\_t)header | PartialFreeBit);

DebugOnly(this->hasPartialFreeObjects = true);

}

// Explicitly instantiate all the sweep modes

template void LargeHeapBlock::SweepObjects<false, SweepMode\_ConcurrentPartial>(Recycler \* recycler);

#endif

#endif

//

// Walk through the objects in this heap block and call finalize

// on them if they're not marked and finalizable.

//

// At the end of this phase, if there were any finalizable objects,

// they would be in the pendingDisposeObject list. When we later call

// sweep on this heapblock, we'd simply null out the header and zero out the memory

// and then Sweep would return PendingDispose as its state

//

void LargeHeapBlock::FinalizeObjects(Recycler\* recycler)

{

const HeapBlockMap& heapBlockMap = recycler->heapBlockMap;

for (uint i = 0; i < this->lastCollectAllocCount; i++)

{

LargeObjectHeader \* header = this->GetHeader(i);

if (header == nullptr)

{

continue;

}

Assert(header->objectIndex == i);

// Skip finalization if the object is alive

if (heapBlockMap.IsMarked(header->GetAddress()))

{

continue;

}

if ((header->GetAttributes(this->heapInfo->recycler->Cookie) & FinalizeBit) == FinalizeBit)

{

recycler->NotifyFree((char \*)header->GetAddress(), header->objectSize);

FinalizeObject(recycler, header);

}

}

}

template <bool pageheap, SweepMode mode>

void

LargeHeapBlock::SweepObjects(Recycler \* recycler)

{

Assert(mode == SweepMode\_InThread || this->isPendingConcurrentSweep);

const HeapBlockMap& heapBlockMap = recycler->heapBlockMap;

#if DBG

uint markCount = GetMarkCount();

// mark count included newly allocated objects

#ifdef CONCURRENT\_GC\_ENABLED

Assert(expectedSweepCount == allocCount - markCount || recycler->collectionState == CollectionStateConcurrentSweep);

#else

Assert(expectedSweepCount == allocCount - markCount);

#endif

Assert(expectedSweepCount != 0 || isForceSweeping);

uint sweepCount = 0;

#endif

for (uint i = 0; i < lastCollectAllocCount; i++)

{

RECYCLER\_STATS\_ADD(recycler, objectSweepScanCount, !isForceSweeping);

LargeObjectHeader \* header = this->GetHeader(i);

if (header == nullptr)

{

#if DBG

Assert(expectedSweepCount != 0);

expectedSweepCount--;

#endif

#if DBG

LargeAllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Index %d empty\n", i);

#endif

continue;

}

Assert(header->objectIndex == i);

// Skip sweep if the object is alive

if (heapBlockMap.IsMarked(header->GetAddress()))

{

#if DBG

Assert((header->GetAttributes(recycler->Cookie) & NewFinalizeBit) == 0);

#endif

RECYCLER\_STATS\_ADD(recycler, largeHeapBlockUsedByteCount, this->GetHeader(i)->objectSize);

continue;

}

size\_t objectSize = header->objectSize;

recycler->NotifyFree((char \*)header->GetAddress(), objectSize);

SweepObject<mode>(recycler, header);

if (this->bucket != nullptr

#ifdef RECYCLER\_STATS

&& !isForceSweeping

#endif

)

{

LargeHeapBlockFreeListEntry\* head = this->freeList.entries;

LargeHeapBlockFreeListEntry\* entry = (LargeHeapBlockFreeListEntry\*) header;

entry->headerIndex = i;

entry->heapBlock = this;

entry->next = head;

entry->objectSize = objectSize;

this->freeList.entries = entry;

}

#if DBG

sweepCount++;

#endif

}

Assert(sweepCount == expectedSweepCount);

this->isPendingConcurrentSweep = false;

}

bool

LargeHeapBlock::TransferSweptObjects()

{

// TODO : Large heap block doesn't do free listing yet

return pendingDisposeObject != nullptr;

}

void

LargeHeapBlock::DisposeObjects(Recycler \* recycler)

{

Assert(this->pendingDisposeObject != nullptr || this->hasDisposeBeenCalled);

while (pendingDisposeObject != nullptr)

{

#if DBG

this->hasDisposeBeenCalled = true;

#endif

LargeObjectHeader \* header = pendingDisposeObject;

pendingDisposeObject = header->GetNext(this->heapInfo->recycler->Cookie);

Assert(header->GetAttributes(this->heapInfo->recycler->Cookie) & FinalizeBit);

Assert(this->HeaderList()[header->objectIndex] == nullptr);

void \* objectAddress = header->GetAddress();

((FinalizableObject \*)objectAddress)->Dispose(false);

Assert(finalizeCount != 0);

finalizeCount--;

bool objectTrimmed = false;

if (this->bucket == nullptr)

{

objectTrimmed = TrimObject(recycler, header, header->objectSize, true /\* need suspend \*/);

}

// GCTODO: Consider free listing items after Dispose too

// GCTODO: Consider compacting heap blocks- if the last n items are free, move the address pointer

// back to before the nth item so we can bump allocate from this heap block

if (!objectTrimmed)

{

FillFreeMemory(recycler, header, sizeof(LargeObjectHeader) + header->objectSize);

}

RECYCLER\_STATS\_INC(recycler, finalizeSweepCount);

#ifdef RECYCLER\_FINALIZE\_CHECK

this->heapInfo->liveFinalizableObjectCount--;

this->heapInfo->pendingDisposableObjectCount--;

#endif

}

}

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

void

LargeHeapBlock::PartialTransferSweptObjects()

{

// Nothing to do

Assert(this->hasPartialFreeObjects);

}

void

LargeHeapBlock::FinishPartialCollect(Recycler \* recycler)

{

Assert(this->hasPartialFreeObjects);

for (uint i = 0; i < allocCount; i++)

{

LargeObjectHeader \* header = this->HeaderList()[i];

if (header != nullptr && IsPartialSweptHeader(header))

{

header = (LargeObjectHeader \*)((size\_t)header & ~PartialFreeBit);

Assert(header->objectIndex == i);

this->HeaderList()[i] = nullptr;

FillFreeMemory(recycler, header, sizeof(LargeObjectHeader) + header->objectSize);

}

}

DebugOnly(this->hasPartialFreeObjects = false);

}

#endif

void

LargeHeapBlock::EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size))

{

for (uint i = 0; i < allocCount; i++)

{

LargeObjectHeader \* header = this->GetHeader(i);

if (header == nullptr)

{

continue;

}

if ((header->GetAttributes(this->heapInfo->recycler->Cookie) & infoBits) != 0)

{

CallBackFunction(header->GetAddress(), header->objectSize);

}

}

}

uint

LargeHeapBlock::GetMaxLargeObjectCount(size\_t pageCount, size\_t firstAllocationSize)

{

size\_t freeSize = (AutoSystemInfo::PageSize \* pageCount) - firstAllocationSize - sizeof(LargeObjectHeader);

Assert(freeSize < AutoSystemInfo::Data.dwAllocationGranularity);

size\_t objectCount = (freeSize / HeapConstants::MaxSmallObjectSize) + 1;

Assert(objectCount <= UINT\_MAX);

return (uint)objectCount;

}

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

void

LargeHeapBlock::Check(bool expectFull, bool expectPending)

{

for (uint i = 0; i < allocCount; i++)

{

LargeObjectHeader \* header = this->HeaderList()[i];

if (header == nullptr)

{

continue;

}

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

header = (LargeObjectHeader \*)((size\_t)header & ~PartialFreeBit);

Assert(this->hasPartialFreeObjects || header == this->HeaderList()[i]);

#endif

Assert(header->objectIndex == i);

}

}

#endif

void LargeHeapBlock::FillFreeMemory(Recycler \* recycler, \_\_in\_bcount(size) void \* address, size\_t size)

{

// For now, we don't do anything in release build because we don't reuse this memory until we return

// the pages to the allocator which will zero out the whole page

#ifdef RECYCLER\_MEMORY\_VERIFY

if (recycler->VerifyEnabled())

{

memset(address, Recycler::VerifyMemFill, size);

return;

}

#endif

#ifdef RECYCLER\_FREE\_MEM\_FILL

memset(address, DbgMemFill, size);

#endif

}

size\_t LargeHeapBlock::GetObjectSize(void\* objectAddress)

{

LargeObjectHeader \* header = GetHeader(objectAddress);

Assert((char \*)header >= this->address);

return header->objectSize;

}

#ifdef RECYCLER\_MEMORY\_VERIFY

void

LargeHeapBlock::Verify(Recycler \* recycler)

{

char \* lastAddress = this->address;

uint verifyFinalizeCount = 0;

for (uint i = 0; i < allocCount; i++)

{

LargeObjectHeader \* header = this->HeaderList()[i];

if (header == nullptr)

{

// Check if the object if on the free list

LargeHeapBlockFreeListEntry\* current = this->freeList.entries;

while (current != nullptr)

{

// Verify the free listed object

if (current->headerIndex == i)

{

BYTE\* objectAddress = (BYTE \*)current + sizeof(LargeObjectHeader);

Recycler::VerifyCheck(current->heapBlock == this, L"Invalid heap block", this, current->heapBlock);

Recycler::VerifyCheck((char \*)current >= lastAddress, L"LargeHeapBlock invalid object header order", this->address, current);

Recycler::VerifyCheckFill(lastAddress, (char \*)current - lastAddress);

recycler->VerifyCheckPad(objectAddress, current->objectSize);

lastAddress = (char \*) objectAddress + current->objectSize;

break;

}

current = current->next;

}

continue;

}

Recycler::VerifyCheck((char \*)header >= lastAddress, L"LargeHeapBlock invalid object header order", this->address, header);

Recycler::VerifyCheckFill(lastAddress, (char \*)header - lastAddress);

Recycler::VerifyCheck(header->objectIndex == i, L"LargeHeapBlock object index mismatch", this->address, &header->objectIndex);

recycler->VerifyCheckPad((BYTE \*)header->GetAddress(), header->objectSize);

verifyFinalizeCount += ((header->GetAttributes(this->heapInfo->recycler->Cookie) & FinalizeBit) != 0);

lastAddress = (char \*)header->GetAddress() + header->objectSize;

}

Recycler::VerifyCheck(verifyFinalizeCount == this->finalizeCount, L"LargeHeapBlock finalize object count mismatch", this->address, &this->finalizeCount);

}

#endif

uint

LargeHeapBlock::GetMarkCount()

{

uint markCount = 0;

const HeapBlockMap& heapBlockMap = this->heapInfo->recycler->heapBlockMap;

for (uint i = 0; i < allocCount; i++)

{

LargeObjectHeader\* header = this->HeaderList()[i];

if (header && header->objectIndex == i && heapBlockMap.IsMarked(header->GetAddress()))

{

markCount++;

}

}

return markCount;

}

#ifdef RECYCLER\_PERF\_COUNTERS

void

LargeHeapBlock::UpdatePerfCountersOnFree()

{

Assert(GetMarkCount() == 0);

size\_t usedCount = 0;

size\_t usedBytes = 0;

for (uint i = 0; i < allocCount; i++)

{

LargeObjectHeader \* header = this->HeaderList()[i];

if (header == nullptr)

{

continue;

}

usedCount++;

usedBytes += header->objectSize;

}

RECYCLER\_PERF\_COUNTER\_SUB(LargeHeapBlockLiveObject, usedCount);

RECYCLER\_PERF\_COUNTER\_SUB(LargeHeapBlockLiveObjectSize, usedBytes);

RECYCLER\_PERF\_COUNTER\_SUB(LargeHeapBlockFreeObjectSize, this->GetPageCount() \* AutoSystemInfo::PageSize - usedBytes);

RECYCLER\_PERF\_COUNTER\_SUB(LiveObject, usedCount);

RECYCLER\_PERF\_COUNTER\_SUB(LiveObjectSize, usedBytes);

RECYCLER\_PERF\_COUNTER\_SUB(FreeObjectSize, this->GetPageCount() \* AutoSystemInfo::PageSize - usedBytes);

}

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

void \*

LargeHeapBlock::GetTrackerData(void \* address)

{

Assert(Recycler::DoProfileAllocTracker());

LargeObjectHeader \* header = GetHeader(address);

Assert((char \*)header >= this->address);

uint index = header->objectIndex;

Assert(index < this->allocCount);

Assert(this->HeaderList()[index] == header);

return this->GetTrackerDataArray()[index];

}

void

LargeHeapBlock::SetTrackerData(void \* address, void \* data)

{

Assert(Recycler::DoProfileAllocTracker());

LargeObjectHeader \* header = GetHeader(address);

Assert((char \*)header >= this->address);

uint index = header->objectIndex;

Assert(index < this->allocCount);

Assert(this->HeaderList()[index] == header);

this->GetTrackerDataArray()[index] = data;

}

void \*\*

LargeHeapBlock::GetTrackerDataArray()

{

// See LargeHeapBlock::GetAllocPlusSize for layout description

return (void \*\*)((char \*)(this + 1) + LargeHeapBlock::GetAllocPlusSize(this->objectCount) - this->objectCount \* sizeof(void \*));

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#if defined(\_M\_X64\_OR\_ARM64)

#define UINT\_PAD\_64BIT(x) uint x

#else

#define UINT\_PAD\_64BIT(x)

#endif

namespace Memory

{

struct LargeObjectHeader

{

uint objectIndex;

UINT\_PAD\_64BIT(unused1);

size\_t objectSize;

private:

LargeObjectHeader \* next;

// xxxxxxxxyyyyyyyy, where xxxxxxxx = attributes field, yyyyyyyy = checksum field

#if LARGEHEAPBLOCK\_ENCODING

ushort attributesAndChecksum;

#else

unsigned char attributes;

unsigned char unused2;

#endif

public:

bool markOnOOMRescan;

#if DBG

bool isExplicitFreed;

#else

unsigned char unused3;

#endif

UINT\_PAD\_64BIT(unused4);

void \*GetAddress();

#ifdef LARGEHEAPBLOCK\_ENCODING

unsigned char CalculateCheckSum(LargeObjectHeader\* next, unsigned char attributes);

LargeObjectHeader\* EncodeNext(uint cookie, LargeObjectHeader\* next);

ushort EncodeAttributesAndChecksum(uint cookie, ushort attributesWithChecksum);

LargeObjectHeader\* DecodeNext(uint cookie, LargeObjectHeader\* next);

ushort DecodeAttributesAndChecksum(uint cookie);

#else

unsigned char \* GetAttributesPtr();

#endif

void SetNext(uint cookie, LargeObjectHeader\* next);

LargeObjectHeader \* GetNext(uint cookie);

void SetAttributes(uint cookie, unsigned char attributes);

unsigned char GetAttributes(uint cookie);

};

class LargeHeapBlock;

class LargeHeapBucket;

struct LargeHeapBlockFreeListEntry

{

uint headerIndex;

size\_t objectSize;

LargeHeapBlock\* heapBlock;

LargeHeapBlockFreeListEntry\* next;

};

struct LargeHeapBlockFreeList

{

public:

LargeHeapBlockFreeList(LargeHeapBlock\* heapBlock):

previous(nullptr),

next(nullptr),

entries(nullptr),

heapBlock(heapBlock)

{

}

LargeHeapBlockFreeList\* previous;

LargeHeapBlockFreeList\* next;

LargeHeapBlockFreeListEntry\* entries;

LargeHeapBlock\* heapBlock;

};

// CONSIDER: Templatizing this so that we don't have free list support if we don't need it

class LargeHeapBlock sealed : public HeapBlock

{

public:

Recycler \* GetRecycler() const;

#if DBG

virtual BOOL IsFreeObject(void\* objectAddress) override;

#endif

virtual BOOL IsValidObject(void\* objectAddress) override;

void Mark(void\* objectAddress, MarkContext \* markContext);

virtual byte\* GetRealAddressFromInterior(void\* interiorAddress) override;

bool TestObjectMarkedBit(void\* objectAddress) override;

void SetObjectMarkedBit(void\* objectAddress) override;

bool FindHeapObject(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject) override;

virtual size\_t GetObjectSize(void\* object) override;

bool FindImplicitRootObject(void\* objectAddress, Recycler \* recycler, RecyclerHeapObjectInfo& heapObject);

size\_t GetPageCount() const { return pageCount; }

LargeHeapBlock \* GetNextBlock() { return next; }

void SetNextBlock(LargeHeapBlock \* next) { this->next = next; }

size\_t GetFreeSize() const { return addressEnd - allocAddressEnd; }

static LargeHeapBlock \* New(\_\_in char \* address, size\_t pageCount, Segment \* segment, uint objectCount, LargeHeapBucket\* bucket);

static void Delete(LargeHeapBlock \* heapBlock);

bool IsInPendingDisposeList() { return isInPendingDisposeList; }

void SetIsInPendingDisposeList(bool isInPendingDisposeList) { this->isInPendingDisposeList = isInPendingDisposeList; }

#if DBG

void SetHasDisposeBeenCalled(bool hasDisposeBeenCalled) { this->hasDisposeBeenCalled = hasDisposeBeenCalled; }

#endif

LargeHeapBlockFreeList\* GetFreeList() { return &this->freeList; }

~LargeHeapBlock();

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t Rescan(Recycler\* recycler, bool isPartialSwept, RescanFlags flags);

#endif

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

void PartialTransferSweptObjects();

void FinishPartialCollect(Recycler \* recycler);

#endif

template<bool pageheap>

void ReleasePages(Recycler \* recycler);

template<bool pageheap>

void ReleasePagesSweep(Recycler \* recycler);

void ReleasePagesShutdown(Recycler \* recycler);

void ResetMarks(ResetMarkFlags flags, Recycler\* recycler);

void ScanInitialImplicitRoots(Recycler \* recycler);

void ScanNewImplicitRoots(Recycler \* recycler);

template<bool pageheap>

SweepState Sweep(RecyclerSweep& recyclerSweep, bool queuePendingSweep);

template <bool pageheap, SweepMode mode>

void SweepObjects(Recycler \* recycler);

bool TransferSweptObjects();

void DisposeObjects(Recycler \* recycler);

void FinalizeObjects(Recycler\* recycler);

void FinalizeAllObjects();

char\* GetBeginAddress() const { return address; }

char\* GetEndAddress() const { return addressEnd; }

#ifdef RECYCLER\_PAGE\_HEAP

void SetEndAllocAddress(\_\_in char\* endAllocAddress) { allocAddressEnd = endAllocAddress; }

#endif

char \* Alloc(size\_t size, ObjectInfoBits attributes);

char \* TryAllocFromFreeList(size\_t size, ObjectInfoBits attributes);

static size\_t GetPagesNeeded(size\_t size, bool multiplyRequest);

static uint GetMaxLargeObjectCount(size\_t pageCount, size\_t firstAllocationSize);

void EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size));

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

void Check(bool expectFull, bool expectPending);

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void Verify(Recycler \* recycler);

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void VerifyMark();

virtual void VerifyMark(void \* objectAddress) override;

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

virtual void UpdatePerfCountersOnFree() override;

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

virtual void \* GetTrackerData(void \* address) override;

virtual void SetTrackerData(void \* address, void \* data) override;

#endif

private:

friend class LargeHeapBucket;

#ifdef RECYCLER\_MEMORY\_VERIFY

friend class Recycler;

#endif

LargeHeapBlock(\_\_in char \* address, size\_t pageCount, Segment \* segment, uint objectCount, LargeHeapBucket\* bucket);

static LargeObjectHeader \* GetHeaderFromAddress(void \* address);

LargeObjectHeader \* GetHeader(void \* address);

LargeObjectHeader \*\* HeaderList();

LargeObjectHeader \* GetHeader(uint index)

{

Assert(index < this->allocCount);

LargeObjectHeader \* header = this->HeaderList()[index];

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

if (IsPartialSweptHeader(header))

{

return nullptr;

}

#endif

return header;

}

uint GetMarkCount();

bool GetObjectHeader(void\* objectAddress, LargeObjectHeader\*\* ppHeader);

BOOL IsNewHeapBlock() const { return lastCollectAllocCount == 0; }

static size\_t GetAllocPlusSize(uint objectCount);

char \* AllocFreeListEntry(size\_t size, ObjectInfoBits attributes, LargeHeapBlockFreeListEntry\* entry);

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

bool RescanOnePage(Recycler \* recycler, DWORD const writeWatchFlags);

size\_t RescanMultiPage(Recycler \* recycler, DWORD const writeWatchFlags);

#endif

template <SweepMode>

void SweepObject(Recycler \* recycler, LargeObjectHeader \* header);

bool TrimObject(Recycler\* recycler, LargeObjectHeader\* header, size\_t sizeOfObject, bool needSuspend = false);

void FinalizeObject(Recycler\* recycler, LargeObjectHeader\* header);

void FillFreeMemory(Recycler \* recycler, \_\_in\_bcount(size) void \* address, size\_t size);

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

bool IsPartialSweptHeader(LargeObjectHeader \* header) const

{

Assert(this->hasPartialFreeObjects || (((size\_t)header & PartialFreeBit) != PartialFreeBit));

return ((size\_t)header & PartialFreeBit) == PartialFreeBit;

}

static const size\_t PartialFreeBit = 0x1;

#endif

size\_t pageCount;

size\_t actualPageCount;

// The number of allocations that have occurred from this heap block

// This only increases, never decreases. Instead, we rely on the mark/weakRef/finalize counts

// to determine if an object has been freed or not. So when we alloc, we keep alloc'ing

// from the last allocation even if there are holes in the large heap block. If we free an object,

// we simply set its header to null. But otherwise, we simply constantly keep increasing allocCount

// till the heap block is full.

uint allocCount;

// Maximum number of objects that can be fit into this heap block

// This is based on the fact that the largest small object size is 1024

// So the smallest large object size is 1025. We can calculate the max object count

// as follows. The total size available to us is the pageCount \* pageSize.

// When we allocate the large heap block, it's to fit a large object. So the amount

// of space remaining is totalSize - sizeOfLargeObject - sizeOfLargeObjectHeader

// So the max number of objects this heap block can support is remainingSize / maxSmallObjectSize + 1

// where 1 is the initial object that was used to create the heap block

uint objectCount;

char \* allocAddressEnd;

char \* addressEnd;

LargeHeapBlock\* next;

LargeObjectHeader \* pendingDisposeObject;

LargeHeapBucket\* bucket;

LargeHeapBlockFreeList freeList;

uint lastCollectAllocCount;

uint finalizeCount;

bool isInPendingDisposeList;

#if DBG

bool hasDisposeBeenCalled;

bool hasPartialFreeObjects;

uint expectedSweepCount;

// The following get set if an object is swept and we freed its pages

bool hadTrimmed;

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

friend class ScriptMemoryDumper;

#endif

friend class HeapInfo;

HeapInfo \* heapInfo;

#ifdef PROFILE\_RECYCLER\_ALLOC

void \*\* GetTrackerDataArray();

#endif

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

//=====================================================================================================

// Initialization

//=====================================================================================================

LargeHeapBucket::~LargeHeapBucket()

{

Recycler\* recycler = this->heapInfo->recycler;

HeapInfo\* autoHeap = this->heapInfo;

ForEachEditingLargeHeapBlock([recycler, autoHeap](LargeHeapBlock \* heapBlock)

{

heapBlock->ReleasePagesShutdown(recycler);

LargeHeapBlock::Delete(heapBlock);

RECYCLER\_SLOW\_CHECK(autoHeap->heapBlockCount[HeapBlock::HeapBlockType::LargeBlockType]--);

});

}

void

LargeHeapBucket::Initialize(HeapInfo \* heapInfo, uint sizeCat, bool supportFreeList)

{

this->heapInfo = heapInfo;

this->sizeCat = sizeCat;

#ifdef RECYCLER\_PAGE\_HEAP

this->isPageHeapEnabled = heapInfo->IsPageHeapEnabledForBlock<LargeAllocationBlockAttributes>(sizeCat);

#endif

this->supportFreeList = supportFreeList;

}

//=====================================================================================================

// Allocation

//=====================================================================================================

char \*

LargeHeapBucket::TryAllocFromNewHeapBlock(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes, bool nothrow)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

#ifdef RECYCLER\_PAGE\_HEAP

if (IsPageHeapEnabled())

{

return this->PageHeapAlloc(recycler, sizeCat, attributes, this->heapInfo->pageHeapMode, true);

}

#endif

LargeHeapBlock \* heapBlock = AddLargeHeapBlock(sizeCat, nothrow);

if (heapBlock == nullptr)

{

return nullptr;

}

char \* memBlock = heapBlock->Alloc(sizeCat, attributes);

Assert(memBlock != nullptr);

return memBlock;

}

char \*

LargeHeapBucket::SnailAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes, bool nothrow)

{

char \* memBlock;

Assert((attributes & InternalObjectInfoBitMask) == attributes);

// No free memory, try to collect with allocated bytes and time heuristic, and concurrently

BOOL collected = recycler->disableCollectOnAllocationHeuristics ? recycler->FinishConcurrent<FinishConcurrentOnAllocation>() :

recycler->CollectNow<CollectOnAllocation>();

if (!collected)

{

memBlock = TryAllocFromNewHeapBlock(recycler, sizeCat, attributes, nothrow);

if (memBlock != nullptr)

{

return memBlock;

}

// Can't even allocate a new block, we need force a collection and

// allocate some free memory, add a new heap block again, or throw out of memory

AllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"LargeHeapBucket::AddLargeHeapBlock failed, forcing in-thread collection\n");

recycler->CollectNow<CollectNowForceInThread>();

}

memBlock = TryAlloc(recycler, sizeCat, attributes);

if (memBlock != nullptr)

{

return memBlock;

}

memBlock = TryAllocFromNewHeapBlock(recycler, sizeCat, attributes, nothrow);

if (memBlock != nullptr)

{

return memBlock;

}

if (nothrow == false)

{

// Can't add a heap block, we are out of memory

// Since nothrow is false, we can throw right here

recycler->OutOfMemory();

}

return nullptr;

}

#ifdef RECYCLER\_PAGE\_HEAP

char\*

LargeHeapBucket::PageHeapAlloc(Recycler \* recycler, size\_t size, ObjectInfoBits attributes, PageHeapMode mode, bool nothrow)

{

size\_t sizeCat = HeapInfo::GetAlignedSizeNoCheck(size);

Segment \* segment;

size\_t pageCount = LargeHeapBlock::GetPagesNeeded(size, this->supportFreeList);

if (pageCount == 0)

{

if (nothrow == false)

{

// overflow

// Since nothrow is false here, it's okay to throw

recycler->OutOfMemory();

}

return nullptr;

}

size\_t actualPageCount = pageCount + 1; // for page heap

char \* baseAddress = recycler->GetRecyclerLargeBlockPageAllocator()->Alloc(&actualPageCount, &segment);

if (baseAddress == nullptr)

{

return nullptr;

}

char\* address = nullptr;

char\* guardPageAddress = nullptr;

DWORD guardPageOldProtectFlags = PAGE\_NOACCESS;

if (heapInfo->pageHeapMode == PageHeapMode::PageHeapModeBlockStart)

{

address = baseAddress + AutoSystemInfo::PageSize;

guardPageAddress = baseAddress;

}

else if (heapInfo->pageHeapMode == PageHeapMode::PageHeapModeBlockEnd)

{

address = baseAddress;

guardPageAddress = baseAddress + pageCount\* AutoSystemInfo::PageSize;

}

else

{

AnalysisAssert(false);

}

if (::VirtualProtect(static\_cast<LPVOID>(guardPageAddress), AutoSystemInfo::PageSize, PAGE\_NOACCESS, &guardPageOldProtectFlags) == FALSE)

{

AssertMsg(false, "Unable to set permission for guard page.");

return nullptr;

}

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

recycler->VerifyZeroFill(address, pageCount \* AutoSystemInfo::PageSize);

#endif

LargeHeapBlock \* heapBlock = LargeHeapBlock::New(address, pageCount, segment, 1, nullptr);

if (!heapBlock)

{

recycler->GetRecyclerLargeBlockPageAllocator()->SuspendIdleDecommit();

recycler->GetRecyclerLargeBlockPageAllocator()->Release(address, actualPageCount, segment);

recycler->GetRecyclerLargeBlockPageAllocator()->ResumeIdleDecommit();

return nullptr;

}

heapBlock->actualPageCount = actualPageCount;

heapBlock->guardPageAddress = guardPageAddress;

heapBlock->guardPageOldProtectFlags = guardPageOldProtectFlags;

heapBlock->pageHeapMode = heapInfo->pageHeapMode;

if (heapBlock->pageHeapMode == PageHeapMode::PageHeapModeBlockEnd)

{

// TODO: pad the address to close-most to the guard page to increase the chance to hit guard page when overflow

// some Mark code need to be updated to support this

// heapBlock->SetEndAllocAddress(address

// + AutoSystemInfo::PageSize - (((AllocSizeMath::Add(sizeCat, sizeof(LargeObjectHeader)) - 1) % AutoSystemInfo::PageSize) / HeapInfo::ObjectGranularity + 1) \* HeapInfo::ObjectGranularity);

}

#if DBG

LargeAllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Allocated new large heap block 0x%p for sizeCat 0x%x\n", heapBlock, sizeCat);

#endif

#ifdef ENABLE\_JS\_ETW

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (segment->GetPageCount() > recycler->GetRecyclerLargeBlockPageAllocator()->GetMaxAllocPageCount())

{

EventWriteJSCRIPT\_INTERNAL\_RECYCLER\_EXTRALARGE\_OBJECT\_ALLOC(size);

}

#endif

#endif

#ifdef PARTIAL\_GC\_ENABLED

recycler->autoHeap.uncollectedNewPageCount += pageCount;

#endif

RECYCLER\_SLOW\_CHECK(this->heapInfo->heapBlockCount[HeapBlock::HeapBlockType::LargeBlockType]++);

heapBlock->heapInfo = this->heapInfo;

Assert(recycler->collectionState != CollectionStateMark);

if (!recycler->heapBlockMap.SetHeapBlock(address, pageCount, heapBlock, HeapBlock::HeapBlockType::LargeBlockType, 0))

{

recycler->GetRecyclerLargeBlockPageAllocator()->SuspendIdleDecommit();

heapBlock->ReleasePages<true>(recycler);

recycler->GetRecyclerLargeBlockPageAllocator()->ResumeIdleDecommit();

LargeHeapBlock::Delete(heapBlock);

RECYCLER\_SLOW\_CHECK(this->heapInfo->heapBlockCount[HeapBlock::HeapBlockType::LargeBlockType]--);

return nullptr;

}

heapBlock->ResetMarks(ResetMarkFlags\_None, recycler);

if (this->largePageHeapBlockList)

{

HeapBlockList::Tail(this->largePageHeapBlockList)->SetNextBlock(heapBlock);

}

else

{

this->largePageHeapBlockList = heapBlock;

}

RECYCLER\_PERF\_COUNTER\_ADD(FreeObjectSize, heapBlock->GetPageCount() \* AutoSystemInfo::PageSize);

char \* memBlock = heapBlock->Alloc(sizeCat, attributes);

Assert(memBlock != nullptr);

if (recycler->ShouldCapturePageHeapAllocStack())

{

heapBlock->CapturePageHeapAllocStack();

}

return memBlock;

}

#endif

LargeHeapBlock\*

LargeHeapBucket::AddLargeHeapBlock(size\_t size, bool nothrow)

{

Recycler\* recycler = this->heapInfo->recycler;

Segment \* segment;

size\_t pageCount = LargeHeapBlock::GetPagesNeeded(size, this->supportFreeList);

if (pageCount == 0)

{

if (nothrow == false)

{

// overflow

// Since nothrow is false here, it's okay to throw

recycler->OutOfMemory();

}

return nullptr;

}

char \* address = nullptr;

address = recycler->GetRecyclerLargeBlockPageAllocator()->Alloc(&pageCount, &segment);

if (address == nullptr)

{

return nullptr;

}

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

recycler->VerifyZeroFill(address, pageCount \* AutoSystemInfo::PageSize);

#endif

uint objectCount = LargeHeapBlock::GetMaxLargeObjectCount(pageCount, size);

LargeHeapBlock \* heapBlock = LargeHeapBlock::New(address, pageCount, segment, objectCount, supportFreeList ? this : nullptr);

#if DBG

LargeAllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Allocated new large heap block 0x%p for sizeCat 0x%x\n", heapBlock, sizeCat);

#endif

#ifdef ENABLE\_JS\_ETW

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (segment->GetPageCount() > recycler->GetRecyclerLargeBlockPageAllocator()->GetMaxAllocPageCount())

{

EventWriteJSCRIPT\_INTERNAL\_RECYCLER\_EXTRALARGE\_OBJECT\_ALLOC(size);

}

#endif

#endif

if (!heapBlock)

{

recycler->GetRecyclerLargeBlockPageAllocator()->SuspendIdleDecommit();

recycler->GetRecyclerLargeBlockPageAllocator()->Release(address, pageCount, segment);

recycler->GetRecyclerLargeBlockPageAllocator()->ResumeIdleDecommit();

return nullptr;

}

#ifdef PARTIAL\_GC\_ENABLED

recycler->autoHeap.uncollectedNewPageCount += pageCount;

#endif

RECYCLER\_SLOW\_CHECK(this->heapInfo->heapBlockCount[HeapBlock::HeapBlockType::LargeBlockType]++);

heapBlock->heapInfo = this->heapInfo;

heapBlock->lastCollectAllocCount = 0;

Assert(recycler->collectionState != CollectionStateMark);

if (!recycler->heapBlockMap.SetHeapBlock(address, pageCount, heapBlock, HeapBlock::HeapBlockType::LargeBlockType, 0))

{

recycler->GetRecyclerLargeBlockPageAllocator()->SuspendIdleDecommit();

heapBlock->ReleasePages<false>(recycler);

recycler->GetRecyclerLargeBlockPageAllocator()->ResumeIdleDecommit();

LargeHeapBlock::Delete(heapBlock);

RECYCLER\_SLOW\_CHECK(this->heapInfo->heapBlockCount[HeapBlock::HeapBlockType::LargeBlockType]--);

return nullptr;

}

heapBlock->SetNextBlock(this->largeBlockList);

this->largeBlockList = heapBlock;

RECYCLER\_PERF\_COUNTER\_ADD(FreeObjectSize, heapBlock->GetPageCount() \* AutoSystemInfo::PageSize);

return heapBlock;

}

char \*

LargeHeapBucket::TryAllocFromFreeList(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

LargeHeapBlockFreeList\* freeListEntry = this->freeList;

// Walk through the free list, find the first entry that can fit our desired size

while (freeListEntry)

{

LargeHeapBlock\* heapBlock = freeListEntry->heapBlock;

char \* memBlock = heapBlock->TryAllocFromFreeList(sizeCat, attributes);

if (memBlock)

{

// Don't need to verify zero fill here since we will do it in LargeHeapBucket::Alloc

return memBlock;

}

else

{

#if DBG

LargeAllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Unable to allocate object of size 0x%x from freelist\n", sizeCat);

#endif

}

freeListEntry = freeListEntry->next;

}

return nullptr;

}

char \*

LargeHeapBucket::TryAllocFromExplicitFreeList(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

FreeObject \* currFreeObject = this->explicitFreeList;

FreeObject \* prevFreeObject = nullptr;

while (currFreeObject != nullptr)

{

char \* memBlock = (char \*)currFreeObject;

LargeObjectHeader \* header = LargeHeapBlock::GetHeaderFromAddress(memBlock);

Assert(header->isExplicitFreed);

Assert(HeapInfo::GetMediumObjectAlignedSizeNoCheck(header->objectSize) == this->sizeCat);

if (header->objectSize < sizeCat)

{

prevFreeObject = currFreeObject;

currFreeObject = currFreeObject->GetNext();

continue;

}

DebugOnly(header->isExplicitFreed = false);

if (prevFreeObject)

{

prevFreeObject->SetNext(currFreeObject->GetNext());

}

else

{

this->explicitFreeList = currFreeObject->GetNext();

}

#ifdef RECYCLER\_MEMORY\_VERIFY

HeapBlock\* heapBlock = recycler->FindHeapBlock(memBlock);

Assert(heapBlock != nullptr);

Assert(heapBlock->IsLargeHeapBlock());

LargeHeapBlock \* largeHeapBlock = (LargeHeapBlock \*)heapBlock;

LargeObjectHeader \* dbgHeader;

Assert(largeHeapBlock->GetObjectHeader(memBlock, &dbgHeader));

Assert(dbgHeader == header);

((FreeObject \*)memBlock)->DebugFillNext();

#endif

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

// TODO: large heap block doesn't separate leaf object on to different page allocator.

// so all the memory should still be zeroed.

memset(memBlock, 0, sizeof(FreeObject));

#endif

header->SetAttributes(recycler->Cookie, (attributes & StoredObjectInfoBitMask));

if ((attributes & ObjectInfoBits::FinalizeBit) != 0)

{

LargeHeapBlock\* heapBlock = (LargeHeapBlock \*)recycler->FindHeapBlock(memBlock);

heapBlock->finalizeCount++;

#ifdef RECYCLER\_FINALIZE\_CHECK

heapInfo->liveFinalizableObjectCount++;

heapInfo->newFinalizableObjectCount++;

#endif

}

return memBlock;

}

return nullptr;

}

//=====================================================================================================

// Free

//=====================================================================================================

void

LargeHeapBucket::ExplicitFree(void \* object, size\_t sizeCat)

{

Assert(HeapInfo::GetMediumObjectAlignedSizeNoCheck(sizeCat) == this->sizeCat);

LargeObjectHeader \* header = LargeHeapBlock::GetHeaderFromAddress(object);

Assert(header->GetAttributes(this->heapInfo->recycler->Cookie) == ObjectInfoBits::NoBit || header->GetAttributes(this->heapInfo->recycler->Cookie) == ObjectInfoBits::LeafBit);

Assert(!header->isExplicitFreed);

DebugOnly(header->isExplicitFreed = true);

Assert(header->objectSize >= sizeCat);

#if DBG

HeapBlock\* heapBlock = this->GetRecycler()->FindHeapBlock(object);

Assert(heapBlock != nullptr);

Assert(heapBlock->IsLargeHeapBlock());

LargeHeapBlock \* largeHeapBlock = (LargeHeapBlock \*)heapBlock;

LargeObjectHeader \* dbgHeader;

Assert(largeHeapBlock->GetObjectHeader(object, &dbgHeader));

Assert(dbgHeader == header);

#endif

FreeObject \* freeObject = (FreeObject \*)object;

freeObject->SetNext(this->explicitFreeList);

this->explicitFreeList = freeObject;

header->SetAttributes(this->heapInfo->recycler->Cookie, ObjectInfoBits::LeafBit); // We can stop scanning it now.

}

//=====================================================================================================

// Collections

//=====================================================================================================

void

LargeHeapBucket::ResetMarks(ResetMarkFlags flags)

{

Recycler\* recycler = this->heapInfo->recycler;

HeapBlockList::ForEach(largeBlockList, [flags, recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ResetMarks(flags, recycler);

});

#ifdef RECYCLER\_PAGE\_HEAP

HeapBlockList::ForEach(largePageHeapBlockList, [flags, recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ResetMarks(flags, recycler);

});

#endif

HeapBlockList::ForEach(fullLargeBlockList, [flags, recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ResetMarks(flags, recycler);

});

HeapBlockList::ForEach(pendingDisposeLargeBlockList, [flags, recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ResetMarks(flags, recycler);

});

#ifdef CONCURRENT\_GC\_ENABLED

Assert(pendingSweepLargeBlockList == nullptr);

#endif

}

void

LargeHeapBucket::ScanInitialImplicitRoots(Recycler \* recycler)

{

HeapBlockList::ForEach(largeBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#ifdef RECYCLER\_PAGE\_HEAP

HeapBlockList::ForEach(largePageHeapBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#endif

HeapBlockList::ForEach(fullLargeBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

HeapBlockList::ForEach(pendingDisposeLargeBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#ifdef CONCURRENT\_GC\_ENABLED

Assert(pendingSweepLargeBlockList == nullptr);

#endif

}

void

LargeHeapBucket::ScanNewImplicitRoots(Recycler \* recycler)

{

HeapBlockList::ForEach(largeBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

#ifdef RECYCLER\_PAGE\_HEAP

HeapBlockList::ForEach(largePageHeapBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

#endif

HeapBlockList::ForEach(fullLargeBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

HeapBlockList::ForEach(pendingDisposeLargeBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->ScanNewImplicitRoots(recycler);

});

#ifdef CONCURRENT\_GC\_ENABLED

Assert(pendingSweepLargeBlockList == nullptr);

#endif

}

//=====================================================================================================

// Sweep

//=====================================================================================================

#pragma region Sweep

template void LargeHeapBucket::Sweep<true>(RecyclerSweep& recyclerSweep);

template void LargeHeapBucket::Sweep<false>(RecyclerSweep& recyclerSweep);

template<bool pageheap>

void

LargeHeapBucket::Sweep(RecyclerSweep& recyclerSweep)

{

// CONCURRENT-TODO: large buckets are not swept in the background currently.

Assert(!recyclerSweep.GetRecycler()->IsConcurrentExecutingState());

LargeHeapBlock \* currentLargeObjectBlocks = largeBlockList;

#ifdef RECYCLER\_PAGE\_HEAP

LargeHeapBlock \* currentLargePageHeapObjectBlocks = largePageHeapBlockList;

#endif

LargeHeapBlock \* currentFullLargeObjectBlocks = fullLargeBlockList;

LargeHeapBlock \* currentDisposeLargeBlockList = pendingDisposeLargeBlockList;

this->largeBlockList = nullptr;

#ifdef RECYCLER\_PAGE\_HEAP

this->largePageHeapBlockList = nullptr;

#endif

this->fullLargeBlockList = nullptr;

// Clear the free list before sweep

// We'll reconstruct the free list during sweep

if (this->supportFreeList)

{

#if DBG

LargeAllocationVerboseTrace(recyclerSweep.GetRecycler()->GetRecyclerFlagsTable(), L"Resetting free list for 0x%x bucket\n", this->sizeCat);

#endif

this->freeList = nullptr;

this->explicitFreeList = nullptr;

}

#ifdef CONCURRENT\_GC\_ENABLED

Assert(this->pendingSweepLargeBlockList == nullptr);

#endif

SweepLargeHeapBlockList<pageheap>(recyclerSweep, currentLargeObjectBlocks);

#ifdef RECYCLER\_PAGE\_HEAP

SweepLargeHeapBlockList<pageheap>(recyclerSweep, currentLargePageHeapObjectBlocks);

#endif

SweepLargeHeapBlockList<pageheap>(recyclerSweep, currentFullLargeObjectBlocks);

SweepLargeHeapBlockList<pageheap>(recyclerSweep, currentDisposeLargeBlockList);

}

template<bool pageheap>

void

LargeHeapBucket::SweepLargeHeapBlockList(RecyclerSweep& recyclerSweep, LargeHeapBlock \* heapBlockList)

{

Recycler \* recycler = recyclerSweep.GetRecycler();

HeapBlockList::ForEachEditing(heapBlockList, [this, &recyclerSweep, recycler](LargeHeapBlock \* heapBlock)

{

this->UnregisterFreeList(heapBlock->GetFreeList());

// CONCURRENT-TODO: Allow large block to be sweep in the background

SweepState state = heapBlock->Sweep<pageheap>(recyclerSweep, false);

// If the block is already in the pending dispose list (re-entrant GC scenario), do nothing, leave it there

if (heapBlock->IsInPendingDisposeList()) return;

switch (state)

{

case SweepStateEmpty:

heapBlock->ReleasePagesSweep<pageheap>(recycler);

LargeHeapBlock::Delete(heapBlock);

RECYCLER\_SLOW\_CHECK(this->heapInfo->heapBlockCount[HeapBlock::HeapBlockType::LargeBlockType]--);

break;

case SweepStateFull:

heapBlock->SetNextBlock(this->fullLargeBlockList);

this->fullLargeBlockList = heapBlock;

break;

case SweepStateSwept:

if (supportFreeList)

{

ConstructFreelist(heapBlock);

}

else

{

ReinsertLargeHeapBlock(heapBlock);

}

break;

case SweepStatePendingDispose:

Assert(!recyclerSweep.IsBackground());

Assert(!recycler->hasPendingTransferDisposedObjects);

heapBlock->SetNextBlock(this->pendingDisposeLargeBlockList);

this->pendingDisposeLargeBlockList = heapBlock;

heapBlock->SetIsInPendingDisposeList(true);

#if DBG

heapBlock->SetHasDisposeBeenCalled(false);

#endif

recycler->hasDisposableObject = true;

break;

#ifdef CONCURRENT\_GC\_ENABLED

case SweepStatePendingSweep:

heapBlock->SetNextBlock(this->pendingSweepLargeBlockList);

this->pendingSweepLargeBlockList = heapBlock;

break;

#endif

}

});

}

void

LargeHeapBucket::ReinsertLargeHeapBlock(LargeHeapBlock \* heapBlock)

{

Assert(!heapBlock->hasPartialFreeObjects);

Assert(!heapBlock->IsInPendingDisposeList());

if (this->largeBlockList != nullptr && heapBlock->GetFreeSize() > this->largeBlockList->GetFreeSize())

{

heapBlock->SetNextBlock(this->largeBlockList->GetNextBlock());

this->largeBlockList->SetNextBlock(this->fullLargeBlockList);

this->fullLargeBlockList = this->largeBlockList;

this->largeBlockList = heapBlock;

}

else

{

heapBlock->SetNextBlock(this->fullLargeBlockList);

this->fullLargeBlockList = heapBlock;

}

}

void

LargeHeapBucket::RegisterFreeList(LargeHeapBlockFreeList\* freeList)

{

Assert(freeList->next == nullptr);

Assert(freeList->previous == nullptr);

LargeHeapBlockFreeList\* head = this->freeList;

if (head)

{

head->previous = freeList;

}

freeList->next = head;

this->freeList = freeList;

}

void

LargeHeapBucket::UnregisterFreeList(LargeHeapBlockFreeList\* freeList)

{

LargeHeapBlockFreeList\* next = freeList->next;

LargeHeapBlockFreeList\* previous = freeList->previous;

if (previous)

{

previous->next = next;

}

if (next)

{

next->previous = previous;

}

freeList->next = nullptr;

freeList->previous = nullptr;

if (freeList == this->freeList)

{

this->freeList = next;

}

}

void

LargeHeapBucket::ConstructFreelist(LargeHeapBlock \* heapBlock)

{

Assert(!heapBlock->hasPartialFreeObjects);

Assert(!heapBlock->IsInPendingDisposeList());

// The free list is the only way we reuse heap block entries

// so if the heap block is allocated from directly, it'll not

// invalidate the free list

LargeHeapBlockFreeList\* freeList = heapBlock->GetFreeList();

Assert(freeList);

if (freeList->entries)

{

this->RegisterFreeList(freeList);

#if DBG

LargeAllocationVerboseTrace(this->GetRecycler()->GetRecyclerFlagsTable(), L"Free list created for 0x%x bucket\n", this->sizeCat);

#endif

}

ReinsertLargeHeapBlock(heapBlock);

}

#pragma endregion

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t

LargeHeapBucket::Rescan(LargeHeapBlock \* list, Recycler \* recycler, bool isPartialSwept, RescanFlags flags)

{

size\_t scannedPageCount = 0;

HeapBlockList::ForEach(list, [recycler, isPartialSwept, flags, &scannedPageCount](LargeHeapBlock \* heapBlock)

{

scannedPageCount += heapBlock->Rescan(recycler, isPartialSwept, flags);

});

return scannedPageCount;

}

size\_t

LargeHeapBucket::Rescan(RescanFlags flags)

{

#ifdef CONCURRENT\_GC\_ENABLED

Assert(pendingSweepLargeBlockList == nullptr);

#endif

size\_t scannedPageCount = 0;

Recycler\* recycler = this->heapInfo->recycler;

scannedPageCount += LargeHeapBucket::Rescan(largeBlockList, recycler, false, flags);

#ifdef RECYCLER\_PAGE\_HEAP

scannedPageCount += LargeHeapBucket::Rescan(largePageHeapBlockList, recycler, false, flags);

#endif

scannedPageCount += LargeHeapBucket::Rescan(fullLargeBlockList, recycler, false, flags);

scannedPageCount += LargeHeapBucket::Rescan(pendingDisposeLargeBlockList, recycler, true, flags);

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

Assert(recycler->inPartialCollectMode || partialSweptLargeBlockList == nullptr);

if (recycler->inPartialCollectMode)

{

scannedPageCount += LargeHeapBucket::Rescan(partialSweptLargeBlockList, recycler, true, flags);

}

#endif

return scannedPageCount;

}

void

LargeHeapBucket::SweepPendingObjects(RecyclerSweep& recyclerSweep)

{

if (recyclerSweep.IsBackground())

{

Recycler \* recycler = recyclerSweep.GetRecycler();

if (recycler->inPartialCollectMode)

{

HeapBlockList::ForEach(this->pendingSweepLargeBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

// Page heap blocks are never swept concurrently

heapBlock->SweepObjects<false, SweepMode\_ConcurrentPartial>(recycler);

});

}

else

{

HeapBlockList::ForEach(this->pendingSweepLargeBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

// Page heap blocks are never swept concurrently

heapBlock->SweepObjects<false, SweepMode\_Concurrent>(recycler);

});

}

}

else

{

Assert(this->pendingSweepLargeBlockList == nullptr);

}

}

#ifdef PARTIAL\_GC\_ENABLED

void

LargeHeapBucket::ConcurrentPartialTransferSweptObjects(RecyclerSweep& recyclerSweep)

{

Assert(recyclerSweep.InPartialCollectMode());

Assert(!recyclerSweep.IsBackground());

RECYCLER\_SLOW\_CHECK(this->VerifyLargeHeapBlockCount());

LargeHeapBlock \* list = this->pendingSweepLargeBlockList;

this->pendingSweepLargeBlockList = nullptr;

HeapBlockList::ForEachEditing(list, [this](LargeHeapBlock \* heapBlock)

{

// GC-REVIEW: We could maybe reuse the large objects

heapBlock->PartialTransferSweptObjects();

heapBlock->SetNextBlock(this->partialSweptLargeBlockList);

this->partialSweptLargeBlockList = heapBlock;

});

RECYCLER\_SLOW\_CHECK(this->VerifyLargeHeapBlockCount());

}

void

LargeHeapBucket::FinishPartialCollect(RecyclerSweep \* recyclerSweep)

{

#ifdef CONCURRENT\_GC\_ENABLED

Recycler\* recycler = this->heapInfo->recycler;

if (recyclerSweep && recyclerSweep->IsBackground())

{

// Leave it in the partialSweptLargeBlockList if we are processing it in the background

// ConcurrentTransferSweptObjects will put it back.

HeapBlockList::ForEachEditing(partialSweptLargeBlockList, [this, recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->FinishPartialCollect(recycler);

});

}

else

{

HeapBlockList::ForEachEditing(partialSweptLargeBlockList, [this, recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->FinishPartialCollect(recycler);

this->ReinsertLargeHeapBlock(heapBlock);

});

this->partialSweptLargeBlockList = nullptr;

}

#endif

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

void

LargeHeapBucket::ConcurrentTransferSweptObjects(RecyclerSweep& recyclerSweep)

{

Assert(!recyclerSweep.InPartialCollectMode());

Assert(!recyclerSweep.IsBackground());

HeapBlockList::ForEachEditing(this->pendingSweepLargeBlockList, [this](LargeHeapBlock \* heapBlock)

{

heapBlock->TransferSweptObjects();

ReinsertLargeHeapBlock(heapBlock);

});

this->pendingSweepLargeBlockList = nullptr;

#ifdef PARTIAL\_GC\_ENABLED

// If we did a background finish partial collect, we have left the partialSweptLargeBlockList

// there because can't reinsert the heap block in the background, do it here now.

HeapBlockList::ForEachEditing(this->partialSweptLargeBlockList, [this](LargeHeapBlock \* heapBlock)

{

ReinsertLargeHeapBlock(heapBlock);

});

this->partialSweptLargeBlockList = nullptr;

#endif

}

#endif

#endif

void

LargeHeapBucket::FinalizeAllObjects()

{

ForEachLargeHeapBlock([](LargeHeapBlock \* heapBlock) { heapBlock->FinalizeAllObjects(); });

}

void

LargeHeapBucket::Finalize(Recycler \* recycler, LargeHeapBlock \* heapBlockList)

{

HeapBlockList::ForEachEditing(heapBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->FinalizeObjects(recycler);

});

}

void

LargeHeapBucket::Finalize()

{

Recycler\* recycler = this->heapInfo->recycler;

// Finalize any free objects in the non-filled large heap blocks

Finalize(recycler, largeBlockList);

#ifdef RECYCLER\_PAGE\_HEAP

Finalize(recycler, largePageHeapBlockList);

#endif

// Finalize any free objects in the filled large heap blocks

Finalize(recycler, fullLargeBlockList);

// Finalize any free objects in the large heap blocks which have objects pending dispose

// This is to handle the case where if during dispose, a GC is triggered, we might have

// found more objects to free. These objects might reside in a block that was moved to the

// pendingDisposeLargeBlockList during the outer GC. So we need to walk through this list

// again and finalize any objects that need to be finalized. If we don't, they would

// not get finalized

Finalize(recycler, pendingDisposeLargeBlockList);

}

void

LargeHeapBucket::DisposeObjects()

{

Recycler \* recycler = this->heapInfo->recycler;

HeapBlockList::ForEach(this->pendingDisposeLargeBlockList, [recycler](LargeHeapBlock \* heapBlock)

{

heapBlock->DisposeObjects(recycler);

});

}

void

LargeHeapBucket::TransferDisposedObjects()

{

Recycler \* recycler = this->heapInfo->recycler;

Assert(!recycler->IsConcurrentExecutingState());

HeapBlockList::ForEachEditing(this->pendingDisposeLargeBlockList, [this](LargeHeapBlock \* heapBlock)

{

/\* GC-TODO: large heap block doesn't support free list yet \*/

heapBlock->SetIsInPendingDisposeList(false);

ReinsertLargeHeapBlock(heapBlock);

});

this->pendingDisposeLargeBlockList = nullptr;

}

void

LargeHeapBucket::EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size))

{

HeapBucket::EnumerateObjects(largeBlockList, infoBits, CallBackFunction);

#ifdef RECYCLER\_PAGE\_HEAP

HeapBucket::EnumerateObjects(largePageHeapBlockList, infoBits, CallBackFunction);

#endif

HeapBucket::EnumerateObjects(fullLargeBlockList, infoBits, CallBackFunction);

// Pending dispose large block list need not be null

// When we enumerate over this list, anything that has been swept/finalized won't be

// enumerated since it needs to have the object header for enumeration

// and we set the header to null upon sweep/finalize

HeapBucket::EnumerateObjects(pendingDisposeLargeBlockList, infoBits, CallBackFunction);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(this->pendingSweepLargeBlockList == nullptr);

#ifdef PARTIAL\_GC\_ENABLED

HeapBucket::EnumerateObjects(partialSweptLargeBlockList, infoBits, CallBackFunction);

#endif

#endif

}

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t

LargeHeapBucket::GetLargeHeapBlockCount(bool checkCount) const

{

size\_t currentLargeHeapBlockCount = HeapBlockList::Count(fullLargeBlockList);

currentLargeHeapBlockCount += HeapBlockList::Count(largeBlockList);

#ifdef RECYCLER\_PAGE\_HEAP

currentLargeHeapBlockCount += HeapBlockList::Count(largePageHeapBlockList);

#endif

currentLargeHeapBlockCount += HeapBlockList::Count(pendingDisposeLargeBlockList);

#ifdef CONCURRENT\_GC\_ENABLED

currentLargeHeapBlockCount += HeapBlockList::Count(pendingSweepLargeBlockList);

#ifdef PARTIAL\_GC\_ENABLED

currentLargeHeapBlockCount += HeapBlockList::Count(partialSweptLargeBlockList);

#endif

#endif

return currentLargeHeapBlockCount;

}

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t

LargeHeapBucket::Check()

{

size\_t currentLargeHeapBlockCount = Check(false, false, largeBlockList);

#ifdef RECYCLER\_PAGE\_HEAP

currentLargeHeapBlockCount += Check(true, false, largePageHeapBlockList);

#endif

currentLargeHeapBlockCount += Check(true, false, fullLargeBlockList);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(pendingSweepLargeBlockList == nullptr);

#ifdef PARTIAL\_GC\_ENABLED

currentLargeHeapBlockCount += Check(false, false, partialSweptLargeBlockList);

#endif

#endif

currentLargeHeapBlockCount += Check(false, true, pendingDisposeLargeBlockList);

return currentLargeHeapBlockCount;

}

template <typename TBlockType>

size\_t

LargeHeapBucket::Check(bool expectFull, bool expectPending, TBlockType \* list, TBlockType \* tail)

{

size\_t heapBlockCount = 0;

HeapBlockList::ForEach(list, tail, [&heapBlockCount, expectFull, expectPending](TBlockType \* heapBlock)

{

heapBlock->Check(expectFull, expectPending);

heapBlockCount++;

});

return heapBlockCount;

}

template size\_t LargeHeapBucket::Check<LargeHeapBlock>(bool expectFull, bool expectPending, LargeHeapBlock \* list, LargeHeapBlock \* tail);

void

LargeHeapBucket::VerifyLargeHeapBlockCount()

{

GetLargeHeapBlockCount(true);

}

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void

LargeHeapBucket::Verify()

{

Recycler \* recycler = this->heapInfo->recycler;

HeapBlockList::ForEach(largeBlockList, [recycler](LargeHeapBlock \* largeHeapBlock)

{

largeHeapBlock->Verify(recycler);

});

#ifdef RECYCLER\_PAGE\_HEAP

HeapBlockList::ForEach(largePageHeapBlockList, [recycler](LargeHeapBlock \* largeHeapBlock)

{

largeHeapBlock->Verify(recycler);

});

#endif

HeapBlockList::ForEach(fullLargeBlockList, [recycler](LargeHeapBlock \* largeHeapBlock)

{

largeHeapBlock->Verify(recycler);

});

HeapBlockList::ForEach(pendingDisposeLargeBlockList, [recycler](LargeHeapBlock \* largeHeapBlock)

{

largeHeapBlock->Verify(recycler);

});

}

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void

LargeHeapBucket::VerifyMark()

{

HeapBlockList::ForEach(largeBlockList, [](LargeHeapBlock \* largeHeapBlock)

{

largeHeapBlock->VerifyMark();

});

HeapBlockList::ForEach(fullLargeBlockList, [](LargeHeapBlock \* largeHeapBlock)

{

largeHeapBlock->VerifyMark();

});

HeapBlockList::ForEach(pendingDisposeLargeBlockList, [](LargeHeapBlock \* largeHeapBlock)

{

largeHeapBlock->VerifyMark();

});

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

class HeapInfo;

class RecyclerSweep;

// CONSIDER: Templatizing this so that we can have separate leaf large allocations

// and finalizable allocations

// CONSIDER: Templatizing this so that we don't have free list support if we don't need it

class LargeHeapBucket: public HeapBucket

{

public:

LargeHeapBucket():

supportFreeList(false),

freeList(nullptr),

explicitFreeList(nullptr),

fullLargeBlockList(nullptr),

largeBlockList(nullptr),

#ifdef RECYCLER\_PAGE\_HEAP

largePageHeapBlockList(nullptr),

#endif

pendingDisposeLargeBlockList(nullptr)

#ifdef CONCURRENT\_GC\_ENABLED

, pendingSweepLargeBlockList(nullptr)

#endif

#ifdef PARTIAL\_GC\_ENABLED

, partialSweptLargeBlockList(nullptr)

#endif

{

}

~LargeHeapBucket();

void Initialize(HeapInfo \* heapInfo, uint sizeCat, bool supportFreeList = false);

LargeHeapBlock\* AddLargeHeapBlock(size\_t size, bool nothrow);

template <ObjectInfoBits attributes, bool nothrow>

char\* Alloc(Recycler \* recycler, size\_t sizeCat);

#ifdef RECYCLER\_PAGE\_HEAP

char \*PageHeapAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes, PageHeapMode mode, bool nothrow);

#endif

void ExplicitFree(void \* object, size\_t sizeCat);

void ResetMarks(ResetMarkFlags flags);

void ScanInitialImplicitRoots(Recycler \* recycler);

void ScanNewImplicitRoots(Recycler \* recycler);

template<bool pageheap>

void Sweep(RecyclerSweep& recyclerSweep);

void ReinsertLargeHeapBlock(LargeHeapBlock \* heapBlock);

void RegisterFreeList(LargeHeapBlockFreeList\* freeList);

void UnregisterFreeList(LargeHeapBlockFreeList\* freeList);

void FinalizeAllObjects();

void Finalize();

void DisposeObjects();

void TransferDisposedObjects();

void EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size));

void Verify();

void VerifyMark();

void VerifyLargeHeapBlockCount();

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t Rescan(RescanFlags flags);

void SweepPendingObjects(RecyclerSweep& recyclerSweep);

#ifdef PARTIAL\_GC\_ENABLED

void FinishPartialCollect(RecyclerSweep \* recyclerSweep);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

void ConcurrentTransferSweptObjects(RecyclerSweep& recyclerSweep);

#ifdef PARTIAL\_GC\_ENABLED

void ConcurrentPartialTransferSweptObjects(RecyclerSweep& recyclerSweep);

#endif

#endif

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t GetLargeHeapBlockCount(bool checkCount = false) const;

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t Check();

template <typename TBlockType>

size\_t Check(bool expectFull, bool expectPending, TBlockType \* list, TBlockType \* tail = nullptr);

#endif

#endif

private:

char \* SnailAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes, bool nothrow);

char \* TryAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes);

char \* TryAllocFromNewHeapBlock(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes, bool nothrow);

char \* TryAllocFromFreeList(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes);

char \* TryAllocFromExplicitFreeList(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes);

template <class Fn> void ForEachLargeHeapBlock(Fn fn);

template <class Fn> void ForEachEditingLargeHeapBlock(Fn fn);

void Finalize(Recycler\* recycler, LargeHeapBlock\* heapBlock);

template<bool pageheap>

void SweepLargeHeapBlockList(RecyclerSweep& recyclerSweep, LargeHeapBlock \* heapBlockList);

void ConstructFreelist(LargeHeapBlock \* heapBlock);

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t Rescan(LargeHeapBlock \* list, Recycler \* recycler, bool isPartialSwept, RescanFlags flags);

#endif

LargeHeapBlock \* fullLargeBlockList;

LargeHeapBlock \* largeBlockList;

#ifdef RECYCLER\_PAGE\_HEAP

LargeHeapBlock \* largePageHeapBlockList;

#endif

LargeHeapBlock \* pendingDisposeLargeBlockList;

#ifdef CONCURRENT\_GC\_ENABLED

LargeHeapBlock \* pendingSweepLargeBlockList;

#ifdef PARTIAL\_GC\_ENABLED

LargeHeapBlock \* partialSweptLargeBlockList;

#endif

#endif

bool supportFreeList;

LargeHeapBlockFreeList\* freeList;

FreeObject \* explicitFreeList;

friend class HeapInfo;

friend class Recycler;

friend class ScriptMemoryDumper;

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

\_\_inline char \*

LargeHeapBucket::TryAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

char \* memBlock;

// Algorithm:

// Try bump allocate from a heap block

// Otherwise, If free list isn't empty, allocate from it

// \* Otherwise, try allocate from a larger heap block (TODO)

// Otherwise allocate new heap block

if (this->largeBlockList != nullptr)

{

memBlock = this->largeBlockList->Alloc(sizeCat, attributes);

if (memBlock != nullptr)

{

// Don't need to verify zero fill here since we will do it in LargeHeapBucket::Alloc

return memBlock;

}

}

if (!this->supportFreeList)

{

return nullptr;

}

memBlock = this->TryAllocFromExplicitFreeList(recycler, sizeCat, attributes);

if (memBlock != nullptr)

{

// Don't need to verify zero fill here since we will do it in LargeHeapBucket::Alloc

return memBlock;

}

return this->TryAllocFromFreeList(recycler, sizeCat, attributes);

}

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char \*

LargeHeapBucket::Alloc(Recycler \* recycler, size\_t sizeCat)

{

Assert(!HeapInfo::IsMediumObject(sizeCat) || HeapInfo::GetMediumObjectAlignedSizeNoCheck(sizeCat) == this->sizeCat);

Assert((attributes & InternalObjectInfoBitMask) == attributes);

char \* memBlock = TryAlloc(recycler, sizeCat, attributes);

if (memBlock == nullptr)

{

memBlock = SnailAlloc(recycler, sizeCat, attributes, nothrow);

Assert(memBlock != nullptr);

}

else

{

#ifdef RECYCLER\_PAGE\_HEAP

Assert(!IsPageHeapEnabled());

#endif

}

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

// TODO: large heap block doesn't separate leaf object on to different page allocator.

// so all the memory should still be zeroed.

recycler->VerifyZeroFill(memBlock, sizeCat);

#endif

return memBlock;

}

template <class Fn>

void

LargeHeapBucket::ForEachLargeHeapBlock(Fn fn)

{

HeapBlockList::ForEach(fullLargeBlockList, fn);

HeapBlockList::ForEach(largeBlockList, fn);

#ifdef RECYCLER\_PAGE\_HEAP

HeapBlockList::ForEach(largePageHeapBlockList, fn);

#endif

HeapBlockList::ForEach(pendingDisposeLargeBlockList, fn);

#ifdef CONCURRENT\_GC\_ENABLED

HeapBlockList::ForEach(pendingSweepLargeBlockList, fn);

#ifdef PARTIAL\_GC\_ENABLED

HeapBlockList::ForEach(partialSweptLargeBlockList, fn);

#endif

#endif

}

template <class Fn>

void

LargeHeapBucket::ForEachEditingLargeHeapBlock(Fn fn)

{

HeapBlockList::ForEachEditing(fullLargeBlockList, fn);

HeapBlockList::ForEachEditing(largeBlockList, fn);

#ifdef RECYCLER\_PAGE\_HEAP

HeapBlockList::ForEachEditing(largePageHeapBlockList, fn);

#endif

HeapBlockList::ForEachEditing(pendingDisposeLargeBlockList, fn);

#ifdef CONCURRENT\_GC\_ENABLED

HeapBlockList::ForEachEditing(pendingSweepLargeBlockList, fn);

#ifdef PARTIAL\_GC\_ENABLED

HeapBlockList::ForEachEditing(partialSweptLargeBlockList, fn);

#endif

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef LEAK\_REPORT

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAR")

CriticalSection LeakReport::s\_cs;

DWORD LeakReport::nestedSectionCount = 0;

DWORD LeakReport::nestedRedirectOutputCount = 0;

AutoFILE LeakReport::file;

FILE \* oldFile = nullptr;

bool LeakReport::openReportFileFailed = false;

LeakReport::UrlRecord \* LeakReport::urlRecordHead = nullptr;

LeakReport::UrlRecord \* LeakReport::urlRecordTail = nullptr;

void

LeakReport::StartRedirectOutput()

{

if (!EnsureLeakReportFile())

{

return;

}

s\_cs.Enter();

if (nestedRedirectOutputCount == 0)

{

Assert(oldFile == nullptr);

oldFile = Output::SetFile(file);

}

nestedRedirectOutputCount++;

}

void

LeakReport::EndRedirectOutput()

{

if (nestedRedirectOutputCount == 0)

{

return;

}

Assert(file != nullptr);

nestedRedirectOutputCount--;

if (nestedRedirectOutputCount == 0)

{

fflush(file);

FILE \* tmpFile = Output::SetFile(oldFile);

Assert(tmpFile == file);

oldFile = nullptr;

}

s\_cs.Leave();

}

void

LeakReport::StartSection(wchar\_t const \* msg, ...)

{

va\_list argptr;

va\_start(argptr, msg);

StartSection(msg, argptr);

va\_end(argptr);

}

void

LeakReport::StartSection(wchar\_t const \* msg, va\_list argptr)

{

s\_cs.Enter();

if (!EnsureLeakReportFile())

{

return;

}

nestedSectionCount++;

Print(L"--------------------------------------------------------------------------------\n");

vfwprintf(file, msg, argptr);

Print(L"\n");

Print(L"--------------------------------------------------------------------------------\n");

}

void

LeakReport::EndSection()

{

s\_cs.Leave();

if (file == nullptr)

{

return;

}

nestedSectionCount--;

}

void

LeakReport::Print(wchar\_t const \* msg, ...)

{

AutoCriticalSection autocs(&s\_cs);

if (!EnsureLeakReportFile())

{

return;

}

va\_list argptr;

va\_start(argptr, msg);

vfwprintf(file, msg, argptr);

va\_end(argptr);

}

bool

LeakReport::EnsureLeakReportFile()

{

AutoCriticalSection autocs(&s\_cs);

if (openReportFileFailed)

{

return false;

}

if (file != nullptr)

{

return true;

}

wchar\_t const \* filename = Js::Configuration::Global.flags.LeakReport;

wchar\_t const \* openMode = L"w+";

wchar\_t defaultFilename[\_MAX\_PATH];

if (filename == nullptr)

{

swprintf\_s(defaultFilename, L"jsleakreport-%u.txt", ::GetCurrentProcessId());

filename = defaultFilename;

openMode = L"a+"; // append mode

}

if (\_wfopen\_s(&file, filename, openMode) != 0)

{

openReportFileFailed = true;

return false;

}

Print(L"================================================================================\n");

Print(L"Chakra Leak Report - PID: %d\n", ::GetCurrentProcessId());

\_\_time64\_t time\_value = \_time64(NULL);

wchar\_t time\_string[26];

struct tm local\_time;

\_localtime64\_s(&local\_time, &time\_value);

\_wasctime\_s(time\_string, &local\_time);

Print(time\_string);

Print(L"\n");

return true;

}

LeakReport::UrlRecord \*

LeakReport::LogUrl(wchar\_t const \* url, void \* globalObject)

{

UrlRecord \* record = NoCheckHeapNewStruct(UrlRecord);

size\_t length = wcslen(url) + 1; // Add 1 for the NULL.

wchar\_t\* urlCopy = NoCheckHeapNewArray(wchar\_t, length);

js\_memcpy\_s(urlCopy, (length - 1) \* sizeof(wchar\_t), url, (length - 1) \* sizeof(wchar\_t));

urlCopy[length - 1] = L'\0';

record->url = urlCopy;

record->time = \_time64(NULL);

record->tid = ::GetCurrentThreadId();

record->next = nullptr;

record->scriptEngine = nullptr;

record->globalObject = globalObject;

AutoCriticalSection autocs(&s\_cs);

if (LeakReport::urlRecordHead == nullptr)

{

Assert(LeakReport::urlRecordTail == nullptr);

LeakReport::urlRecordHead = record;

LeakReport::urlRecordTail = record;

}

else

{

LeakReport::urlRecordTail->next = record;

LeakReport::urlRecordTail = record;

}

return record;

}

void

LeakReport::DumpUrl(DWORD tid)

{

AutoCriticalSection autocs(&s\_cs);

if (!EnsureLeakReportFile())

{

return;

}

UrlRecord \* prev = nullptr;

UrlRecord \*\* pprev = &LeakReport::urlRecordHead;

UrlRecord \* curr = \*pprev;

while (curr != nullptr)

{

if (curr->tid == tid)

{

wchar\_t timeStr[26];

struct tm local\_time;

\_localtime64\_s(&local\_time, &curr->time);

\_wasctime\_s(timeStr, &local\_time);

timeStr[wcslen(timeStr) - 1] = 0;

Print(L"%s - (%p, %p) %s\n", timeStr, curr->scriptEngine, curr->globalObject, curr->url);

\*pprev = curr->next;

NoCheckHeapDeleteArray(wcslen(curr->url) + 1, curr->url);

NoCheckHeapDelete(curr);

}

else

{

pprev = &curr->next;

prev = curr;

}

curr = \*pprev;

}

if (prev == nullptr)

{

LeakReport::urlRecordTail = nullptr;

}

else if (prev->next == nullptr)

{

LeakReport::urlRecordTail = prev;

}

}

AutoLeakReportSection::AutoLeakReportSection(Js::ConfigFlagsTable& flags, wchar\_t const \* msg, ...):

m\_flags(flags)

{

if (flags.IsEnabled(Js::LeakReportFlag))

{

va\_list argptr;

va\_start(argptr, msg);

LeakReport::StartSection(msg, argptr);

va\_end(argptr);

}

}

AutoLeakReportSection::~AutoLeakReportSection()

{

if (m\_flags.IsEnabled(Js::LeakReportFlag))

{

LeakReport::EndSection();

}

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

#ifdef LEAK\_REPORT

class LeakReport

{

public:

class UrlRecord

{

public:

void \* scriptEngine;

private:

wchar\_t const \* url;

\_\_time64\_t time;

DWORD tid;

UrlRecord \* next;

void \* globalObject;

friend class LeakReport;

};

static void StartRedirectOutput();

static void EndRedirectOutput();

static void StartSection(wchar\_t const \* msg, ...);

static void StartSection(wchar\_t const \* msg, va\_list argptr);

static void EndSection();

static void Print(wchar\_t const \* msg, ...);

static UrlRecord \* LogUrl(wchar\_t const \* url, void \* globalObject);

static void DumpUrl(DWORD tid);

private:

static CriticalSection s\_cs;

static AutoFILE file;

static bool openReportFileFailed;

static DWORD nestedSectionCount;

static DWORD nestedRedirectOutputCount;

static UrlRecord \* urlRecordHead;

static UrlRecord \* urlRecordTail;

static bool EnsureLeakReportFile();

};

class AutoLeakReportSection

{

public:

AutoLeakReportSection(Js::ConfigFlagsTable& flags, wchar\_t const \* msg, ...);

~AutoLeakReportSection();

private:

Js::ConfigFlagsTable& m\_flags;

};

#define STRINGIFY2(x,y) x ## y

#define STRINGIFY(x,y) STRINGIFY2(x,y)

#define LEAK\_REPORT\_PRINT(msg, ...) if (Js::Configuration::Global.flags.IsEnabled(Js::LeakReportFlag)) LeakReport::Print(msg, \_\_VA\_ARGS\_\_)

#define AUTO\_LEAK\_REPORT\_SECTION(flags, msg, ...) AutoLeakReportSection STRINGIFY(\_\_autoLeakReportSection, \_\_COUNTER\_\_)(flags, msg, \_\_VA\_ARGS\_\_)

#else

#define LEAK\_REPORT\_PRINT(msg, ...)

#define AUTO\_LEAK\_REPORT\_SECTION(msg, ...)

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#if defined(\_M\_IX86) || defined(\_M\_X64)

// For prefetch

#include <mmintrin.h>

#endif

MarkContext::MarkContext(Recycler \* recycler, PagePool \* pagePool) :

recycler(recycler),

pagePool(pagePool),

markStack(pagePool),

trackStack(pagePool)

{

}

MarkContext::~MarkContext()

{

#ifdef RECYCLER\_MARK\_TRACK

this->markMap = nullptr;

#endif

}

#ifdef RECYCLER\_MARK\_TRACK

void MarkContext::OnObjectMarked(void\* object, void\* parent)

{

if (!this->markMap->ContainsKey(object))

{

this->markMap->AddNew(object, parent);

}

}

#endif

void MarkContext::Init(uint reservedPageCount)

{

markStack.Init(reservedPageCount);

trackStack.Init();

}

void MarkContext::Clear()

{

markStack.Clear();

trackStack.Clear();

}

void MarkContext::Abort()

{

markStack.Abort();

trackStack.Abort();

pagePool->ReleaseFreePages();

}

void MarkContext::Release()

{

markStack.Release();

trackStack.Release();

pagePool->ReleaseFreePages();

}

uint MarkContext::Split(uint targetCount, \_\_in\_ecount(targetCount) MarkContext \*\* targetContexts)

{

Assert(targetCount > 0 && targetCount <= PageStack<MarkCandidate>::MaxSplitTargets);

\_\_analysis\_assume(targetCount <= PageStack<MarkCandidate>::MaxSplitTargets);

PageStack<MarkCandidate> \* targetStacks[PageStack<MarkCandidate>::MaxSplitTargets];

for (uint i = 0; i < targetCount; i++)

{

targetStacks[i] = &targetContexts[i]->markStack;

}

return this->markStack.Split(targetCount, targetStacks);

}

void MarkContext::ProcessTracked()

{

if (trackStack.IsEmpty())

{

return;

}

FinalizableObject \* trackedObject;

while (trackStack.Pop(&trackedObject))

{

MarkTrackedObject(trackedObject);

}

Assert(trackStack.IsEmpty());

trackStack.Release();

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

namespace Memory

{

class Recycler;

typedef JsUtil::SynchronizedDictionary<void \*, void \*, NoCheckHeapAllocator, PrimeSizePolicy, RecyclerPointerComparer, JsUtil::SimpleDictionaryEntry, Js::DefaultListLockPolicy, CriticalSection> MarkMap;

class MarkContext

{

private:

struct MarkCandidate

{

void \*\* obj;

size\_t byteCount;

};

public:

static const int MarkCandidateSize = sizeof(MarkCandidate);

MarkContext(Recycler \* recycler, PagePool \* pagePool);

~MarkContext();

void Init(uint reservedPageCount);

void Clear();

Recycler \* GetRecycler() { return this->recycler; }

bool AddMarkedObject(void \* obj, size\_t byteCount);

bool AddTrackedObject(FinalizableObject \* obj);

template <bool parallel, bool interior>

void Mark(void \* candidate, void \* parentReference);

template <bool parallel>

void MarkInterior(void \* candidate);

template <bool parallel, bool interior>

void ScanObject(void \*\* obj, size\_t byteCount);

template <bool parallel, bool interior>

void ScanMemory(void \*\* obj, size\_t byteCount);

template <bool parallel, bool interior>

void ProcessMark();

void MarkTrackedObject(FinalizableObject \* obj);

void ProcessTracked();

uint Split(uint targetCount, \_\_in\_ecount(targetCount) MarkContext \*\* targetContexts);

void Abort();

void Release();

bool HasPendingMarkObjects() const { return !markStack.IsEmpty(); }

bool HasPendingTrackObjects() const { return !trackStack.IsEmpty(); }

bool HasPendingObjects() const { return HasPendingMarkObjects() || HasPendingTrackObjects(); }

PageAllocator \* GetPageAllocator() { return this->pagePool->GetPageAllocator(); }

bool IsEmpty()

{

if (HasPendingObjects())

{

return false;

}

Assert(pagePool->IsEmpty());

Assert(!GetPageAllocator()->DisableAllocationOutOfMemory());

return true;

}

#if DBG

void VerifyPostMarkState()

{

Assert(this->markStack.HasChunk());

}

#endif

void Cleanup()

{

Assert(!HasPendingObjects());

Assert(!GetPageAllocator()->DisableAllocationOutOfMemory());

this->pagePool->ReleaseFreePages();

}

void DecommitPages() { this->pagePool->Decommit(); }

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

void SetMaxPageCount(size\_t maxPageCount) { markStack.SetMaxPageCount(maxPageCount); trackStack.SetMaxPageCount(maxPageCount); }

#endif

#ifdef RECYCLER\_MARK\_TRACK

void SetMarkMap(MarkMap\* markMap)

{

this->markMap = markMap;

}

#endif

private:

Recycler \* recycler;

PagePool \* pagePool;

PageStack<MarkCandidate> markStack;

PageStack<FinalizableObject \*> trackStack;

#ifdef RECYCLER\_MARK\_TRACK

MarkMap\* markMap;

void OnObjectMarked(void\* object, void\* parent);

#endif

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

\_\_inline

bool MarkContext::AddMarkedObject(void \* objectAddress, size\_t objectSize)

{

Assert(objectAddress != nullptr);

Assert(objectSize > 0);

Assert(objectSize % sizeof(void \*) == 0);

FAULTINJECT\_MEMORY\_MARK\_NOTHROW(L"AddMarkedObject", objectSize);

#if DBG\_DUMP

if (recycler->forceTraceMark || recycler->GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase))

{

Output::Print(L" %p", objectAddress);

}

#endif

RECYCLER\_STATS\_INTERLOCKED\_INC(recycler, scanCount);

MarkCandidate markCandidate;

markCandidate.obj = (void \*\*) objectAddress;

markCandidate.byteCount = objectSize;

return markStack.Push(markCandidate);

}

\_\_inline

bool MarkContext::AddTrackedObject(FinalizableObject \* obj)

{

Assert(obj != nullptr);

Assert(recycler->DoQueueTrackedObject());

#ifdef PARTIAL\_GC\_ENABLED

Assert(!recycler->inPartialCollectMode);

#endif

FAULTINJECT\_MEMORY\_MARK\_NOTHROW(L"AddTrackedObject", 0);

return trackStack.Push(obj);

}

template <bool parallel, bool interior>

\_\_inline

void MarkContext::ScanMemory(void \*\* obj, size\_t byteCount)

{

Assert(byteCount != 0);

Assert(byteCount % sizeof(void \*) == 0);

void \*\* objEnd = obj + (byteCount / sizeof(void \*));

void \* parentObject = (void\*)obj;

#if DBG\_DUMP

if (recycler->forceTraceMark || recycler->GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase))

{

Output::Print(L"Scanning %p(%8d): ", obj, byteCount);

}

#endif

do

{

// We need to ensure that the compiler does not reintroduce reads to the object after inlining.

// This could cause the value to change after the marking checks (e.g., the null/low address check).

// Intrinsics avoid the expensive memory barrier on ARM (due to /volatile:ms).

#if defined(\_M\_ARM64)

void \* candidate = reinterpret\_cast<void \*>(\_\_iso\_volatile\_load64(reinterpret\_cast<volatile \_\_int64 \*>(obj)));

#elif defined(\_M\_ARM)

void \* candidate = reinterpret\_cast<void \*>(\_\_iso\_volatile\_load32(reinterpret\_cast<volatile \_\_int32 \*>(obj)));

#else

void \* candidate = \*(static\_cast<void \* volatile \*>(obj));

#endif

Mark<parallel, interior>(candidate, parentObject);

obj++;

} while (obj != objEnd);

#if DBG\_DUMP

if (recycler->forceTraceMark || recycler->GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase))

{

Output::Print(L"\n");

Output::Flush();

}

#endif

}

template <bool parallel, bool interior>

\_\_inline

void MarkContext::ScanObject(void \*\* obj, size\_t byteCount)

{

BEGIN\_DUMP\_OBJECT(recycler, obj);

ScanMemory<parallel, interior>(obj, byteCount);

END\_DUMP\_OBJECT(recycler);

}

template <bool parallel, bool interior>

\_\_inline

void MarkContext::Mark(void \* candidate, void \* parentReference)

{

// We should never reach here while we are processing Rescan.

// Otherwise our rescanState could be out of sync with mark state.

Assert(!recycler->isProcessingRescan);

if ((size\_t)candidate < 0x10000)

{

RECYCLER\_STATS\_INTERLOCKED\_INC(recycler, tryMarkNullCount);

return;

}

if (interior)

{

Assert(recycler->enableScanInteriorPointers

|| (!recycler->IsConcurrentState() && recycler->collectionState != CollectionStateParallelMark));

recycler->heapBlockMap.MarkInterior<parallel>(candidate, this);

return;

}

if (!HeapInfo::IsAlignedAddress(candidate))

{

RECYCLER\_STATS\_INTERLOCKED\_INC(recycler, tryMarkUnalignedCount);

return;

}

recycler->heapBlockMap.Mark<parallel>(candidate, this);

#ifdef RECYCLER\_MARK\_TRACK

this->OnObjectMarked(candidate, parentReference);

#endif

}

\_\_inline

void MarkContext::MarkTrackedObject(FinalizableObject \* trackedObject)

{

Assert(!recycler->queueTrackedObject);

#ifdef PARTIAL\_GC\_ENABLED

Assert(!recycler->inPartialCollectMode);

#endif

Assert(!recycler->IsConcurrentExecutingState());

Assert(!(recycler->collectionState == CollectionStateParallelMark));

// Mark is not expected to throw.

BEGIN\_NO\_EXCEPTION

{

trackedObject->Mark(recycler);

}

END\_NO\_EXCEPTION

}

template <bool parallel, bool interior>

\_\_inline

void MarkContext::ProcessMark()

{

#ifdef RECYCLER\_STRESS

if (recycler->GetRecyclerFlagsTable().RecyclerInduceFalsePositives)

{

// InduceFalsePositives logic doesn't support parallel marking

if (!parallel)

{

recycler->heapBlockMap.InduceFalsePositives(recycler);

}

}

#endif

#if defined(\_M\_IX86) || defined(\_M\_X64)

MarkCandidate current, next;

while (markStack.Pop(&current))

{

// Process entries and prefetch as we go.

while (markStack.Pop(&next))

{

// Prefetch the next entry so it's ready when we need it.

\_mm\_prefetch((char \*)next.obj, \_MM\_HINT\_T0);

// Process the previously retrieved entry.

ScanObject<parallel, interior>(current.obj, current.byteCount);

current = next;

}

// The stack is empty, but we still have a previously retrieved entry; process it now.

ScanObject<parallel, interior>(current.obj, current.byteCount);

// Processing that entry may have generated more entries in the mark stack, so continue the loop.

}

#else

// \_mm\_prefetch intrinsic is specific to Intel platforms.

// CONSIDER: There does seem to be a compiler intrinsic for prefetch on ARM,

// however, the information on this is scarce, so for now just don't do prefetch on ARM.

MarkCandidate current;

while (markStack.Pop(&current))

{

ScanObject<parallel, interior>(current.obj, current.byteCount);

}

#endif

Assert(markStack.IsEmpty());

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#ifdef MARKSTACK\_TEST

#define MARKSTACK\_DIAG 1

#endif

#define DEFAULT\_CHUNK\_SIZE 2 MEGABYTES // Reserve in 8 MB chunks

#define DEFAULT\_PAGE\_SIZE 4 KILOBYTES

#define DEFAULT\_COMMIT\_SIZE ((256 KILOBYTES) - DEFAULT\_PAGE\_SIZE) // Commit in 1 MB slices adjusted for the guard page

namespace Helpers

{

void\* CommitMemory(void\* address, DWORD size, DWORD flags) throw();

BOOL DecommitMemory(void\* address, DWORD size) throw();

void\* ReserveMemory(DWORD size, DWORD flags) throw();

BOOL ReleaseMemory(void\* address, DWORD size) throw();

};

//

// Definition of the MarkStack class

// This is a guard-page based stack class that's used for the purposes of marking objects in the Recycler

// By default, it reserves an 8 MB chunk of memory, and commits in 1 MB chunks.

// At the end of 1 MB of memory is a guard page. When we've added 1 MB's worth of entries into the mark stack,

// the stack is moved to the guard page. When we attempt to write to this entry, a trap is triggered.

// At this point, we grow the stack. We then resume execution. Note that an entry is written to the start

// of the guard page when the trap is initially triggered but the rest of the page is left blank.

// When we pop the stack, we \*do\* do a bounds check. If the stack is not at the start, we simply move the stack back

// and return the current value. If the stack is at the start, we then "pop" the chunk. When the chunk is popped,

// the stack points to the start of the guard page.

//

// The stack consists of "Chunks", which is regions of memory that are initially reserved

// When we need more of the chunk, we commit pages within a chunk. If there are no more pages to commit, we allocate a new chunk.

// Chunks are linked through a doubly linked list whose pointers live on the start of the chunk

//

// By default, Chunks are 8MB, and commits are in 1MB ranges.

// There is a preallocated chunk- this means that when the recycler is created, we reserve 8MB for the purpose of the mark stack

// We commit 1MB of this memory up front (so it becomes part of the fixed cost of a Recycler instance)

// The preallocated chunk memory remains reserved for the lifetime of the Recycler

namespace MarkStack

{

struct Chunk

{

Chunk\* next;

Chunk\* previous;

};

struct MarkCandidate

{

void \*\* obj;

size\_t byteCount;

};

template <uint ChunkSize, uint PageSize, uint CommitSize>

class MarkStack

{

public:

\_\_forceinline MarkCandidate\* Push(void\*\* obj, size\_t byteCount) throw()

{

// We first cache the current stack locally

// We then increment the stack pointer- at this point it could be pointing to an address that is past the guard page

// We then try to write the item to the cached location. If that's within the guard page, the official stack gets adjusted

MarkCandidate\* item = (MarkCandidate\*)stack;

stack = stack + sizeof(MarkCandidate);

item->obj = obj;

item->byteCount = byteCount;

#if MARKSTACK\_DIAG

count++;

#endif

#if DBG

if (this->chunkCount > 0)

{

for (Chunk\* chunk = this->preAllocatedChunk; chunk->next != null; chunk = chunk->next)

{

if (chunk->next == null)

{

char\* page = (((char\*)chunk) + ChunkSize - PageSize);

// Don't write to the old page when a new chunk is pushed.

::VirtualProtect((LPVOID)page, PageSize, PAGE\_NOACCESS, NULL);

}

}

}

#endif

return item;

}

MarkStack() throw();

~MarkStack() throw();

void Clear() throw();

bool Empty() throw()

{

return (stack == start && chunkTail == preAllocatedChunk);

}

\_\_forceinline MarkCandidate\* Pop() throw()

{

// TODO: Can we make this faster?

// One option (at the expense of wasting another 4K/Chunk) is to use a guard page at the start of the chunk too

// Then we can eliminate this bound-check

// Currently, the fast case takes 5 instructions:

// 2 to load stack and start

// 1 to check if they're not equal

// 1 to subtract the stack

// 1 to reload it back into the register

if (stack != start)

{

stack = stack - sizeof(MarkCandidate);

#if MARKSTACK\_DIAG

count--;

#endif

return (MarkCandidate\*)stack;

}

return SlowPop();

}

#if MARKSTACK\_DIAG

size\_t Count() { return count; }

#endif

static int HandleException(LPEXCEPTION\_POINTERS pEP, Recycler\* recycler) throw();

#ifndef MARKSTACK\_TEST

private:

#endif

static int HandleExceptionInternal(LPEXCEPTION\_POINTERS pEP, MarkStack<ChunkSize, PageSize, CommitSize>\* markStack) throw();

char\* start;

char\* stack;

char\* end;

Chunk\* preAllocatedChunk;

Chunk\* chunkTail;

// We copy the stack contents here if we pop the chunk

// and have to reset the guard page

MarkCandidate cachedEnd;

#if MARKSTACK\_DIAG || defined(F\_JSETW)

unsigned int chunkCount;

#endif

private:

bool GrowStack() throw();

void FreeNonPreAllocatedChunks() throw();

void CreatePreallocatedChunk() throw();

void ResetPreAllocatedChunk() throw();

\_\_forceinline MarkCandidate\* SlowPop() throw();

// Chunk manipulation

Chunk\* ReserveMemoryForNewChunk() throw();

void FreeChunk(Chunk\* chunk) throw();

char\* CommitAndPushChunk(Chunk\* chunk) throw();

\_\_forceinline Chunk\* PopChunk() throw();

char\* CommitNextPage() throw();

// Memory wrappers

void\* CommitMemory(void\* address, DWORD size, DWORD flags) throw();

BOOL DecommitMemory(void\* address, DWORD size) throw();

void\* ReserveMemory(DWORD size, DWORD flags) throw();

BOOL ReleaseMemory(void\* address, DWORD size) throw();

#if MARKSTACK\_DIAG

//

// Debug structure to keep track of every call to VirtualAlloc with MEM\_RESERVE

//

struct VirtualAllocationEntry

{

void\* address;

DWORD size;

};

typedef SList<VirtualAllocationEntry, HeapAllocator> VAEList;

//

// Find an allocation entry corresponding to the given address

// This is fairly naive- we walk through the list of all entries and find one within

// whose range this address lives

//

VAEList \_allocations;

VirtualAllocationEntry\* CheckAllocationEntryExists(void\* address)

{

VAEList::Iterator iterator(&this->\_allocations);

while (iterator.Next())

{

VirtualAllocationEntry& entry = iterator.Data();

// Make sure we don't already have the new address isn't in range of existing allocations

if (entry.address <= address && address < ((char\*)entry.address) + entry.size)

{

return (&entry);

}

}

return nullptr;

}

size\_t count;

size\_t committedBytes;

size\_t reservedBytes;

#endif

void\* lastGuardPage;

Chunk staticChunk;

};

};

typedef MarkStack::MarkStack<DEFAULT\_CHUNK\_SIZE, DEFAULT\_PAGE\_SIZE, DEFAULT\_COMMIT\_SIZE> RecyclerMarkStack;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#include "MemoryLogger.h"

#ifdef ENABLE\_TRACE

namespace Js

{

MemoryLogger::MemoryLogger(TAllocator\* alloc, ULONG count) : m\_alloc(alloc), m\_capacity(count), m\_current(0)

{

m\_log = AnewArrayZ(m\_alloc, wchar\_t\*, m\_capacity);

}

MemoryLogger::~MemoryLogger()

{

Adelete(m\_alloc, m\_log);

}

void MemoryLogger::Write(const wchar\_t\* msg)

{

#ifdef EXCEPTION\_CHECK

// In most cases this will be called at runtime when we have exception check enabled.

AutoNestedHandledExceptionType autoNestedHandledExceptionType(ExceptionType\_DisableCheck);

#endif

AutoCriticalSection autocs(&m\_criticalSection); // TODO: with circular buffer now we can use much granular lock.

// Create a copy of the message.

size\_t len = wcslen(msg);

wchar\_t\* buf = AnewArray(m\_alloc, wchar\_t, len + 1);

js\_wmemcpy\_s(buf, len + 1, msg, len + 1); // Copy with the NULL-terminator.

// m\_current is the next position to write to.

if (m\_log[m\_current])

{

Adelete(m\_alloc, m\_log[m\_current]);

}

m\_log[m\_current] = buf;

m\_current = (m\_current + 1) % m\_capacity;

}

MemoryLogger\* MemoryLogger::Create(TAllocator\* alloc, ULONG count)

{

#ifdef EXCEPTION\_CHECK

AutoNestedHandledExceptionType autoNestedHandledExceptionType(ExceptionType\_DisableCheck);

#endif

return Anew(alloc, MemoryLogger, alloc, count);

}

#ifdef STACK\_BACK\_TRACE

StackTraceHelper\* StackTraceHelper::Create(TAllocator\* alloc)

{

#ifdef EXCEPTION\_CHECK

AutoNestedHandledExceptionType autoNestedHandledExceptionType(ExceptionType\_DisableCheck);

#endif

return Anew(alloc, StackTraceHelper, alloc);

}

// Capture and fill buffer.

ULONG StackTraceHelper::GetStackTrace(ULONG framesToSkip, ULONG framesToCapture, void\*\* stackFrames)

{

#ifdef EXCEPTION\_CHECK

AutoNestedHandledExceptionType autoNestedHandledExceptionType(ExceptionType\_DisableCheck);

#endif

StackBackTrace\* stackTrace = this->GetStackBackTrace(framesToCapture);

ULONG capturedCount = stackTrace->Capture(framesToSkip);

Assert(capturedCount <= framesToCapture);

ULONG i = 0;

stackTrace->Map([&i, framesToCapture, stackFrames](void\* funcAddr){

Assert(i <= framesToCapture);

stackFrames[i++] = funcAddr;

});

return capturedCount;

}

// Capture and print.

size\_t StackTraceHelper::PrintStackTrace(ULONG framesToSkip, ULONG framesToCapture)

{

#ifdef EXCEPTION\_CHECK

AutoNestedHandledExceptionType autoNestedHandledExceptionType(ExceptionType\_DisableCheck);

#endif

StackBackTrace\* stackTrace = this->GetStackBackTrace(framesToCapture);

stackTrace->Capture(framesToSkip);

return stackTrace->Print();

}

StackBackTrace\* StackTraceHelper::GetStackBackTrace(ULONG frameCount)

{

// Create, if we haven't already for calling thread.

if (!s\_stackBackTrace)

{

s\_stackBackTrace = StackBackTrace::Create(m\_alloc, frameCount);

}

Assert(s\_stackBackTrace->GetRequestedFrameCount() >= frameCount);

return s\_stackBackTrace;

}

\_\_declspec(thread) StackBackTrace\* StackTraceHelper::s\_stackBackTrace = nullptr;

#endif STACK\_BACK\_TRACE

}

#endif ENABLE\_TRACE

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef ENABLE\_TRACE

namespace Js

{

// Logs in memory/inproc into StringBuilder.

// Thread safe.

// Uses Arena but can be templatized for other allocators.

// TODO: Consider unchaining and using circular buffer when reaching limit.

class MemoryLogger : public ILogger

{

typedef ArenaAllocator TAllocator;

private:

ULONG m\_current;

ULONG m\_capacity; // The number of elements in circular buffer.

wchar\_t\*\* m\_log; // Points to a circular buffer of wchar\_t\*.

TAllocator\* m\_alloc;

CriticalSection m\_criticalSection;

public:

static MemoryLogger\* Create(TAllocator\* alloc, ULONG elementCount);

MemoryLogger(TAllocator\* alloc, ULONG elementCount);

~MemoryLogger();

void Write(const wchar\_t\* msg) override;

};

#ifdef STACK\_BACK\_TRACE

// Used by output.cpp to print stack trace.

// Separate class to aboid build errors with jscript9diag.dll which doesn't link with memory.lib.

class StackTraceHelper : public IStackTraceHelper

{

typedef ArenaAllocator TAllocator;

private:

TAllocator\* m\_alloc;

\_\_declspec(thread) static StackBackTrace\* s\_stackBackTrace;

StackBackTrace\* GetStackBackTrace(ULONG frameCount);

public:

static StackTraceHelper\* Create(TAllocator\* alloc);

virtual size\_t PrintStackTrace(ULONG framesToSkip, ULONG framesToCapture) override;

virtual ULONG GetStackTrace(ULONG framesToSkip, ULONG framesToCapture, void\*\* stackFrames) override;

private:

StackTraceHelper(TAllocator\* alloc) : m\_alloc(alloc) {}

void Print();

};

#endif STACK\_BACK\_TRACE

} // namespace Js.

#endif ENABLE\_TRACE

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

// These are empty stubs here but DLL can supply an .OBJ with an implementation

#ifndef ETW\_MEMORY\_TRACKING

void ArenaMemoryTracking::Activate()

{

// Called to activate arena memory tracking

}

// ArenaMemoryTracking stubs

void ArenaMemoryTracking::ArenaCreated(Allocator \*arena, \_\_in LPCWSTR name)

{

// Called when arena is created.

}

void ArenaMemoryTracking::ArenaDestroyed(Allocator \*arena)

{

// Called when arena is destroyed

}

void ArenaMemoryTracking::ReportAllocation(Allocator \*arena, void \*address, size\_t size)

{

// Called when size bytes at address are allocated

}

void ArenaMemoryTracking::ReportReallocation(Allocator \*arena, void \*address, size\_t existingSize, size\_t newSize)

{

// Called when a reallocation where newSize < exsitingSize.

// This will only be called if newSize < existingSize.

// This is to inform a tracking that a realloc is taking place and ReportFree() will be called on address + newSize soon

// and the ReportFree for address will report newSize instead of existing size

}

void ArenaMemoryTracking::ReportFree(Allocator \*arena, void \*address, size\_t size)

{

// Called when the when size bytes at address are freed. address was either reported by ReportAllocation() or as a

// result of a buffer being split reported by ReportReallocation().

// IMPORTANT: ReportFree() will always be called after ReportReallocation() to report the newly free memory of address + newSize.

}

void ArenaMemoryTracking::ReportFreeAll(Allocator \*arena)

{

// Called when all the arena memory currently allocated is bulk freed.

}

void RecyclerMemoryTracking::Activate()

{

// Called to active recycler memory tracking

}

// RecyclerMemoryTracking stubs

bool RecyclerMemoryTracking::IsActive()

{

// Should return when tracking is active. This is used to force ReportFree() calls. Without this ReportFree() is only called on

// finalizable memory which is only part of the memory allocated in the recycler.

return false;

}

void RecyclerMemoryTracking::ReportRecyclerCreate(Recycler \* recycler)

{

// Called when a recycler is created.

}

void RecyclerMemoryTracking::ReportRecyclerDestroy(Recycler \* recycler)

{

// Called when a recycler is freed.

}

void RecyclerMemoryTracking::ReportAllocation(Recycler \* recycler, \_\_in void \*address, size\_t size)

{

// Called when size bytes at address are allocated from the recycler.

}

void RecyclerMemoryTracking::ReportFree(Recycler \* recycler, \_\_in void \*address, size\_t size)

{

// Called when size bytes at address are freed.

}

void RecyclerMemoryTracking::ReportUnallocated(Recycler \* recycler, \_\_in void\* address, \_\_in void \*endAddress, size\_t sizeCat)

{

// Even though the memory is not really allocated between address and endAddress,

// the recycler initially treats it as allocated and a ReportFree() will be called on it even

// though ReportAllocation() is never called. This can be treated as equivalent of the parent

// requesting the following be performed:

//

// while (address + sizeCat <= endAddress)

// {

// ReportFree(address, sizeCat);

// address += sizeCat;

// }

//

// if address where a (char \*)

}

#endif

// PageTracking stubs

void PageTracking::Activate()

{

// Called to activate page allocator tracking

}

void PageTracking::PageAllocatorCreated(PageAllocator \*pageAllocator)

{

// Called when a page allocator is created.

}

void PageTracking::PageAllocatorDestroyed(PageAllocator \*pageAllocator)

{

// Called when a page allocator is destroyed.

}

void PageTracking::ReportAllocation(PageAllocator \*pageAllocator, \_\_in void \*address, size\_t size)

{

// Called when size bytes are allocated at address.

}

void PageTracking::ReportFree(PageAllocator \*pageAllocator, \_\_in void \*address, size\_t size)

{

// Called when size bytes are freed at address.

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// Allows an external memory tracking utility (such as MemSpect) to track recycler allocations

// Routines called to allow tracking memory. These routines are empty for RET

// builds but allow external tools (such as Memspect) to track arena allocations individually

// for CHK and TEST builds. Requires the appropriate configuration flag, e.g. -MemspectEnabled.

namespace Memory

{

class ArenaAllocator;

class ArenaMemoryTracking

{

public:

static void Activate();

static void ArenaCreated(Allocator \*arena, \_\_in LPCWSTR name);

static void ArenaDestroyed(Allocator \*arena);

static void ReportAllocation(Allocator \*arena, void \*address, size\_t size);

static void ReportReallocation(Allocator \*arena, void \*address, size\_t existingSize, size\_t newSize);

static void ReportFree(Allocator \*arena, void \*address, size\_t size);

static void ReportFreeAll(Allocator \*arena);

};

class Recycler;

class RecyclerMemoryTracking

{

public:

static bool IsActive();

static void Activate();

static void ReportRecyclerCreate(Recycler \* recycler);

static void ReportRecyclerDestroy(Recycler \* recycler);

static void ReportAllocation(Recycler \* recycler, \_\_in void \*address, size\_t size);

static void ReportFree(Recycler \* recycler, \_\_in void \*address, size\_t size);

static void ReportUnallocated(Recycler \* recycler, \_\_in void\* address, \_\_in void \*endAddress, size\_t sizeCat);

};

class PageTracking

{

public:

static void Activate();

static void PageAllocatorCreated(PageAllocator \*pageAllocator);

static void PageAllocatorDestroyed(PageAllocator \*pageAllocator);

static void ReportAllocation(PageAllocator \*pageAllocator, \_\_in void \*address, size\_t size);

static void ReportFree(PageAllocator \*pageAllocator, \_\_in void \*address, size\_t size);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#define UpdateMinimum(dst, src) if (dst > src) { dst = src; }

//=============================================================================================================

// Segment

//=============================================================================================================

template<typename T>

SegmentBase<T>::SegmentBase(PageAllocatorBase<T> \* allocator, size\_t pageCount) :

allocator(allocator),

address(nullptr),

trailingGuardPageCount(0),

leadingGuardPageCount(0),

secondaryAllocPageCount(allocator->secondaryAllocPageCount),

secondaryAllocator(nullptr)

#if defined(\_M\_X64\_OR\_ARM64) && defined(RECYCLER\_WRITE\_BARRIER)

, isWriteBarrierAllowed(false)

#endif

{

this->segmentPageCount = pageCount + secondaryAllocPageCount;

}

template<typename T>

SegmentBase<T>::~SegmentBase()

{

Assert(this->allocator != nullptr);

if (this->address)

{

char\* originalAddress = this->address - (leadingGuardPageCount \* AutoSystemInfo::PageSize);

allocator->GetVirtualAllocator()->Free(originalAddress, GetPageCount() \* AutoSystemInfo::PageSize, MEM\_RELEASE);

allocator->ReportFree(this->segmentPageCount \* AutoSystemInfo::PageSize); //Note: We reported the guard pages free when we decommitted them during segment initialization

#if defined(\_M\_X64\_OR\_ARM64) && defined(RECYCLER\_WRITE\_BARRIER\_BYTE)

RecyclerWriteBarrierManager::OnSegmentFree(this->address, this->segmentPageCount);

#endif

}

if(this->secondaryAllocator)

{

this->secondaryAllocator->Delete();

this->secondaryAllocator = nullptr;

}

}

template<typename T>

bool SegmentBase<T>::IsInPreReservedHeapPageAllocator() const

{

return this->allocator->GetVirtualAllocator() != nullptr;

}

template<typename T>

bool

SegmentBase<T>::Initialize(DWORD allocFlags, bool excludeGuardPages)

{

Assert(this->address == nullptr);

char\* originalAddress = nullptr;

bool addGuardPages = false;

if (!excludeGuardPages)

{

addGuardPages = (this->segmentPageCount \* AutoSystemInfo::PageSize) > VirtualAllocThreshold;

#if \_M\_IX86\_OR\_ARM32

unsigned int randomNumber = static\_cast<unsigned int>(Math::Rand());

addGuardPages = addGuardPages && (randomNumber % 4 == 1);

#endif

#if DEBUG

addGuardPages = addGuardPages || Js::Configuration::Global.flags.ForceGuardPages;

#endif

if (addGuardPages)

{

unsigned int randomNumber = static\_cast<unsigned int>(Math::Rand());

this->leadingGuardPageCount = randomNumber % maxGuardPages + minGuardPages;

this->trailingGuardPageCount = minGuardPages;

}

}

// We can only allocate with this granularity using VirtualAlloc

size\_t totalPages = Math::Align<size\_t>(this->segmentPageCount + leadingGuardPageCount + trailingGuardPageCount, AutoSystemInfo::Data.GetAllocationGranularityPageCount());

this->segmentPageCount = totalPages - (leadingGuardPageCount + trailingGuardPageCount);

#ifdef FAULT\_INJECTION

if(Js::FaultInjection::Global.ShouldInjectFault(Js::FaultInjection::Global.NoThrow))

{

this->address = nullptr;

return(address != nullptr);

}

#endif

if (!this->allocator->RequestAlloc(totalPages \* AutoSystemInfo::PageSize))

{

return nullptr;

}

this->address = (char \*) GetAllocator()->GetVirtualAllocator()->Alloc(NULL, totalPages \* AutoSystemInfo::PageSize, MEM\_RESERVE | allocFlags, PAGE\_READWRITE, this->IsInCustomHeapAllocator());

originalAddress = this->address;

if (originalAddress != nullptr)

{

if (addGuardPages)

{

#if DBG\_DUMP

GUARD\_PAGE\_TRACE(L"Number of Leading Guard Pages: %d\n", leadingGuardPageCount);

GUARD\_PAGE\_TRACE(L"Starting address of Leading Guard Pages: 0x%p\n", address);

GUARD\_PAGE\_TRACE(L"Offset of Segment Start address: 0x%p\n", this->address + (leadingGuardPageCount\*AutoSystemInfo::PageSize));

GUARD\_PAGE\_TRACE(L"Starting address of Trailing Guard Pages: 0x%p\n", address + ((leadingGuardPageCount + this->segmentPageCount)\*AutoSystemInfo::PageSize));

#endif

#pragma warning(suppress: 6250)

GetAllocator()->GetVirtualAllocator()->Free(address, leadingGuardPageCount\*AutoSystemInfo::PageSize, MEM\_DECOMMIT);

#pragma warning(suppress: 6250)

GetAllocator()->GetVirtualAllocator()->Free(address + ((leadingGuardPageCount + this->segmentPageCount)\*AutoSystemInfo::PageSize), trailingGuardPageCount\*AutoSystemInfo::PageSize, MEM\_DECOMMIT);

this->allocator->ReportFree((leadingGuardPageCount + trailingGuardPageCount)\*AutoSystemInfo::PageSize);

this->address = this->address + (leadingGuardPageCount\*AutoSystemInfo::PageSize);

}

if (!allocator->CreateSecondaryAllocator(this, &this->secondaryAllocator))

{

GetAllocator()->GetVirtualAllocator()->Free(originalAddress, GetPageCount() \* AutoSystemInfo::PageSize, MEM\_RELEASE);

this->allocator->ReportFree(totalPages \* AutoSystemInfo::PageSize);

this->address = nullptr;

}

#if defined(\_M\_X64\_OR\_ARM64) && defined(RECYCLER\_WRITE\_BARRIER\_BYTE)

else if (!RecyclerWriteBarrierManager::OnSegmentAlloc(this->address, this->segmentPageCount))

{

GetAllocator()->GetVirtualAllocator()->Free(originalAddress, GetPageCount() \* AutoSystemInfo::PageSize, MEM\_RELEASE);

this->allocator->ReportFree(totalPages \* AutoSystemInfo::PageSize);

this->address = nullptr;

}

else

{

this->isWriteBarrierAllowed = true;

}

#endif

}

if (this->address == nullptr)

{

this->allocator->ReportFailure(totalPages \* AutoSystemInfo::PageSize);

}

return (this->address != nullptr);

}

//=============================================================================================================

// PageSegment

//=============================================================================================================

template<typename T>

PageSegmentBase<T>::PageSegmentBase(PageAllocatorBase<T> \* allocator, bool external) :

SegmentBase(allocator, allocator->maxAllocPageCount), decommitPageCount(0)

{

Assert(this->segmentPageCount == allocator->maxAllocPageCount + allocator->secondaryAllocPageCount);

if (external)

{

this->freePageCount = 0;

this->ClearAllInFreePagesBitVector();

}

else

{

this->freePageCount = this->GetAvailablePageCount();

uint maxPageCount = GetMaxPageCount();

this->SetRangeInFreePagesBitVector(0, this->freePageCount);

if (this->freePageCount != maxPageCount)

{

this->ClearRangeInFreePagesBitVector(this->freePageCount, (maxPageCount - this->freePageCount));

}

Assert(this->GetCountOfFreePages() == this->freePageCount);

}

}

#ifdef PAGEALLOCATOR\_PROTECT\_FREEPAGE

template<typename T>

bool

PageSegmentBase<T>::Initialize(DWORD allocFlags, bool excludeGuardPages)

{

Assert(freePageCount + allocator->secondaryAllocPageCount == this->segmentPageCount || freePageCount == 0);

if (\_\_super::Initialize(allocFlags, excludeGuardPages))

{

if (freePageCount != 0)

{

DWORD oldProtect;

BOOL vpresult = ::VirtualProtect(this->address, this->GetAvailablePageCount() \* AutoSystemInfo::PageSize, PAGE\_NOACCESS, &oldProtect);

Assert(vpresult && oldProtect == PAGE\_READWRITE);

}

return true;

}

return false;

}

#endif

template<typename T>

void

PageSegmentBase<T>::Prime()

{

#ifndef PAGEALLOCATOR\_PROTECT\_FREEPAGE

for (uint i = 0; i < this->GetAvailablePageCount(); i++)

{

this->address[i \* AutoSystemInfo::PageSize] = NULL;

}

#endif

}

template<typename T>

bool

PageSegmentBase<T>::IsAllocationPageAligned(\_\_in char\* address, size\_t pageCount, PageHeapMode pageHeapFlags)

{

#ifdef RECYCLER\_PAGE\_HEAP

if (pageHeapFlags != PageHeapMode::PageHeapModeOff)

{

// In PageHeap mode, we should ensure that if the guard page

// is in the front, the page after the guard page is aligned

if ((pageHeapFlags == PageHeapMode::PageHeapModeBlockStart))

{

address += AutoSystemInfo::PageSize;

}

// We don't care about whether the guard pages themselves are aligned

// or fit in the chunk, so don't count the guard page for the purposes

// of alignment

pageCount--;

}

#endif

// Require that allocations are aligned at a boundary

// corresponding to the page count

// REVIEW: This might actually lead to additional address space fragmentation

// because of the leading guard pages feature in the page allocator

// We can restrict the guard pages to be an even number to improve the chances

// of having the first allocation be aligned but that reduces the effectiveness

// of having a random number of guard pages

uintptr\_t mask = (pageCount \* AutoSystemInfo::PageSize) - 1;

if ((reinterpret\_cast<uintptr\_t>(address)& mask) == 0)

{

return true;

}

return false;

}

template<typename T>

template <bool notPageAligned>

char \*

PageSegmentBase<T>::AllocPages(uint pageCount, PageHeapMode pageHeapFlags)

{

Assert(freePageCount != 0);

Assert(freePageCount == (uint)this->GetCountOfFreePages());

if (freePageCount < pageCount)

{

return nullptr;

}

Assert(!IsFull());

#pragma prefast(push)

#pragma prefast(suppress:\_\_WARNING\_LOOP\_INDEX\_UNDERFLOW, "Prefast about overflow when multiplying index.")

uint index = this->GetNextBitInFreePagesBitVector(0);

while (index != -1)

{

Assert(index < allocator->GetMaxAllocPageCount());

if (GetAvailablePageCount() - index < pageCount)

{

break;

}

if (pageCount == 1 || this->TestRangeInFreePagesBitVector(index, pageCount))

{

char \* allocAddress = this->address + index \* AutoSystemInfo::PageSize;

if (pageCount > 1 && !notPageAligned)

{

if (!IsAllocationPageAligned(allocAddress, pageCount, pageHeapFlags))

{

index = this->freePages.GetNextBit(index + 1);

continue;

}

}

this->ClearRangeInFreePagesBitVector(index, pageCount);

freePageCount -= pageCount;

Assert(freePageCount == (uint)this->GetCountOfFreePages());

#ifdef PAGEALLOCATOR\_PROTECT\_FREEPAGE

DWORD oldProtect;

BOOL vpresult = ::VirtualProtect(allocAddress, pageCount \* AutoSystemInfo::PageSize, PAGE\_READWRITE, &oldProtect);

Assert(vpresult && oldProtect == PAGE\_NOACCESS);

#endif

return allocAddress;

}

index = this->GetNextBitInFreePagesBitVector(index + 1);

}

#pragma prefast(pop)

return nullptr;

}

#pragma prefast(push)

#pragma prefast(suppress:\_\_WARNING\_LOOP\_INDEX\_UNDERFLOW, "Prefast about overflow when multiplying index.")

template<typename TVirtualAlloc>

template<typename T, bool notPageAligned>

char \*

PageSegmentBase<TVirtualAlloc>::AllocDecommitPages(uint pageCount, T freePages, T decommitPages, PageHeapMode pageHeapFlags)

{

Assert(freePageCount == (uint)this->GetCountOfFreePages());

Assert(decommitPageCount == (uint)this->GetCountOfDecommitPages());

Assert(decommitPageCount != 0);

if (freePageCount + decommitPageCount < pageCount)

{

return nullptr;

}

Assert(secondaryAllocator == nullptr || secondaryAllocator->CanAllocate());

T freeAndDecommitPages = freePages;

freeAndDecommitPages.Or(&decommitPages);

uint oldFreePageCount = freePageCount;

uint index = freeAndDecommitPages.GetNextBit(0);

while (index != -1)

{

Assert(index < allocator->GetMaxAllocPageCount());

if (GetAvailablePageCount() - index < pageCount)

{

break;

}

if (pageCount == 1 || freeAndDecommitPages.TestRange(index, pageCount))

{

char \* pages = this->address + index \* AutoSystemInfo::PageSize;

if (!notPageAligned)

{

if (!IsAllocationPageAligned(pages, pageCount, pageHeapFlags))

{

index = freeAndDecommitPages.GetNextBit(index + 1);

continue;

}

}

void \* ret = GetAllocator()->GetVirtualAllocator()->Alloc(pages, pageCount \* AutoSystemInfo::PageSize, MEM\_COMMIT, PAGE\_READWRITE, this->IsInCustomHeapAllocator());

if (ret != nullptr)

{

Assert(ret == pages);

this->ClearRangeInFreePagesBitVector(index, pageCount);

this->ClearRangeInDecommitPagesBitVector(index, pageCount);

uint newFreePageCount = this->GetCountOfFreePages();

freePageCount = freePageCount - oldFreePageCount + newFreePageCount;

decommitPageCount -= pageCount - (oldFreePageCount - newFreePageCount);

Assert(freePageCount == (uint)this->GetCountOfFreePages());

Assert(decommitPageCount == (uint)this->GetCountOfDecommitPages());

return pages;

}

else if (pageCount == 1)

{

// if we failed to commit one page, we should just give up.

return nullptr;

}

}

index = freeAndDecommitPages.GetNextBit(index + 1);

}

return nullptr;

}

#pragma prefast(pop)

template<typename T>

void

PageSegmentBase<T>::ReleasePages(\_\_in void \* address, uint pageCount)

{

Assert(address >= this->address);

Assert(pageCount <= allocator->maxAllocPageCount);

Assert(((uint)(((char \*)address) - this->address)) <= (allocator->maxAllocPageCount - pageCount) \* AutoSystemInfo::PageSize);

Assert(!IsFreeOrDecommitted(address, pageCount));

uint base = this->GetBitRangeBase(address);

this->SetRangeInFreePagesBitVector(base, pageCount);

this->freePageCount += pageCount;

Assert(freePageCount == (uint)this->GetCountOfFreePages());

#ifdef PAGEALLOCATOR\_PROTECT\_FREEPAGE

DWORD oldProtect;

BOOL vpresult = ::VirtualProtect(address, pageCount \* AutoSystemInfo::PageSize, PAGE\_NOACCESS, &oldProtect);

Assert(vpresult && oldProtect == PAGE\_READWRITE);

#endif

}

template<typename T>

void

PageSegmentBase<T>::ChangeSegmentProtection(DWORD protectFlags, DWORD expectedOldProtectFlags)

{

// TODO: There is a discrepancy in PageSegmentBase

// The segment page count is initialized in PageSegmentBase::Initialize. It takes into account

// the guard pages + any additional pages for alignment.

// However, the free page count is calculated for the segment before initialize is called.

// In practice, what happens is the following. The initial segment page count is 256. This

// ends up being the free page count too. When initialize is called, we allocate the guard

// pages and the alignment pages, which causes the total page count to be 272. The segment

// page count is then calculated as total - guard, which means 256 <= segmentPageCount < totalPageCount

// The code in PageSegment's constructor will mark the pages between 256 and 272 as in use,

// which is why it generally works. However, it breaks in the case where we want to know the end

// address of the page. It should really be address + 256 \* 4k but this->GetEndAddress will return

// a value greater than that. Need to do a pass through the counts and make sure that it's rational.

// For now, simply calculate the end address from the allocator's page count

char\* segmentEndAddress = this->address + (this->allocator->GetMaxAllocPageCount() \* AutoSystemInfo::PageSize);

for (char\* address = this->address; address < segmentEndAddress; address += AutoSystemInfo::PageSize)

{

if (!IsFreeOrDecommitted(address))

{

char\* endAddress = address;

do

{

endAddress += AutoSystemInfo::PageSize;

} while (endAddress < segmentEndAddress && !IsFreeOrDecommitted(endAddress));

Assert(((uintptr\_t)(endAddress - address)) < UINT\_MAX);

DWORD regionSize = (DWORD) (endAddress - address);

DWORD oldProtect = 0;

#if DBG

MEMORY\_BASIC\_INFORMATION info = { 0 };

::VirtualQuery(address, &info, sizeof(MEMORY\_BASIC\_INFORMATION));

Assert(info.Protect == expectedOldProtectFlags);

#endif

BOOL fSuccess = ::VirtualProtect(address, regionSize, protectFlags, &oldProtect);

Assert(fSuccess == TRUE);

Assert(oldProtect == expectedOldProtectFlags);

address = endAddress;

}

}

}

template<typename T>

template <bool onlyUpdateState>

void

PageSegmentBase<T>::DecommitPages(\_\_in void \* address, uint pageCount)

{

Assert(address >= this->address);

Assert(pageCount <= allocator->maxAllocPageCount);

Assert(((uint)(((char \*)address) - this->address)) <= (allocator->maxAllocPageCount - pageCount) \* AutoSystemInfo::PageSize);

Assert(!IsFreeOrDecommitted(address, pageCount));

uint base = this->GetBitRangeBase(address);

this->SetRangeInDecommitPagesBitVector(base, pageCount);

this->decommitPageCount += pageCount;

if (!onlyUpdateState)

{

#pragma warning(suppress: 6250)

GetAllocator()->GetVirtualAllocator()->Free(address, pageCount \* AutoSystemInfo::PageSize, MEM\_DECOMMIT);

}

Assert(decommitPageCount == (uint)this->GetCountOfDecommitPages());

}

template<typename T>

size\_t

PageSegmentBase<T>::DecommitFreePages(size\_t pageToDecommit)

{

Assert(pageToDecommit != 0);

char \* currentAddress = this->address;

uint decommitCount = 0;

for (uint i = 0; i < this->GetAvailablePageCount(); i++)

{

if (this->TestInFreePagesBitVector(i))

{

this->ClearBitInFreePagesBitVector(i);

this->SetBitInDecommitPagesBitVector(i);

#pragma warning(suppress: 6250)

GetAllocator()->GetVirtualAllocator()->Free(currentAddress, AutoSystemInfo::PageSize, MEM\_DECOMMIT);

decommitCount++;

}

currentAddress += AutoSystemInfo::PageSize;

if (decommitCount == pageToDecommit)

{

break;

}

}

Assert(decommitCount <= this->freePageCount);

this->decommitPageCount += decommitCount;

this->freePageCount -= decommitCount;

return decommitCount;

}

//=============================================================================================================

// PageAllocator

//=============================================================================================================

#if DBG

#define ASSERT\_THREAD() AssertMsg(ValidThreadAccess(), "Page allocation should only be used by a single thread");

#else

#define ASSERT\_THREAD()

#endif

/\*

\* Global counter to keep track of the total used bytes by the page allocator

\* per process for performance tooling. This is reported through the

\* JSCRIPT\_PAGE\_ALLOCATOR\_USED\_SIZE ETW event.

\*/

static size\_t totalUsedBytes = 0;

static size\_t maxUsedBytes = 0;

template<typename T>

size\_t PageAllocatorBase<T>::GetAndResetMaxUsedBytes()

{

size\_t value = maxUsedBytes;

maxUsedBytes = 0;

return value;

}

template<typename T>

size\_t

PageAllocatorBase<T>::GetProcessUsedBytes()

{

return totalUsedBytes;

}

template<typename T>

PageAllocatorBase<T>::BackgroundPageQueue::BackgroundPageQueue()

{

::InitializeSListHead(&freePageList);

DebugOnly(this->isZeroPageQueue = false);

}

template<typename T>

PageAllocatorBase<T>::ZeroPageQueue::ZeroPageQueue()

{

::InitializeSListHead(&pendingZeroPageList);

DebugOnly(this->isZeroPageQueue = true);

}

template<typename T>

uint

PageAllocatorBase<T>::GetMaxAllocPageCount()

{

return maxAllocPageCount;

}

template<typename T>

PageAllocatorBase<T>::PageAllocatorBase(AllocationPolicyManager \* policyManager,

#ifndef JD\_PRIVATE

Js::ConfigFlagsTable& flagTable,

#endif

PageAllocatorType type,

uint maxFreePageCount, bool zeroPages, BackgroundPageQueue \* backgroundPageQueue, uint maxAllocPageCount, uint secondaryAllocPageCount,

bool stopAllocationOnOutOfMemory, bool excludeGuardPages) :

policyManager(policyManager),

#ifndef JD\_PRIVATE

pageAllocatorFlagTable(flagTable),

#endif

maxFreePageCount(maxFreePageCount),

freePageCount(0),

allocFlags(0),

zeroPages(zeroPages),

queueZeroPages(false),

hasZeroQueuedPages(false),

backgroundPageQueue(backgroundPageQueue),

minFreePageCount(0),

isUsed(false),

idleDecommitEnterCount(1),

isClosed(false),

stopAllocationOnOutOfMemory(stopAllocationOnOutOfMemory),

disableAllocationOutOfMemory(false),

secondaryAllocPageCount(secondaryAllocPageCount),

excludeGuardPages(excludeGuardPages),

virtualAllocator(nullptr),

type(type)

, reservedBytes(0)

, committedBytes(0)

, usedBytes(0)

, numberOfSegments(0)

{

AssertMsg(Math::IsPow2(maxAllocPageCount + secondaryAllocPageCount), "Illegal maxAllocPageCount: Why is this not a power of 2 aligned?");

this->maxAllocPageCount = maxAllocPageCount;

#if DBG

// By default, a page allocator is not associated with any thread context

// Any host which wishes to associate it with a thread context must do so explicitly

this->threadContextHandle = NULL;

this->concurrentThreadId = (DWORD)-1;

#endif

#if DBG

this->disableThreadAccessCheck = false;

this->debugMinFreePageCount = 0;

#endif

#if DBG\_DUMP

this->decommitPageCount = 0;

this->debugName = nullptr;

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

this->verifyEnabled = false;

this->disablePageReuse = false;

#endif

#ifdef PROFILE\_MEM

this->memoryData = MemoryProfiler::GetPageMemoryData(type);

#endif

PageTracking::PageAllocatorCreated((PageAllocator\*)this);

}

template<typename T>

PageAllocatorBase<T>::~PageAllocatorBase()

{

AssertMsg(this->ValidThreadAccess(), "Page allocator tear-down should only happen on the owning thread");

#if DBG

Assert(!HasMultiThreadAccess());

#endif

SubUsedBytes(usedBytes);

SubCommittedBytes(committedBytes);

SubReservedBytes(reservedBytes);

ReleaseSegmentList(&segments);

ReleaseSegmentList(&fullSegments);

ReleaseSegmentList(&emptySegments);

ReleaseSegmentList(&decommitSegments);

ReleaseSegmentList(&largeSegments);

PageTracking::PageAllocatorDestroyed((PageAllocator\*)this);

}

template<typename T>

void

PageAllocatorBase<T>::StartQueueZeroPage()

{

Assert(HasZeroPageQueue());

Assert(!queueZeroPages);

queueZeroPages = true;

}

template<typename T>

void

PageAllocatorBase<T>::StopQueueZeroPage()

{

Assert(HasZeroPageQueue());

Assert(queueZeroPages);

queueZeroPages = false;

}

template<typename T>

bool

PageAllocatorBase<T>::HasZeroPageQueue() const

{

bool hasZeroPageQueue = (ZeroPages() && this->backgroundPageQueue != nullptr);

Assert(backgroundPageQueue == nullptr || hasZeroPageQueue == backgroundPageQueue->isZeroPageQueue);

return hasZeroPageQueue;

}

#if DBG

template<typename T>

bool

PageAllocatorBase<T>::HasZeroQueuedPages() const

{

Assert(!HasZeroPageQueue() || hasZeroQueuedPages ||

::QueryDepthSList(&(((ZeroPageQueue \*)this->backgroundPageQueue)->pendingZeroPageList)) == 0);

return hasZeroQueuedPages;

}

#endif

template<typename T>

PageAllocation \*

PageAllocatorBase<T>::AllocPagesForBytes(size\_t requestBytes)

{

Assert(!isClosed);

ASSERT\_THREAD();

uint pageSize = AutoSystemInfo::PageSize;

uint addSize = sizeof(PageAllocation) + pageSize - 1; // this shouldn't overflow

// overflow check

size\_t allocSize = AllocSizeMath::Add(requestBytes, addSize);

if (allocSize == (size\_t)-1)

{

return nullptr;

}

size\_t pages = allocSize / pageSize;

return this->AllocAllocation(pages);

}

template<typename T>

PageSegmentBase<T> \*

PageAllocatorBase<T>::AllocPageSegment(DListBase<PageSegmentBase<T>>& segmentList, PageAllocatorBase<T> \* pageAllocator, bool external)

{

PageSegmentBase<T> \* segment = segmentList.PrependNode(&NoThrowNoMemProtectHeapAllocator::Instance, pageAllocator, external);

if (segment == nullptr)

{

return nullptr;

}

if (!segment->Initialize((external ? 0 : MEM\_COMMIT) | pageAllocator->allocFlags, pageAllocator->excludeGuardPages))

{

segmentList.RemoveHead(&NoThrowNoMemProtectHeapAllocator::Instance);

return nullptr;

}

return segment;

}

template<typename T>

PageSegmentBase<T> \*

PageAllocatorBase<T>::AddPageSegment(DListBase<PageSegmentBase<T>>& segmentList)

{

Assert(!HasMultiThreadAccess());

PageSegmentBase<T> \* segment = AllocPageSegment(segmentList, this, false);

if (segment != nullptr)

{

LogAllocSegment(segment);

this->AddFreePageCount(maxAllocPageCount);

}

return segment;

}

template<>

char \*

HeapPageAllocator<PreReservedVirtualAllocWrapper>::InitPageSegment()

{

Assert(virtualAllocator);

PageSegmentBase<PreReservedVirtualAllocWrapper> \* firstPreReservedSegment = AddPageSegment(emptySegments);

if (firstPreReservedSegment == nullptr)

{

return nullptr;

}

return firstPreReservedSegment->GetAddress();

}

template<>

char \*

HeapPageAllocator<VirtualAllocWrapper>::InitPageSegment()

{

Assert(false);

return nullptr;

}

template<typename T>

PageSegmentBase<T> \*

HeapPageAllocator<T>::AddPageSegment(DListBase<PageSegmentBase<T>>& segmentList)

{

Assert(!HasMultiThreadAccess());

PageSegmentBase<T> \* segment = AllocPageSegment(segmentList, this, false);

if (segment != nullptr)

{

LogAllocSegment(segment);

this->AddFreePageCount(maxAllocPageCount);

}

return segment;

}

template<typename T>

template <bool notPageAligned>

char \*

PageAllocatorBase<T>::TryAllocFreePages(uint pageCount, PageSegmentBase<T> \*\* pageSegment, PageHeapMode pageHeapFlags)

{

Assert(!HasMultiThreadAccess());

if (this->freePageCount < pageCount)

{

return nullptr;

}

FAULTINJECT\_MEMORY\_NOTHROW(this->debugName, pageCount\*4096);

DListBase<PageSegmentBase<T>>::EditingIterator i(&segments);

while (i.Next())

{

PageSegmentBase<T> \* freeSegment = &i.Data();

char \* pages = freeSegment->AllocPages<notPageAligned>(pageCount, pageHeapFlags);

if (pages != nullptr)

{

LogAllocPages(pageCount);

if (freeSegment->GetFreePageCount() == 0)

{

i.MoveCurrentTo(&fullSegments);

}

this->freePageCount -= pageCount;

\*pageSegment = freeSegment;

#if DBG

UpdateMinimum(this->debugMinFreePageCount, this->freePageCount);

#endif

this->FillAllocPages(pages, pageCount);

return pages;

}

}

if (pageCount == 1 && backgroundPageQueue != nullptr)

{

FreePageEntry \* freePage = (FreePageEntry \*)::InterlockedPopEntrySList(&backgroundPageQueue->freePageList);

if (freePage != nullptr)

{

#if DBG

UpdateMinimum(this->debugMinFreePageCount, this->freePageCount);

#endif

\*pageSegment = freePage->segment;

char \* pages;

if (freePage->pageCount != 1)

{

uint pageIndex = --freePage->pageCount;

::InterlockedPushEntrySList(&backgroundPageQueue->freePageList, freePage);

pages = (char \*)freePage + pageIndex \* AutoSystemInfo::PageSize;

}

else

{

pages = (char \*)freePage;

memset(pages, 0, sizeof(FreePageEntry));

}

this->FillAllocPages(pages, pageCount);

return (char \*)pages;

}

}

return nullptr;

}

template<typename T>

void

PageAllocatorBase<T>::FillAllocPages(\_\_in void \* address, uint pageCount)

{

#if DBG

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

for (size\_t i = 0; i < AutoSystemInfo::PageSize \* pageCount; i++)

{

// new pages are filled with zeros, old pages are filled with DbgMemFill

Assert(((byte \*)address)[i] == 0 || ((byte \*)address)[i] == DbgMemFill);

}

#endif

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

if (verifyEnabled)

{

memset(address, Recycler::VerifyMemFill, AutoSystemInfo::PageSize \* pageCount);

return;

}

#endif

#if DBG

if (ZeroPages())

{

// for release build, the page is zeroed in ReleasePages

memset(address, 0, AutoSystemInfo::PageSize \* pageCount);

}

#endif

}

template<typename T>

void

PageAllocatorBase<T>::FillFreePages(\_\_in void \* address, uint pageCount)

{

#if DBG

memset(address, DbgMemFill, AutoSystemInfo::PageSize \* pageCount);

#else

#ifdef RECYCLER\_MEMORY\_VERIFY

if (verifyEnabled)

{

return;

}

#endif

if (ZeroPages())

{

memset(address, 0, AutoSystemInfo::PageSize \* pageCount);

}

#endif

}

template<typename T>

template <bool notPageAligned>

char \*

PageAllocatorBase<T>::TryAllocDecommitedPages(uint pageCount, PageSegmentBase<T> \*\* pageSegment, PageHeapMode pageHeapFlags)

{

Assert(!HasMultiThreadAccess());

DListBase<PageSegmentBase<T>>::EditingIterator i(&decommitSegments);

while (i.Next())

{

PageSegmentBase<T> \* freeSegment = &i.Data();

uint oldFreePageCount = freeSegment->GetFreePageCount();

uint oldDecommitPageCount = freeSegment->GetDecommitPageCount();

char \* pages = freeSegment->DoAllocDecommitPages<notPageAligned>(pageCount, pageHeapFlags);

if (pages != nullptr)

{

this->freePageCount = this->freePageCount - oldFreePageCount + freeSegment->GetFreePageCount();

#if DBG\_DUMP

this->decommitPageCount = this->decommitPageCount - oldDecommitPageCount + freeSegment->GetDecommitPageCount();

#endif

#if DBG

UpdateMinimum(this->debugMinFreePageCount, this->freePageCount);

#endif

uint recommitPageCount = pageCount - (oldFreePageCount - freeSegment->GetFreePageCount());

LogRecommitPages(recommitPageCount);

LogAllocPages(pageCount);

if (freeSegment->GetDecommitPageCount() == 0)

{

auto toList = GetSegmentList(freeSegment);

i.MoveCurrentTo(toList);

}

\*pageSegment = freeSegment;

return pages;

}

}

return nullptr;

}

template<typename T>

PageAllocation \*

PageAllocatorBase<T>::AllocAllocation(size\_t pageCount)

{

PageAllocation \* pageAllocation;

SegmentBase<T> \* segment;

if (pageCount > this->maxAllocPageCount)

{

// We need some space reserved for secondary allocations

segment = AllocSegment(pageCount);

if (segment == nullptr)

{

return nullptr;

}

pageAllocation = (PageAllocation \*)segment->GetAddress();

pageAllocation->pageCount = segment->GetAvailablePageCount();

}

else

{

Assert(pageCount <= UINT\_MAX);

pageAllocation = (PageAllocation \*)AllocPages((uint)pageCount, (PageSegmentBase<T> \*\*)&segment);

if (pageAllocation == nullptr)

{

return nullptr;

}

pageAllocation->pageCount = pageCount;

}

pageAllocation->segment = segment;

return pageAllocation;

}

template<typename T>

SegmentBase<T> \*

PageAllocatorBase<T>::AllocSegment(size\_t pageCount)

{

Assert(!isClosed);

ASSERT\_THREAD();

// Even though we don't idle decommit large segments, we still need to consider these allocations

// as using the page allocator

this->isUsed = true;

SegmentBase<T> \* segment = largeSegments.PrependNode(&NoThrowNoMemProtectHeapAllocator::Instance, this, pageCount);

if (segment == nullptr)

{

return nullptr;

}

if (!segment->Initialize(MEM\_COMMIT | allocFlags, excludeGuardPages))

{

largeSegments.RemoveHead(&NoThrowNoMemProtectHeapAllocator::Instance);

return nullptr;

}

LogAllocSegment(segment);

LogAllocPages(segment->GetPageCount());

PageTracking::ReportAllocation((PageAllocator\*)this, segment->GetAddress(), AutoSystemInfo::PageSize \* segment->GetPageCount());

#ifdef RECYCLER\_MEMORY\_VERIFY

if (verifyEnabled)

{

memset(segment->GetAddress(), Recycler::VerifyMemFill, AutoSystemInfo::PageSize \* segment->GetPageCount());

}

#endif

return segment;

}

char \*

PageAllocatorBase<VirtualAllocWrapper>::Alloc(size\_t \* pageCount, SegmentBase<VirtualAllocWrapper> \*\* segment)

{

Assert(virtualAllocator == nullptr);

return AllocInternal<false>(pageCount, segment);

}

char \*

PageAllocatorBase<PreReservedVirtualAllocWrapper>::Alloc(size\_t \* pageCount, SegmentBase<PreReservedVirtualAllocWrapper> \*\* segment)

{

Assert(virtualAllocator);

if (virtualAllocator->IsPreReservedRegionPresent())

{

return AllocInternal<false>(pageCount, segment);

}

else

{

return nullptr;

}

}

template<typename T>

template <bool doPageAlign>

char \*

PageAllocatorBase<T>::AllocInternal(size\_t \* pageCount, SegmentBase<T> \*\* segment)

{

char \* addr = nullptr;

if (\*pageCount > this->maxAllocPageCount)

{

// Don't bother trying to do single chunk allocation here

// We're allocating a new segment. If the segment size is

// within a single chunk, great, otherwise, doesn't matter

// We need some space reserved for secondary allocations

SegmentBase<T> \* newSegment = this->AllocSegment(\*pageCount);

if (newSegment != nullptr)

{

addr = newSegment->GetAddress();

\*pageCount = newSegment->GetAvailablePageCount();

\*segment = newSegment;

}

}

else

{

Assert(\*pageCount <= UINT\_MAX);

PageSegmentBase<T> \* pageSegment;

if (doPageAlign)

{

// TODO: Remove this entire codepath since doPageAlign is not being used anymore

addr = this->AllocPagesPageAligned((uint)\*pageCount, &pageSegment, PageHeapMode::PageHeapModeOff);

}

else

{

addr = this->AllocPages((uint) \*pageCount, &pageSegment);

}

if (addr != nullptr)

{

\*segment = pageSegment;

}

}

return addr;

}

template<typename T>

void PageAllocatorBase<T>::UpdateMinFreePageCount()

{

UpdateMinimum(minFreePageCount, freePageCount);

Assert(debugMinFreePageCount == minFreePageCount);

}

template<typename T>

void PageAllocatorBase<T>::ResetMinFreePageCount()

{

minFreePageCount = freePageCount;

#if DBG

debugMinFreePageCount = freePageCount;

#endif

}

template<typename T>

void PageAllocatorBase<T>::ClearMinFreePageCount()

{

minFreePageCount = 0;

#if DBG

debugMinFreePageCount = 0;

#endif

}

template<>

char \*

PageAllocatorBase<VirtualAllocWrapper>::AllocPages(uint pageCount, PageSegmentBase<VirtualAllocWrapper> \*\* pageSegment)

{

Assert(virtualAllocator == nullptr);

return AllocPagesInternal<true /\* noPageAligned \*/>(pageCount, pageSegment);

}

template<>

char \*

PageAllocatorBase<PreReservedVirtualAllocWrapper>::AllocPages(uint pageCount, PageSegmentBase<PreReservedVirtualAllocWrapper> \*\* pageSegment)

{

Assert(virtualAllocator);

if (virtualAllocator->IsPreReservedRegionPresent())

{

return AllocPagesInternal<true /\* noPageAligned \*/>(pageCount, pageSegment);

}

else

{

return nullptr;

}

}

template<typename T>

char \*

PageAllocatorBase<T>::AllocPagesPageAligned(uint pageCount, PageSegmentBase<T> \*\* pageSegment, PageHeapMode pageHeapFlags)

{

return AllocPagesInternal<false /\* noPageAligned \*/>(pageCount, pageSegment, pageHeapFlags);

}

template<typename T>

template <bool notPageAligned>

char \*

PageAllocatorBase<T>::AllocPagesInternal(uint pageCount, PageSegmentBase<T> \*\* pageSegment, PageHeapMode pageHeapModeFlags)

{

Assert(!isClosed);

ASSERT\_THREAD();

Assert(pageCount <= this->maxAllocPageCount);

this->isUsed = true;

SuspendIdleDecommit();

char \* allocation = TryAllocFreePages<notPageAligned>(pageCount, pageSegment, pageHeapModeFlags);

if (allocation == nullptr)

{

allocation = SnailAllocPages<notPageAligned>(pageCount, pageSegment, pageHeapModeFlags);

}

ResumeIdleDecommit();

PageTracking::ReportAllocation((PageAllocator\*)this, allocation, AutoSystemInfo::PageSize \* pageCount);

return allocation;

}

template<typename T>

void

PageAllocatorBase<T>::OnAllocFromNewSegment(uint pageCount, \_\_in void\* pages, SegmentBase<T>\* newSegment)

{

DListBase<PageSegmentBase<T>>\* targetSegmentList = (pageCount == maxAllocPageCount) ? &fullSegments : &segments;

LogAllocPages(pageCount);

this->FillAllocPages(pages, pageCount);

this->freePageCount -= pageCount;

#if DBG

UpdateMinimum(this->debugMinFreePageCount, this->freePageCount);

#endif

Assert(targetSegmentList != nullptr);

emptySegments.MoveHeadTo(targetSegmentList);

}

template<typename T>

template <bool notPageAligned>

char \*

PageAllocatorBase<T>::SnailAllocPages(uint pageCount, PageSegmentBase<T> \*\* pageSegment, PageHeapMode pageHeapFlags)

{

Assert(!HasMultiThreadAccess());

char \* pages = nullptr;

PageSegmentBase<T> \* newSegment = nullptr;

if (!emptySegments.Empty())

{

newSegment = &emptySegments.Head();

if (!notPageAligned && !PageSegmentBase<T>::IsAllocationPageAligned(newSegment->GetAddress(), pageCount, pageHeapFlags))

{

newSegment = nullptr;

// Scan through the empty segments for a segment that can fit this allocation

FOREACH\_DLISTBASE\_ENTRY\_EDITING(PageSegmentBase<T>, emptySegment, &this->emptySegments, iter)

{

if (PageSegmentBase<T>::IsAllocationPageAligned(emptySegment.GetAddress(), pageCount, pageHeapFlags))

{

iter.MoveCurrentTo(&this->emptySegments);

newSegment = &emptySegment;

break;

}

}

NEXT\_DLISTBASE\_ENTRY\_EDITING

}

if (newSegment != nullptr)

{

pages = newSegment->AllocPages<notPageAligned>(pageCount, pageHeapFlags);

if (pages != nullptr)

{

OnAllocFromNewSegment(pageCount, pages, newSegment);

\*pageSegment = newSegment;

return pages;

}

}

}

pages = TryAllocDecommitedPages<notPageAligned>(pageCount, pageSegment, pageHeapFlags);

if (pages != nullptr)

{

// TryAllocDecommitedPages may give out a mix of free pages and decommitted pages.

// Free pages are filled with 0xFE in debug build, so we need to zero them

// out before giving it out. In release build, free page is already zeroed

// in ReleasePages

this->FillAllocPages(pages, pageCount);

return pages;

}

Assert(pages == nullptr);

// At this point, we haven't been able to allocate either from the

// decommitted pages, or from the empty segment list, so we'll

// try allocating a segment. In a page allocator with a pre-reserved segment,

// we're not allowed to allocate additional segments so return here.

// Otherwise, add a new segment and allocate from it

newSegment = AddPageSegment(emptySegments);

if (newSegment == nullptr)

{

return nullptr;

}

pages = newSegment->AllocPages<notPageAligned>(pageCount, pageHeapFlags);

if (notPageAligned)

{

// REVIEW: Is this true for single-chunk allocations too? Are new segments guaranteed to

// allow for single-chunk allocations to succeed?

Assert(pages != nullptr);

}

if (pages != nullptr)

{

OnAllocFromNewSegment(pageCount, pages, newSegment);

\*pageSegment = newSegment;

}

return pages;

}

template<typename T>

DListBase<PageSegmentBase<T>> \*

PageAllocatorBase<T>::GetSegmentList(PageSegmentBase<T> \* segment)

{

Assert(!HasMultiThreadAccess());

return

(segment->IsAllDecommitted()) ? nullptr :

(segment->IsFull()) ? &fullSegments :

(segment->ShouldBeInDecommittedList()) ? &decommitSegments :

(segment->IsEmpty()) ? &emptySegments :

&segments;

}

template<typename T>

void

PageAllocatorBase<T>::ReleaseAllocation(PageAllocation \* allocation)

{

SuspendIdleDecommit();

ReleaseAllocationNoSuspend(allocation);

ResumeIdleDecommit();

}

template<typename T>

void

PageAllocatorBase<T>::ReleaseAllocationNoSuspend(PageAllocation \* allocation)

{

this->Release((char \*)allocation, allocation->pageCount, allocation->segment);

}

template<typename T>

void

PageAllocatorBase<T>::Release(void \* address, size\_t pageCount, void \* segmentParam)

{

SegmentBase<T> \* segment = (SegmentBase<T>\*)segmentParam;

Assert(!HasMultiThreadAccess());

Assert(segment->GetAllocator() == this);

if (pageCount > this->maxAllocPageCount)

{

Assert(address == segment->GetAddress());

Assert(pageCount == segment->GetAvailablePageCount());

this->ReleaseSegment(segment);

}

else

{

Assert(pageCount <= UINT\_MAX);

this->ReleasePages(address, static\_cast<uint>(pageCount), (PageSegmentBase<T> \*)segment);

}

}

template<typename T>

void

PageAllocatorBase<T>::ReleaseSegment(SegmentBase<T> \* segment)

{

ASSERT\_THREAD();

#if defined(RECYCLER\_MEMORY\_VERIFY) || defined(ARENA\_MEMORY\_VERIFY)

if (disablePageReuse)

{

DWORD oldProtect;

BOOL vpresult = ::VirtualProtect(segment->GetAddress(), segment->GetPageCount() \* AutoSystemInfo::PageSize, PAGE\_NOACCESS, &oldProtect);

Assert(vpresult && oldProtect == PAGE\_READWRITE);

return;

}

#endif

PageTracking::ReportFree((PageAllocator\*)this, segment->GetAddress(), AutoSystemInfo::PageSize \* segment->GetPageCount());

LogFreePages(segment->GetPageCount());

LogFreeSegment(segment);

largeSegments.RemoveElement(&NoThrowNoMemProtectHeapAllocator::Instance, segment);

}

template<typename T>

void

PageAllocatorBase<T>::AddFreePageCount(uint pageCount)

{

// minFreePageCount is only updated on release of a page or before decommit

// so that we don't have to update it on every page allocation.

UpdateMinFreePageCount();

this->freePageCount += pageCount;

}

template<typename T>

void

PageAllocatorBase<T>::ReleasePages(\_\_in void \* address, uint pageCount, \_\_in void \* segmentParam)

{

Assert(pageCount <= this->maxAllocPageCount);

PageSegmentBase<T> \* segment = (PageSegmentBase<T>\*) segmentParam;

ASSERT\_THREAD();

Assert(!HasMultiThreadAccess());

#if defined(RECYCLER\_MEMORY\_VERIFY) || defined(ARENA\_MEMORY\_VERIFY)

if (disablePageReuse)

{

DWORD oldProtect;

BOOL vpresult = ::VirtualProtect(address, pageCount \* AutoSystemInfo::PageSize, PAGE\_NOACCESS, &oldProtect);

Assert(vpresult && oldProtect == PAGE\_READWRITE);

return;

}

#endif

PageTracking::ReportFree((PageAllocator\*)this, address, AutoSystemInfo::PageSize \* pageCount);

DListBase<PageSegmentBase<T>> \* fromSegmentList = GetSegmentList(segment);

Assert(fromSegmentList != nullptr);

/\*\*

\* The logic here is as follows:

\* - If we have sufficient pages already, such that the newly free pages are going

\* to cause us to exceed the threshold of free pages we want:

\* - First check and see if we have empty segments. If we do, just release that

\* entire segment back to the operating system, and add the current segments

\* free pages to our free page pool

\* - Otherwise, if there are no empty segments (i.e our memory is fragmented),

\* decommit the pages that are being released so that they don't count towards

\* our working set

\* - If we don't have enough pages:

\* - If we're in the free page queuing mode where we have a "pages to zero out" queue

\* put it in that queue and we're done

\* - Otherwise, zero it out, and add it to the free page pool

\* Now that we've either decommited or freed the pages in the segment,

\* move the segment to the right segment list

\*/

if (this->freePageCount + pageCount > maxFreePageCount)

{

// Release a whole segment if possible to reduce the number of VirtualFree and fragmentation

if (!ZeroPages() && !emptySegments.Empty())

{

Assert(emptySegments.Head().GetDecommitPageCount() == 0);

LogFreeSegment(&emptySegments.Head());

emptySegments.RemoveHead(&NoThrowNoMemProtectHeapAllocator::Instance);

this->freePageCount -= maxAllocPageCount;

#if DBG

UpdateMinimum(this->debugMinFreePageCount, this->freePageCount);

memset(address, DbgMemFill, AutoSystemInfo::PageSize \* pageCount);

#endif

segment->ReleasePages(address, pageCount);

LogFreePages(pageCount);

this->AddFreePageCount(pageCount);

}

else

{

segment->DecommitPages<false>(address, pageCount);

LogFreePages(pageCount);

LogDecommitPages(pageCount);

#if DBG\_DUMP

this->decommitPageCount += pageCount;

#endif

}

}

else

{

if (QueueZeroPages())

{

Assert(HasZeroPageQueue());

AddPageToZeroQueue(address, pageCount, segment);

return;

}

this->FillFreePages((char \*)address, pageCount);

segment->ReleasePages(address, pageCount);

LogFreePages(pageCount);

this->AddFreePageCount(pageCount);

}

TransferSegment(segment, fromSegmentList);

}

template<class T>

typename PageAllocatorBase<T>::FreePageEntry \*

PageAllocatorBase<T>::PopPendingZeroPage()

{

Assert(HasZeroPageQueue());

return (PageAllocatorBase<T>::FreePageEntry \*)::InterlockedPopEntrySList(&(((PageAllocatorBase<T>::ZeroPageQueue \*) backgroundPageQueue)->pendingZeroPageList));

}

template<typename T>

void

PageAllocatorBase<T>::AddPageToZeroQueue(\_\_in void \* address, uint pageCount, \_\_in PageSegmentBase<T> \* pageSegment)

{

Assert(HasZeroPageQueue());

Assert(pageSegment->GetAllocator() == this);

FreePageEntry \* entry = (FreePageEntry \*)address;

entry->segment = pageSegment;

entry->pageCount = pageCount;

::InterlockedPushEntrySList(&(((ZeroPageQueue \*)backgroundPageQueue)->pendingZeroPageList), entry);

this->hasZeroQueuedPages = true;

}

template<typename T>

void

PageAllocatorBase<T>::TransferSegment(PageSegmentBase<T> \* segment, DListBase<PageSegmentBase<T>> \* fromSegmentList)

{

DListBase<PageSegmentBase<T>> \* toSegmentList = GetSegmentList(segment);

if (fromSegmentList != toSegmentList)

{

if (toSegmentList)

{

AssertMsg(segment->GetSecondaryAllocator() == nullptr || fromSegmentList != &fullSegments || segment->GetSecondaryAllocator()->CanAllocate(),

"If it's being moved from a full segment it should be able to do secondary allocations");

fromSegmentList->MoveElementTo(segment, toSegmentList);

}

else

{

LogFreePartiallyDecommitedPageSegment(segment);

fromSegmentList->RemoveElement(&NoThrowNoMemProtectHeapAllocator::Instance, segment);

#if DBG\_DUMP

this->decommitPageCount -= maxAllocPageCount;

#endif

}

}

}

template<typename T>

void

PageAllocatorBase<T>::BackgroundZeroQueuedPages()

{

Assert(HasZeroPageQueue());

AutoCriticalSection autocs(&backgroundPageQueue->backgroundPageQueueCriticalSection);

ZeroQueuedPages();

}

template<typename T>

void

PageAllocatorBase<T>::ZeroQueuedPages()

{

Assert(HasZeroPageQueue());

while (true)

{

FreePageEntry \* freePageEntry = PopPendingZeroPage();

if (freePageEntry == nullptr)

{

break;

}

PageSegmentBase<T> \* segment = freePageEntry->segment;

uint pageCount = freePageEntry->pageCount;

memset(freePageEntry, 0, pageCount \* AutoSystemInfo::PageSize);

QueuePages(freePageEntry, pageCount, segment);

}

this->hasZeroQueuedPages = false;

}

template<typename T>

void

PageAllocatorBase<T>::BackgroundReleasePages(void \* address, uint pageCount, PageSegmentBase<T> \* segment)

{

FillFreePages(address, pageCount);

QueuePages(address, pageCount, segment);

}

template<typename T>

void

PageAllocatorBase<T>::QueuePages(void \* address, uint pageCount, PageSegmentBase<T> \* segment)

{

Assert(backgroundPageQueue);

FreePageEntry \* freePageEntry = (FreePageEntry \*)address;

freePageEntry->segment = segment;

freePageEntry->pageCount = pageCount;

::InterlockedPushEntrySList(&backgroundPageQueue->freePageList, freePageEntry);

}

template<typename T>

void

PageAllocatorBase<T>::FlushBackgroundPages()

{

Assert(!HasMultiThreadAccess());

Assert(backgroundPageQueue);

// We can have additional pages queued up to be zeroed out here

// and that's okay since they'll eventually be zeroed out before being flushed

uint newFreePages = 0;

while (true)

{

FreePageEntry \* freePageEntry = (FreePageEntry \*)::InterlockedPopEntrySList(&backgroundPageQueue->freePageList);

if (freePageEntry == nullptr)

{

break;

}

PageSegmentBase<T> \* segment = freePageEntry->segment;

uint pageCount = freePageEntry->pageCount;

DListBase<PageSegmentBase<T>> \* fromSegmentList = GetSegmentList(segment);

Assert(fromSegmentList != nullptr);

memset(freePageEntry, 0, sizeof(FreePageEntry));

segment->ReleasePages(freePageEntry, pageCount);

newFreePages += pageCount;

TransferSegment(segment, fromSegmentList);

}

LogFreePages(newFreePages);

PAGE\_ALLOC\_VERBOSE\_TRACE(L"New free pages: %d\n", newFreePages);

this->AddFreePageCount(newFreePages);

}

template<typename T>

void

PageAllocatorBase<T>::SuspendIdleDecommit()

{

#ifdef IDLE\_DECOMMIT\_ENABLED

if (this->idleDecommitEnterCount != 0)

{

return;

}

Assert(this->IsIdleDecommitPageAllocator());

((IdleDecommitPageAllocator \*)this)->cs.Enter();

PAGE\_ALLOC\_VERBOSE\_TRACE(L"SuspendIdleDecommit");

#endif

}

template<typename T>

void

PageAllocatorBase<T>::ResumeIdleDecommit()

{

#ifdef IDLE\_DECOMMIT\_ENABLED

if (this->idleDecommitEnterCount != 0)

{

return;

}

Assert(this->IsIdleDecommitPageAllocator());

PAGE\_ALLOC\_VERBOSE\_TRACE(L"ResumeIdleDecommit");

((IdleDecommitPageAllocator \*)this)->cs.Leave();

#endif

}

template<typename T>

void

PageAllocatorBase<T>::DecommitNow(bool all)

{

Assert(!HasMultiThreadAccess());

#if DBG\_DUMP

size\_t deleteCount = 0;

#endif

// First, drain the zero page queue.

// This will cause the free page count to be accurate

if (HasZeroPageQueue())

{

int numZeroPagesFreed = 0;

// There might be queued zero pages. Drain them first

while (true)

{

FreePageEntry \* freePageEntry = PopPendingZeroPage();

if (freePageEntry == nullptr)

{

break;

}

PAGE\_ALLOC\_TRACE\_AND\_STATS(L"Freeing page from zero queue");

PageSegmentBase<T> \* segment = freePageEntry->segment;

uint pageCount = freePageEntry->pageCount;

numZeroPagesFreed += pageCount;

DListBase<PageSegmentBase<T>> \* fromSegmentList = GetSegmentList(segment);

Assert(fromSegmentList != nullptr);

// Check for all here, since the actual free page count can't be determined

// until we've flushed the zeroed page queue

if (all)

{

// Decommit them immediately if we are decommitting all pages.

segment->DecommitPages<false>(freePageEntry, pageCount);

LogFreePages(pageCount);

LogDecommitPages(pageCount);

if (segment->IsAllDecommitted())

{

LogFreePartiallyDecommitedPageSegment(segment);

fromSegmentList->RemoveElement(&NoThrowNoMemProtectHeapAllocator::Instance, segment);

#if DBG\_DUMP

deleteCount += maxAllocPageCount;

#endif

continue;

}

}

else

{

// Zero them and release them in case we don't decommit them.

memset(freePageEntry, 0, pageCount \* AutoSystemInfo::PageSize);

segment->ReleasePages(freePageEntry, pageCount);

LogFreePages(pageCount);

}

TransferSegment(segment, fromSegmentList);

}

// Take the lock to make sure the recycler thread has finished zeroing out the pages after

// we drained the queue

backgroundPageQueue->backgroundPageQueueCriticalSection.Enter();

this->hasZeroQueuedPages = false;

Assert(!this->HasZeroQueuedPages());

backgroundPageQueue->backgroundPageQueueCriticalSection.Leave();

FlushBackgroundPages();

}

if (this->freePageCount == 0)

{

Assert(debugMinFreePageCount == 0);

return;

}

PAGE\_ALLOC\_TRACE\_AND\_STATS(L"Decommit now");

// minFreePageCount is not updated on every page allocate,

// so we have to do a final update here.

UpdateMinFreePageCount();

size\_t newFreePageCount;

if (all)

{

newFreePageCount = this->GetFreePageLimit();

PAGE\_ALLOC\_TRACE\_AND\_STATS(L"Full decommit");

}

else

{

// Decommit half the min free page count since last partial decommit

Assert(this->minFreePageCount <= this->freePageCount);

newFreePageCount = this->freePageCount - (this->minFreePageCount / 2);

// Ensure we don't decommit down to fewer than our partial decommit minimum

newFreePageCount = max(newFreePageCount, static\_cast<size\_t>(MinPartialDecommitFreePageCount));

PAGE\_ALLOC\_TRACE\_AND\_STATS(L"Partial decommit");

}

if (newFreePageCount >= this->freePageCount)

{

PAGE\_ALLOC\_TRACE\_AND\_STATS(L"No pages to decommit");

return;

}

size\_t pageToDecommit = this->freePageCount - newFreePageCount;

PAGE\_ALLOC\_TRACE\_AND\_STATS(L"Decommit page count = %d", pageToDecommit);

PAGE\_ALLOC\_TRACE\_AND\_STATS(L"Free page count = %d", this->freePageCount);

PAGE\_ALLOC\_TRACE\_AND\_STATS(L"New free page count = %d", newFreePageCount);

#if DBG\_DUMP

size\_t decommitCount = 0;

#endif

// decommit from page that already has other decommitted page already

{

DListBase<PageSegmentBase<T>>::EditingIterator i(&decommitSegments);

while (pageToDecommit > 0 && i.Next())

{

size\_t pageDecommited = i.Data().DecommitFreePages(pageToDecommit);

LogDecommitPages(pageDecommited);

#if DBG\_DUMP

decommitCount += pageDecommited;

#endif

if (i.Data().GetDecommitPageCount() == maxAllocPageCount)

{

LogFreePartiallyDecommitedPageSegment(&i.Data());

i.RemoveCurrent(&NoThrowNoMemProtectHeapAllocator::Instance);

#if DBG\_DUMP

deleteCount += maxAllocPageCount;

#endif

}

pageToDecommit -= pageDecommited;

}

}

// decommit pages that are empty

while (pageToDecommit > 0 && !emptySegments.Empty())

{

if (pageToDecommit >= maxAllocPageCount)

{

Assert(emptySegments.Head().GetDecommitPageCount() == 0);

LogFreeSegment(&emptySegments.Head());

emptySegments.RemoveHead(&NoThrowNoMemProtectHeapAllocator::Instance);

pageToDecommit -= maxAllocPageCount;

#if DBG\_DUMP

decommitCount += maxAllocPageCount;

deleteCount += maxAllocPageCount;

#endif

}

else

{

size\_t pageDecommited = emptySegments.Head().DecommitFreePages(pageToDecommit);

LogDecommitPages(pageDecommited);

#if DBG\_DUMP

decommitCount += pageDecommited;

#endif

Assert(pageDecommited == pageToDecommit);

emptySegments.MoveHeadTo(&decommitSegments);

pageToDecommit = 0;

}

}

{

DListBase<PageSegmentBase<T>>::EditingIterator i(&segments);

while (pageToDecommit > 0 && i.Next())

{

size\_t pageDecommited = i.Data().DecommitFreePages(pageToDecommit);

LogDecommitPages(pageDecommited);

#if DBG\_DUMP

decommitCount += pageDecommited;

#endif

Assert(i.Data().GetDecommitPageCount() != 0);

Assert(i.Data().GetDecommitPageCount() <= maxAllocPageCount);

i.MoveCurrentTo(&decommitSegments);

pageToDecommit -= pageDecommited;

}

}

Assert(pageToDecommit == 0);

#if DBG\_DUMP

Assert(this->freePageCount == newFreePageCount + decommitCount);

#endif

this->freePageCount = newFreePageCount;

#if DBG

UpdateMinimum(this->debugMinFreePageCount, this->freePageCount);

Check();

#endif

#if DBG\_DUMP

this->decommitPageCount += (decommitCount - deleteCount);

if (CUSTOM\_PHASE\_TRACE1(this->pageAllocatorFlagTable, Js::PageAllocatorPhase))

{

if (CUSTOM\_PHASE\_STATS1(this->pageAllocatorFlagTable, Js::PageAllocatorPhase))

{

Output::Print(L" After decommit now:\n");

this->DumpStats();

}

Output::Flush();

}

#endif

}

template<typename T>

void

PageAllocatorBase<T>::AddReservedBytes(size\_t bytes)

{

reservedBytes += bytes;

#ifdef PERF\_COUNTERS

GetReservedSizeCounter() += bytes;

GetTotalReservedSizeCounter() += bytes;

#endif

}

template<typename T>

void

PageAllocatorBase<T>::SubReservedBytes(size\_t bytes)

{

reservedBytes -= bytes;

#ifdef PERF\_COUNTERS

GetReservedSizeCounter() -= bytes;

GetTotalReservedSizeCounter() -= bytes;

#endif

}

template<typename T>

void

PageAllocatorBase<T>::AddCommittedBytes(size\_t bytes)

{

committedBytes += bytes;

#ifdef PERF\_COUNTERS

GetCommittedSizeCounter() += bytes;

GetTotalCommittedSizeCounter() += bytes;

#endif

}

template<typename T>

void

PageAllocatorBase<T>::SubCommittedBytes(size\_t bytes)

{

committedBytes -= bytes;

#ifdef PERF\_COUNTERS

GetCommittedSizeCounter() -= bytes;

GetTotalCommittedSizeCounter() -= bytes;

#endif

}

template<typename T>

void

PageAllocatorBase<T>::AddUsedBytes(size\_t bytes)

{

usedBytes += bytes;

#if defined(\_M\_X64\_OR\_ARM64)

size\_t lastTotalUsedBytes = ::InterlockedExchangeAdd64((volatile LONG64 \*)&totalUsedBytes, bytes);

#else

DWORD lastTotalUsedBytes = ::InterlockedExchangeAdd(&totalUsedBytes, bytes);

#endif

if (totalUsedBytes > maxUsedBytes)

{

maxUsedBytes = totalUsedBytes;

}

// ETW events from different threads may be reported out of order, producing an

// incorrect representation of current used bytes in the process. We've determined that this is an

// acceptable issue, which will be mitigated at the level of the application consuming the event.

JS\_ETW(EventWriteJSCRIPT\_PAGE\_ALLOCATOR\_USED\_SIZE(lastTotalUsedBytes + bytes));

#ifndef ENABLE\_JS\_ETW

Unused(lastTotalUsedBytes);

#endif

#ifdef PERF\_COUNTERS

GetUsedSizeCounter() += bytes;

GetTotalUsedSizeCounter() += bytes;

#endif

}

template<typename T>

void

PageAllocatorBase<T>::SubUsedBytes(size\_t bytes)

{

Assert(bytes <= usedBytes);

Assert(bytes <= totalUsedBytes);

usedBytes -= bytes;

#if defined(\_M\_X64\_OR\_ARM64)

size\_t lastTotalUsedBytes = ::InterlockedExchangeAdd64((volatile LONG64 \*)&totalUsedBytes, -(LONG64)bytes);

#else

DWORD lastTotalUsedBytes = ::InterlockedExchangeSubtract(&totalUsedBytes, bytes);

#endif

// ETW events from different threads may be reported out of order, producing an

// incorrect representation of current used bytes in the process. We've determined that this is an

// acceptable issue, which will be mitigated at the level of the application consuming the event.

JS\_ETW(EventWriteJSCRIPT\_PAGE\_ALLOCATOR\_USED\_SIZE(lastTotalUsedBytes - bytes));

#ifndef ENABLE\_JS\_ETW

Unused(lastTotalUsedBytes);

#endif

#ifdef PERF\_COUNTERS

GetUsedSizeCounter() -= bytes;

GetTotalUsedSizeCounter() -= bytes;

#endif

}

template<typename T>

void

PageAllocatorBase<T>::AddNumberOfSegments(size\_t segmentCount)

{

numberOfSegments += segmentCount;

}

template<typename T>

void

PageAllocatorBase<T>::SubNumberOfSegments(size\_t segmentCount)

{

numberOfSegments -= segmentCount;

}

template<typename T>

void

PageAllocatorBase<T>::IntegrateSegments(DListBase<PageSegmentBase<T>>& segmentList, uint segmentCount, size\_t pageCount)

{

#if DBG

size\_t debugPageCount = 0;

uint debugSegmentCount = 0;

DListBase<PageSegmentBase<T>>::Iterator i(&segmentList);

while (i.Next())

{

Assert(i.Data().GetAllocator() == this);

debugSegmentCount++;

debugPageCount += i.Data().GetPageCount();

}

Assert(debugSegmentCount == segmentCount);

Assert(debugPageCount == pageCount);

#endif

LogAllocSegment(segmentCount, pageCount);

LogAllocPages(pageCount);

this->SuspendIdleDecommit();

segmentList.MoveTo(&this->fullSegments);

this->ResumeIdleDecommit();

}

template<typename T>

void

PageAllocatorBase<T>::LogAllocSegment(SegmentBase<T> \* segment)

{

LogAllocSegment(1, segment->GetPageCount());

}

template<typename T>

void

PageAllocatorBase<T>::LogAllocSegment(uint segmentCount, size\_t pageCount)

{

size\_t bytes = pageCount \* AutoSystemInfo::PageSize;

AddReservedBytes(bytes);

AddCommittedBytes(bytes);

AddNumberOfSegments(segmentCount);

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->allocSegmentCount += segmentCount;

this->memoryData->allocSegmentBytes += pageCount \* AutoSystemInfo::PageSize;

this->memoryData->currentCommittedPageCount += pageCount;

this->memoryData->peakCommittedPageCount = max(this->memoryData->peakCommittedPageCount, this->memoryData->currentCommittedPageCount);

}

#endif

}

template<typename T>

void

PageAllocatorBase<T>::LogFreeSegment(SegmentBase<T> \* segment)

{

size\_t bytes = segment->GetPageCount() \* AutoSystemInfo::PageSize;

SubCommittedBytes(bytes);

SubReservedBytes(bytes);

SubNumberOfSegments(1);

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->releaseSegmentCount++;

this->memoryData->releaseSegmentBytes += segment->GetPageCount() \* AutoSystemInfo::PageSize;

this->memoryData->currentCommittedPageCount -= segment->GetPageCount();

}

#endif

}

template<typename T>

void

PageAllocatorBase<T>::LogFreeDecommittedSegment(SegmentBase<T> \* segment)

{

SubReservedBytes(segment->GetPageCount() \* AutoSystemInfo::PageSize);

SubNumberOfSegments(1);

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->releaseSegmentCount++;

this->memoryData->releaseSegmentBytes += segment->GetPageCount() \* AutoSystemInfo::PageSize;

}

#endif

}

template<typename T>

void

PageAllocatorBase<T>::LogFreePages(size\_t pageCount)

{

SubUsedBytes(pageCount \* AutoSystemInfo::PageSize);

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->releasePageCount += pageCount;

}

#endif

}

template<typename T>

void

PageAllocatorBase<T>::LogFreePartiallyDecommitedPageSegment(PageSegmentBase<T> \* pageSegment)

{

AddCommittedBytes(pageSegment->GetDecommitPageCount() \* AutoSystemInfo::PageSize);

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->currentCommittedPageCount += pageSegment->GetDecommitPageCount();

}

#endif

LogFreeSegment(pageSegment);

}

template<typename T>

void

PageAllocatorBase<T>::LogAllocPages(size\_t pageCount)

{

AddUsedBytes(pageCount \* AutoSystemInfo::PageSize);

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->allocPageCount += pageCount;

}

#endif

}

template<typename T>

void

PageAllocatorBase<T>::LogRecommitPages(size\_t pageCount)

{

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->recommitPageCount += pageCount;

}

#endif

LogCommitPages(pageCount);

}

template<typename T>

void

PageAllocatorBase<T>::LogCommitPages(size\_t pageCount)

{

AddCommittedBytes(pageCount \* AutoSystemInfo::PageSize);

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->currentCommittedPageCount += pageCount;

this->memoryData->peakCommittedPageCount = max(this->memoryData->peakCommittedPageCount, this->memoryData->currentCommittedPageCount);

}

#endif

}

template<typename T>

void

PageAllocatorBase<T>::LogDecommitPages(size\_t pageCount)

{

SubCommittedBytes(pageCount \* AutoSystemInfo::PageSize);

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->decommitPageCount += pageCount;

this->memoryData->currentCommittedPageCount -= pageCount;

}

#endif

}

#if DBG\_DUMP

template<typename T>

void

PageAllocatorBase<T>::DumpStats() const

{

Output::Print(L" Full/Partial/Empty/Decommit/Large Segments: %4d %4d %4d %4d %4d\n",

fullSegments.Count(), segments.Count(), emptySegments.Count(), decommitSegments.Count(), largeSegments.Count());

Output::Print(L" Free/Decommit/Min Free Pages : %4d %4d %4d\n",

this->freePageCount, this->decommitPageCount, this->minFreePageCount);

}

#endif

#if DBG

template<typename T>

void

PageAllocatorBase<T>::Check()

{

Assert(!this->HasZeroQueuedPages());

size\_t currentFreePageCount = 0;

DListBase<PageSegmentBase<T>>::Iterator segmentsIterator(&segments);

while (segmentsIterator.Next())

{

currentFreePageCount += segmentsIterator.Data().GetFreePageCount();

}

DListBase<PageSegmentBase<T>>::Iterator fullSegmentsIterator(&fullSegments);

while (fullSegmentsIterator.Next())

{

currentFreePageCount += fullSegmentsIterator.Data().GetFreePageCount();

}

DListBase<PageSegmentBase<T>>::Iterator emptySegmentsIterator(&emptySegments);

while (emptySegmentsIterator.Next())

{

currentFreePageCount += emptySegmentsIterator.Data().GetFreePageCount();

}

DListBase<PageSegmentBase<T>>::Iterator decommitSegmentsIterator(&decommitSegments);

while (decommitSegmentsIterator.Next())

{

currentFreePageCount += decommitSegmentsIterator.Data().GetFreePageCount();

}

Assert(freePageCount == currentFreePageCount);

}

#endif

template<typename T>

HeapPageAllocator<T>::HeapPageAllocator(AllocationPolicyManager \* policyManager, bool allocXdata, bool excludeGuardPages) :

PageAllocatorBase(policyManager,

Js::Configuration::Global.flags,

PageAllocatorType\_CustomHeap,

/\*maxFreePageCount\*/ 0,

/\*zeroPages\*/ false,

/\*zeroPageQueue\*/ nullptr,

/\*maxAllocPageCount\*/ allocXdata ? (DefaultMaxAllocPageCount - XDATA\_RESERVE\_PAGE\_COUNT) : DefaultMaxAllocPageCount,

/\*secondaryAllocPageCount=\*/ allocXdata ? XDATA\_RESERVE\_PAGE\_COUNT : 0,

/\*stopAllocationOnOutOfMemory\*/ false,

excludeGuardPages),

allocXdata(allocXdata)

{

}

template<typename T>

void

HeapPageAllocator<T>::ReleaseDecommited(void \* address, size\_t pageCount, \_\_in void \* segmentParam)

{

SegmentBase<T> \* segment = (SegmentBase<T>\*) segmentParam;

if (pageCount > this->maxAllocPageCount)

{

Assert(address == segment->GetAddress());

Assert(pageCount == segment->GetAvailablePageCount());

this->ReleaseDecommitedSegment(segment);

}

else

{

Assert(pageCount <= UINT\_MAX);

this->TrackDecommitedPages(address, (uint)pageCount, (PageSegment \*)segment);

}

}

template<typename T>

void

HeapPageAllocator<T>::ReleaseDecommitedSegment(\_\_in SegmentBase<T>\* segment)

{

ASSERT\_THREAD();

LogFreeDecommittedSegment(segment);

largeSegments.RemoveElement(&NoThrowNoMemProtectHeapAllocator::Instance, segment);

}

// decommit the page but don't release it

template<typename T>

void

HeapPageAllocator<T>::DecommitPages(\_\_in char\* address, size\_t pageCount = 1)

{

Assert(pageCount <= MAXUINT32);

#pragma prefast(suppress:\_\_WARNING\_WIN32UNRELEASEDVADS, "The remainder of the clean-up is done later.");

virtualAllocator->Free(address, pageCount \* AutoSystemInfo::PageSize, MEM\_DECOMMIT);

LogFreePages(pageCount);

LogDecommitPages(pageCount);

}

template<typename TVirtualAlloc>

template <typename T>

void PageAllocatorBase<TVirtualAlloc>::ReleaseSegmentList(DListBase<T> \* segmentList)

{

segmentList->Clear(&NoThrowNoMemProtectHeapAllocator::Instance);

}

template<typename T>

BOOL

HeapPageAllocator<T>::ProtectPages(\_\_in char\* address, size\_t pageCount, \_\_in void\* segmentParam, DWORD dwVirtualProtectFlags, DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag)

{

SegmentBase<T> \* segment = (SegmentBase<T>\*)segmentParam;

#if DBG

Assert(address >= segment->GetAddress());

Assert(((uint)(((char \*)address) - segment->GetAddress()) <= (segment->GetPageCount() - pageCount) \* AutoSystemInfo::PageSize));

Assert(dwOldVirtualProtectFlags != NULL);

if (IsPageSegment(segment))

{

PageSegmentBase<T> \* pageSegment = static\_cast<PageSegmentBase<T>\*>(segment);

AssertMsg(pageCount <= MAXUINT32, "PageSegment should always be smaller than 4G pages");

Assert(!pageSegment->IsFreeOrDecommitted(address, static\_cast<uint>(pageCount)));

}

#endif

#if DBG\_DUMP || defined(RECYCLER\_TRACE)

if (this->pageAllocatorFlagTable.IsEnabled(Js::TraceProtectPagesFlag))

{

Output::Print(L"VirtualProtect(0x%p, %d, %d, %d)\n", address, pageCount, pageCount \* AutoSystemInfo::PageSize, dwVirtualProtectFlags);

}

#endif

// check address alignment, and that the address is in correct range

if (((uintptr\_t)address & (AutoSystemInfo::PageSize - 1)) != 0

|| address < segment->GetAddress()

|| ((uint)(((char \*)address) - segment->GetAddress()) > (segment->GetPageCount() - pageCount) \* AutoSystemInfo::PageSize))

{

CustomHeap\_BadPageState\_fatal\_error((ULONG\_PTR)this);

return FALSE;

}

MEMORY\_BASIC\_INFORMATION memBasicInfo;

// check old protection on all pages about to change, ensure the fidelity

size\_t bytes = VirtualQuery(address, &memBasicInfo, sizeof(memBasicInfo));

if (bytes == 0

|| memBasicInfo.RegionSize < pageCount \* AutoSystemInfo::PageSize

|| desiredOldProtectFlag != memBasicInfo.Protect

)

{

CustomHeap\_BadPageState\_fatal\_error((ULONG\_PTR)this);

return FALSE;

}

\*dwOldVirtualProtectFlags = memBasicInfo.Protect;

DWORD oldProtect; // this is only for first page

BOOL retVal = ::VirtualProtect(address, pageCount \* AutoSystemInfo::PageSize, dwVirtualProtectFlags, &oldProtect);

Assert(oldProtect == \*dwOldVirtualProtectFlags);

return retVal;

}

template<typename T>

void

HeapPageAllocator<T>::TrackDecommitedPages(void \* address, uint pageCount, \_\_in void\* segmentParam)

{

PageSegmentBase<T> \* segment = (PageSegmentBase<T>\*)segmentParam;

ASSERT\_THREAD();

Assert(!HasMultiThreadAccess());

Assert(pageCount <= this->maxAllocPageCount);

DListBase<PageSegmentBase<T>> \* fromSegmentList = GetSegmentList(segment);

// Update the state of the segment with the decommitted pages

segment->DecommitPages<true>(address, pageCount);

// Move the segment to its appropriate list

TransferSegment(segment, fromSegmentList);

}

template<typename T>

bool HeapPageAllocator<T>::AllocSecondary(void\* segmentParam, ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation)

{

SegmentBase<T> \* segment = (SegmentBase<T> \*)segmentParam;

Assert(segment->GetSecondaryAllocator());

bool success;

if (IsPageSegment(segment))

{

PageSegmentBase<T>\* pageSegment = static\_cast<PageSegmentBase<T>\*>(segment);

// We should get the segment list BEFORE xdata allocation happens.

DListBase<PageSegmentBase<T>> \* fromSegmentList = GetSegmentList(pageSegment);

success = segment->GetSecondaryAllocator()->Alloc(functionStart, functionSize, pdataCount, xdataSize, allocation);

// If no more XDATA allocations can take place.

if (success && !pageSegment->CanAllocSecondary() && fromSegmentList != &fullSegments)

{

AssertMsg(GetSegmentList(pageSegment) == &fullSegments, "This segment should now be in the full list if it can't allocate secondary");

OUTPUT\_TRACE(Js::EmitterPhase, L"XDATA Wasted pages:%u\n", pageSegment->GetFreePageCount());

this->freePageCount -= pageSegment->GetFreePageCount();

fromSegmentList->MoveElementTo(pageSegment, &fullSegments);

#if DBG

UpdateMinimum(this->debugMinFreePageCount, this->freePageCount);

#endif

}

}

else

{

// A large segment should always be able to do secondary allocations

Assert(segment->CanAllocSecondary());

success = segment->GetSecondaryAllocator()->Alloc(functionStart, functionSize, pdataCount, xdataSize, allocation);

}

#ifdef \_M\_X64

// In ARM it's OK to have xdata size be 0

AssertMsg(allocation->address != nullptr, "All segments that cannot allocate xdata should have been already moved to full segments list");

#endif

return success;

}

template<typename T>

void HeapPageAllocator<T>::ReleaseSecondary(const SecondaryAllocation& allocation, void\* segmentParam)

{

SegmentBase<T> \* segment = (SegmentBase<T>\*)segmentParam;

Assert(allocation.address != nullptr);

Assert(segment->GetSecondaryAllocator());

if (IsPageSegment(segment))

{

PageSegmentBase<T>\* pageSegment = static\_cast<PageSegmentBase<T>\*>(segment);

auto fromList = GetSegmentList(pageSegment);

pageSegment->GetSecondaryAllocator()->Release(allocation);

auto toList = GetSegmentList(pageSegment);

if (fromList != toList)

{

OUTPUT\_TRACE(Js::EmitterPhase, L"XDATA reclaimed pages:%u\n", pageSegment->GetFreePageCount());

fromList->MoveElementTo(pageSegment, toList);

AssertMsg(fromList == &fullSegments, "Releasing a secondary allocator should make a state change only if the segment was originally in the full list");

AssertMsg(pageSegment->CanAllocSecondary(), "It should be allocate secondary now");

this->AddFreePageCount(pageSegment->GetFreePageCount());

}

}

else

{

Assert(segment->CanAllocSecondary());

segment->GetSecondaryAllocator()->Release(allocation);

}

}

template<typename T>

bool

HeapPageAllocator<T>::IsAddressFromAllocator(\_\_in void\* address)

{

DListBase<PageSegmentBase<T>>::Iterator segmentsIterator(&segments);

while (segmentsIterator.Next())

{

if (IsAddressInSegment(address, segmentsIterator.Data()))

{

return true;

}

}

DListBase<PageSegmentBase<T>>::Iterator fullSegmentsIterator(&fullSegments);

while (fullSegmentsIterator.Next())

{

if (IsAddressInSegment(address, fullSegmentsIterator.Data()))

{

return true;

}

}

DListBase<SegmentBase<T>>::Iterator largeSegmentsIterator(&largeSegments);

while (largeSegmentsIterator.Next())

{

if (IsAddressInSegment(address, largeSegmentsIterator.Data()))

{

return true;

}

}

DListBase<PageSegmentBase<T>>::Iterator decommitSegmentsIterator(&decommitSegments);

while (decommitSegmentsIterator.Next())

{

if (IsAddressInSegment(address, decommitSegmentsIterator.Data()))

{

return true;

}

}

return false;

}

template<typename T>

bool

PageAllocatorBase<T>::IsAddressInSegment(\_\_in void\* address, const PageSegmentBase<T>& segment)

{

bool inSegment = this->IsAddressInSegment(address, static\_cast<const SegmentBase<T>&>(segment));

if (inSegment)

{

return !segment.IsFreeOrDecommitted(address);

}

return inSegment;

}

template<typename T>

bool

PageAllocatorBase<T>::IsAddressInSegment(\_\_in void\* address, const SegmentBase<T>& segment)

{

return segment.IsInSegment(address);

}

#if PDATA\_ENABLED

#include "Memory\XDataAllocator.h"

template<typename T>

bool HeapPageAllocator<T>::CreateSecondaryAllocator(SegmentBase<T>\* segment, SecondaryAllocator\*\* allocator)

{

Assert(segment->GetAllocator() == this);

// If we are not allocating xdata there is nothing to do

// ARM might allocate XDATA but not have a reserved region for it (no secondary alloc reserved space)

if(!allocXdata)

{

Assert(segment->GetSecondaryAllocSize() == 0);

\*allocator = nullptr;

return true;

}

XDataAllocator\* secondaryAllocator = HeapNewNoThrow(XDataAllocator, (BYTE\*)segment->GetSecondaryAllocStartAddress(), segment->GetSecondaryAllocSize());

bool success = false;

if(secondaryAllocator)

{

if(secondaryAllocator->Initialize((BYTE\*)segment->GetAddress(), (BYTE\*)segment->GetEndAddress()))

{

success = true;

}

else

{

HeapDelete(secondaryAllocator);

secondaryAllocator = nullptr;

}

}

\*allocator = secondaryAllocator;

return success;

}

#endif

template<typename T>

uint PageSegmentBase<T>::GetCountOfFreePages() const

{

return this->freePages.Count();

}

template<typename T>

uint PageSegmentBase<T>::GetNextBitInFreePagesBitVector(uint index) const

{

return this->freePages.GetNextBit(index);

}

template<typename T>

BOOLEAN PageSegmentBase<T>::TestRangeInFreePagesBitVector(uint index, uint pageCount) const

{

return this->freePages.TestRange(index, pageCount);

}

template<typename T>

BOOLEAN PageSegmentBase<T>::TestInFreePagesBitVector(uint index) const

{

return this->freePages.Test(index);

}

template<typename T>

void PageSegmentBase<T>::ClearAllInFreePagesBitVector()

{

return this->freePages.ClearAll();

}

template<typename T>

void PageSegmentBase<T>::ClearRangeInFreePagesBitVector(uint index, uint pageCount)

{

return this->freePages.ClearRange(index, pageCount);

}

template<typename T>

void PageSegmentBase<T>::SetRangeInFreePagesBitVector(uint index, uint pageCount)

{

return this->freePages.SetRange(index, pageCount);

}

template<typename T>

void PageSegmentBase<T>::ClearBitInFreePagesBitVector(uint index)

{

return this->freePages.Clear(index);

}

template<typename T>

BOOLEAN PageSegmentBase<T>::TestInDecommitPagesBitVector(uint index) const

{

return this->decommitPages.Test(index);

}

template<typename T>

BOOLEAN PageSegmentBase<T>::TestRangeInDecommitPagesBitVector(uint index, uint pageCount) const

{

return this->decommitPages.TestRange(index, pageCount);

}

template<typename T>

void PageSegmentBase<T>::SetRangeInDecommitPagesBitVector(uint index, uint pageCount)

{

return this->decommitPages.SetRange(index, pageCount);

}

template<typename T>

void PageSegmentBase<T>::ClearRangeInDecommitPagesBitVector(uint index, uint pageCount)

{

return this->decommitPages.ClearRange(index, pageCount);

}

template<typename T>

uint PageSegmentBase<T>::GetCountOfDecommitPages() const

{

return this->decommitPages.Count();

}

template<typename T>

void PageSegmentBase<T>::SetBitInDecommitPagesBitVector(uint index)

{

this->decommitPages.Set(index);

}

template<typename T>

template <bool noPageAligned>

char \* PageSegmentBase<T>::DoAllocDecommitPages(uint pageCount, PageHeapMode pageHeapFlags)

{

return this->AllocDecommitPages<PageSegmentBase<T>::PageBitVector, noPageAligned>(pageCount, this->freePages, this->decommitPages, pageHeapFlags);

}

template<typename T>

uint PageSegmentBase<T>::GetMaxPageCount()

{

return MaxPageCount;

}

//Instantiate all the Templates in this class below.

template class PageAllocatorBase < PreReservedVirtualAllocWrapper >;

template class PageAllocatorBase < VirtualAllocWrapper >;

template class HeapPageAllocator < PreReservedVirtualAllocWrapper >;

template class HeapPageAllocator < VirtualAllocWrapper >;

template class SegmentBase < VirtualAllocWrapper > ;

template class SegmentBase < PreReservedVirtualAllocWrapper >;

template class PageSegmentBase < VirtualAllocWrapper >;

template class PageSegmentBase < PreReservedVirtualAllocWrapper >;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "PageAllocatorDefines.h"

#ifdef PROFILE\_MEM

struct PageMemoryData;

#endif

class CodeGenNumberThreadAllocator;

namespace Memory

{

typedef void\* FunctionTableHandle;

#if DBG\_DUMP && !defined(JD\_PRIVATE)

#define GUARD\_PAGE\_TRACE(...) \

if (Js::Configuration::Global.flags.PrintGuardPageBounds) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#define PAGE\_ALLOC\_TRACE(format, ...) PAGE\_ALLOC\_TRACE\_EX(false, false, format, \_\_VA\_ARGS\_\_)

#define PAGE\_ALLOC\_VERBOSE\_TRACE(format, ...) PAGE\_ALLOC\_TRACE\_EX(true, false, format, \_\_VA\_ARGS\_\_)

#define PAGE\_ALLOC\_TRACE\_AND\_STATS(format, ...) PAGE\_ALLOC\_TRACE\_EX(false, true, format, \_\_VA\_ARGS\_\_)

#define PAGE\_ALLOC\_VERBOSE\_TRACE\_AND\_STATS(format, ...) PAGE\_ALLOC\_TRACE\_EX(true, true, format, \_\_VA\_ARGS\_\_)

#define PAGE\_ALLOC\_TRACE\_EX(verbose, stats, format, ...) \

if (this->pageAllocatorFlagTable.Trace.IsEnabled(Js::PageAllocatorPhase)) \

{ \

if (!verbose || this->pageAllocatorFlagTable.Verbose) \

{ \

Output::Print(L"%p : %p> PageAllocator(%p): ", GetCurrentThreadContextId(), ::GetCurrentThreadId(), this); \

if (debugName != nullptr) \

{ \

Output::Print(L"[%s] ", this->debugName); \

} \

Output::Print(format, \_\_VA\_ARGS\_\_); \

Output::Print(L"\n"); \

if (stats && this->pageAllocatorFlagTable.Stats.IsEnabled(Js::PageAllocatorPhase)) \

{ \

this->DumpStats(); \

} \

Output::Flush(); \

} \

}

#else

#define PAGE\_ALLOC\_TRACE(format, ...)

#define PAGE\_ALLOC\_VERBOSE\_TRACE(format, ...)

#define PAGE\_ALLOC\_TRACE\_AND\_STATS(format, ...)

#define PAGE\_ALLOC\_VERBOSE\_TRACE\_AND\_STATS(format, ...)

#endif

#ifdef \_M\_X64

#define XDATA\_RESERVE\_PAGE\_COUNT (2) // Number of pages per page segment (32 pages) reserved for xdata.

#else

#define XDATA\_RESERVE\_PAGE\_COUNT (0) // ARM uses the heap, so it's not required.

#endif

//

// Allocation done by the secondary allocator

//

struct SecondaryAllocation

{

BYTE\* address; // address of the allocation by the secondary allocator

SecondaryAllocation() : address(nullptr)

{

}

};

//

// For every page segment a page allocator can create a secondary allocator which can have a specified

// number of pages reserved for secondary allocations. These pages are always reserved at the end of the

// segment. The PageAllocator itself cannot allocate from the region demarcated for the secondary allocator.

// Currently this is used for xdata allocations.

//

class SecondaryAllocator

{

public:

virtual bool Alloc(ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* xdata) = 0;

virtual void Release(const SecondaryAllocation& allocation) = 0;

virtual void Delete() = 0;

virtual bool CanAllocate() = 0;

virtual ~SecondaryAllocator() {};

};

/\*

\* A segment is a collection of pages. A page corresponds to the concept of an

\* OS memory page. Segments allocate memory using the OS VirtualAlloc call.

\* It'll allocate the pageCount \* page size number of bytes, the latter being

\* a system-wide constant.

\*/

template<typename TVirtualAlloc>

class SegmentBase

{

public:

SegmentBase(PageAllocatorBase<TVirtualAlloc> \* allocator, size\_t pageCount);

virtual ~SegmentBase();

size\_t GetPageCount() const { return segmentPageCount; }

// Some pages are reserved upfront for secondary allocations

// which are done by a secondary allocator as opposed to the PageAllocator

size\_t GetAvailablePageCount() const { return segmentPageCount - secondaryAllocPageCount; }

char\* GetSecondaryAllocStartAddress() const { return (this->address + GetAvailablePageCount() \* AutoSystemInfo::PageSize); }

uint GetSecondaryAllocSize() const { return this->secondaryAllocPageCount \* AutoSystemInfo::PageSize; }

char\* GetAddress() const { return address; }

char\* GetEndAddress() const { return GetSecondaryAllocStartAddress(); }

bool CanAllocSecondary() { Assert(secondaryAllocator); return secondaryAllocator->CanAllocate(); }

PageAllocatorBase<TVirtualAlloc>\* GetAllocator() const { return allocator; }

bool IsInPreReservedHeapPageAllocator() const;

bool Initialize(DWORD allocFlags, bool excludeGuardPages);

#if DBG

virtual bool IsPageSegment() const

{

return false;

}

#endif

bool IsInSegment(void\* address) const

{

void\* start = static\_cast<void\*>(GetAddress());

void\* end = static\_cast<void\*>(GetEndAddress());

return (address >= start && address < end);

}

bool IsInCustomHeapAllocator() const

{

return this->allocator->type == PageAllocatorType::PageAllocatorType\_CustomHeap;

}

SecondaryAllocator\* GetSecondaryAllocator() { return secondaryAllocator; }

#if defined(\_M\_X64\_OR\_ARM64) && defined(RECYCLER\_WRITE\_BARRIER)

bool IsWriteBarrierAllowed()

{

return isWriteBarrierAllowed;

}

#endif

protected:

#if \_M\_IX86\_OR\_ARM32

static const uint VirtualAllocThreshold = 524288; // 512kb As per spec

#else // \_M\_X64\_OR\_ARM64

static const uint VirtualAllocThreshold = 1048576; // 1MB As per spec : when we cross this threshold of bytes, we should add guard pages

#endif

static const uint maxGuardPages = 15;

static const uint minGuardPages = 1;

SecondaryAllocator\* secondaryAllocator;

char \* address;

PageAllocatorBase<TVirtualAlloc> \* allocator;

size\_t segmentPageCount;

uint trailingGuardPageCount;

uint leadingGuardPageCount;

uint secondaryAllocPageCount;

#if defined(\_M\_X64\_OR\_ARM64) && defined(RECYCLER\_WRITE\_BARRIER)

bool isWriteBarrierAllowed;

#endif

};

/\*

\* Page Segments allows a client to deal with virtual memory on a page level

\* unlike Segment, which gives you access on a segment basis. Pages managed

\* by the page segment are initially in a “free list”, and have the no access

\* bit set on them. When a client wants pages, we get them from the free list

\* and commit them into memory. When the client no longer needs those pages,

\* we simply decommit them- this means that the pages are still reserved for

\* the process but are not a part of its working set and has no physical

\* storage associated with it.

\*/

template<typename TVirtualAlloc>

class PageSegmentBase : public SegmentBase<TVirtualAlloc>

{

public:

PageSegmentBase(PageAllocatorBase<TVirtualAlloc> \* allocator, bool external);

// Maximum possible size of a PageSegment; may be smaller.

static const uint MaxDataPageCount = 256; // 1 MB

static const uint MaxGuardPageCount = 16;

static const uint MaxPageCount = MaxDataPageCount + MaxGuardPageCount; // 272 Pages

typedef BVStatic<MaxPageCount> PageBitVector;

uint GetAvailablePageCount() const

{

size\_t availablePageCount = SegmentBase::GetAvailablePageCount();

Assert(availablePageCount < MAXUINT32);

return static\_cast<uint>(availablePageCount);

}

void Prime();

#ifdef PAGEALLOCATOR\_PROTECT\_FREEPAGE

bool Initialize(DWORD allocFlags, bool excludeGuardPages);

#endif

uint GetFreePageCount() const { return freePageCount; }

uint GetDecommitPageCount() const { return decommitPageCount; }

static bool IsAllocationPageAligned(\_\_in char\* address, size\_t pageCount, PageHeapMode pageHeapFlags);

template <typename T, bool notPageAligned>

char \* AllocDecommitPages(uint pageCount, T freePages, T decommitPages, PageHeapMode pageHeapFlags);

template <bool notPageAligned>

char \* AllocPages(uint pageCount, PageHeapMode pageHeapFlags);

void ReleasePages(\_\_in void \* address, uint pageCount);

template <bool onlyUpdateState>

void DecommitPages(\_\_in void \* address, uint pageCount);

uint GetCountOfFreePages() const;

uint GetNextBitInFreePagesBitVector(uint index) const;

BOOLEAN TestRangeInFreePagesBitVector(uint index, uint pageCount) const;

BOOLEAN TestInFreePagesBitVector(uint index) const;

void ClearAllInFreePagesBitVector();

void ClearRangeInFreePagesBitVector(uint index, uint pageCount);

void SetRangeInFreePagesBitVector(uint index, uint pageCount);

void ClearBitInFreePagesBitVector(uint index);

uint GetCountOfDecommitPages() const;

BOOLEAN TestInDecommitPagesBitVector(uint index) const;

BOOLEAN TestRangeInDecommitPagesBitVector(uint index, uint pageCount) const;

void SetRangeInDecommitPagesBitVector(uint index, uint pageCount);

void SetBitInDecommitPagesBitVector(uint index);

void ClearRangeInDecommitPagesBitVector(uint index, uint pageCount);

template <bool notPageAligned>

char \* DoAllocDecommitPages(uint pageCount, PageHeapMode pageHeapFlags);

uint GetMaxPageCount();

size\_t DecommitFreePages(size\_t pageToDecommit);

bool IsEmpty() const

{

return this->freePageCount == this->GetAvailablePageCount();

}

//

// If a segment has decommitted pages - then it's not considered full as allocations can take place from it

// However, if secondary allocations cannot be made from it - it's considered full nonetheless

//

bool IsFull() const

{

return (this->freePageCount == 0 && !ShouldBeInDecommittedList()) ||

(this->secondaryAllocator != nullptr && !this->secondaryAllocator->CanAllocate());

}

bool IsAllDecommitted() const

{

return this->GetAvailablePageCount() == this->decommitPageCount;

}

bool ShouldBeInDecommittedList() const

{

return this->decommitPageCount != 0;

}

bool IsFreeOrDecommitted(void\* address, uint pageCount) const

{

Assert(IsInSegment(address));

uint base = GetBitRangeBase(address);

return this->TestRangeInDecommitPagesBitVector(base, pageCount) || this->TestRangeInFreePagesBitVector(base, pageCount);

}

bool IsFreeOrDecommitted(void\* address) const

{

Assert(IsInSegment(address));

uint base = GetBitRangeBase(address);

return this->TestInDecommitPagesBitVector(base) || this->TestInFreePagesBitVector(base);

}

PageBitVector GetUnAllocatedPages() const

{

PageBitVector unallocPages = freePages;

unallocPages.Or(&decommitPages);

return unallocPages;

}

void ChangeSegmentProtection(DWORD protectFlags, DWORD expectedOldProtectFlags);

#if DBG

bool IsPageSegment() const override

{

return true;

}

#endif

//---------- Private members ---------------/

private:

uint GetBitRangeBase(void\* address) const

{

uint base = ((uint)(((char \*)address) - this->address)) / AutoSystemInfo::PageSize;

return base;

}

PageBitVector freePages;

PageBitVector decommitPages;

uint freePageCount;

uint decommitPageCount;

};

template<typename TVirtualAlloc = VirtualAllocWrapper>

class HeapPageAllocator;

/\*

\* A Page Allocation is an allocation made by a page allocator

\* This has a base address, and tracks the number of pages that

\* were allocated from the segment

\*/

class PageAllocation

{

public:

char \* GetAddress() const { return ((char \*)this) + sizeof(PageAllocation); }

size\_t GetSize() const { return pageCount \* AutoSystemInfo::PageSize - sizeof(PageAllocation); }

size\_t GetPageCount() const { return pageCount; }

void\* GetSegment() const { return segment; }

private:

size\_t pageCount;

void \* segment;

friend class PageAllocatorBase<VirtualAllocWrapper>;

friend class PageAllocatorBase<PreReservedVirtualAllocWrapper>;

friend class HeapPageAllocator<>;

};

/\*

\* This allocator is responsible for allocating and freeing pages. It does

\* so by virtue of allocating segments for groups of pages, and then handing

\* out memory in these segments. It's also responsible for free segments.

\* This class also controls the idle decommit thread, which decommits pages

\* when they're no longer needed

\*/

template<typename TVirtualAlloc>

class PageAllocatorBase

{

friend class CodeGenNumberThreadAllocator;

// Allowing recycler to report external memory allocation.

friend class Recycler;

public:

static uint const DefaultMaxFreePageCount = 0x400; // 4 MB

static uint const DefaultLowMaxFreePageCount = 0x100; // 1 MB for low-memory process

static uint const MinPartialDecommitFreePageCount = 0x1000; // 16 MB

static uint const DefaultMaxAllocPageCount = 32; // 128K

static uint const DefaultSecondaryAllocPageCount = 0;

static size\_t GetProcessUsedBytes();

static size\_t GetAndResetMaxUsedBytes();

virtual BOOL ProtectPages(\_\_in char\* address, size\_t pageCount, \_\_in void\* segment, DWORD dwVirtualProtectFlags, DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag)

{

Assert(false);

return false;

}

virtual bool AllocSecondary(void\* segment, ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation)

{

Assert(false);

return false;

}

virtual void ReleaseSecondary(const SecondaryAllocation& allocation, void\* segment)

{

Assert(false);

}

virtual void DecommitPages(\_\_in char\* address, int pageCount)

{

Assert(false);

}

virtual void TrackDecommitedPages(void \* address, uint pageCount, \_\_in void\* segment)

{

Assert(false);

}

struct BackgroundPageQueue

{

BackgroundPageQueue();

SLIST\_HEADER freePageList;

CriticalSection backgroundPageQueueCriticalSection;

#if DBG

bool isZeroPageQueue;

#endif

};

struct ZeroPageQueue : BackgroundPageQueue

{

ZeroPageQueue();

SLIST\_HEADER pendingZeroPageList;

};

PageAllocatorBase(AllocationPolicyManager \* policyManager,

#ifndef JD\_PRIVATE

Js::ConfigFlagsTable& flags = Js::Configuration::Global.flags,

#endif

PageAllocatorType type = PageAllocatorType\_Max,

uint maxFreePageCount = DefaultMaxFreePageCount,

bool zeroPages = false,

BackgroundPageQueue \* backgroundPageQueue = nullptr,

uint maxAllocPageCount = DefaultMaxAllocPageCount,

uint secondaryAllocPageCount = DefaultSecondaryAllocPageCount,

bool stopAllocationOnOutOfMemory = false,

bool excludeGuardPages = false);

virtual ~PageAllocatorBase();

bool IsClosed() const { return isClosed; }

void Close() { Assert(!isClosed); isClosed = true; }

AllocationPolicyManager \* GetAllocationPolicyManager() { return policyManager; }

uint GetMaxAllocPageCount();

//VirtualAllocator APIs

TVirtualAlloc \* GetVirtualAllocator() { return virtualAllocator; }

void SetVirtualAllocator(TVirtualAlloc \* virtualAllocator)

{

Assert(virtualAllocator != nullptr);

PVOID oldVirtualAllocator = InterlockedCompareExchangePointer((PVOID\*) &(this->virtualAllocator), virtualAllocator, NULL);

AssertMsg(oldVirtualAllocator == nullptr || oldVirtualAllocator == (PVOID)virtualAllocator, "Trying to set a new value for VirtualAllocWrapper ? - INVALID");

}

bool IsPreReservedPageAllocator() { return virtualAllocator != nullptr; }

PageAllocation \* AllocPagesForBytes(size\_t requestedBytes);

PageAllocation \* AllocAllocation(size\_t pageCount);

void ReleaseAllocation(PageAllocation \* allocation);

void ReleaseAllocationNoSuspend(PageAllocation \* allocation);

char \* Alloc(size\_t \* pageCount, SegmentBase<TVirtualAlloc> \*\* segment);

void Release(void \* address, size\_t pageCount, void \* segment);

char \* AllocPages(uint pageCount, PageSegmentBase<TVirtualAlloc> \*\* pageSegment);

char \* AllocPagesPageAligned(uint pageCount, PageSegmentBase<TVirtualAlloc> \*\* pageSegment, PageHeapMode pageHeapFlags);

void ReleasePages(\_\_in void \* address, uint pageCount, \_\_in void \* pageSegment);

void BackgroundReleasePages(void \* address, uint pageCount, PageSegmentBase<TVirtualAlloc> \* pageSegment);

// Decommit

void DecommitNow(bool all = true);

void SuspendIdleDecommit();

void ResumeIdleDecommit();

void StartQueueZeroPage();

void StopQueueZeroPage();

void ZeroQueuedPages();

void BackgroundZeroQueuedPages();

void FlushBackgroundPages();

bool DisableAllocationOutOfMemory() const { return disableAllocationOutOfMemory; }

void ResetDisableAllocationOutOfMemory() { disableAllocationOutOfMemory = false; }

#ifdef RECYCLER\_MEMORY\_VERIFY

void EnableVerify() { verifyEnabled = true; }

#endif

#if defined(RECYCLER\_NO\_PAGE\_REUSE) || defined(ARENA\_MEMORY\_VERIFY)

void ReenablePageReuse() { Assert(disablePageReuse); disablePageReuse = false; }

bool DisablePageReuse() { bool wasDisablePageReuse = disablePageReuse; disablePageReuse = true; return wasDisablePageReuse; }

#endif

#if DBG

bool HasZeroQueuedPages() const;

virtual void SetDisableThreadAccessCheck() { disableThreadAccessCheck = true;}

virtual void SetEnableThreadAccessCheck() { disableThreadAccessCheck = false; }

virtual bool IsIdleDecommitPageAllocator() const { return false; }

virtual bool HasMultiThreadAccess() const { return false; }

bool ValidThreadAccess()

{

DWORD currentThreadId = ::GetCurrentThreadId();

return disableThreadAccessCheck ||

(this->concurrentThreadId == -1 && this->threadContextHandle == NULL) || // JIT thread after close

(this->concurrentThreadId != -1 && this->concurrentThreadId == currentThreadId) ||

this->threadContextHandle == GetCurrentThreadContextId();

}

virtual void UpdateThreadContextHandle(ThreadContextId updatedThreadContextHandle) { threadContextHandle = updatedThreadContextHandle; }

void SetConcurrentThreadId(DWORD threadId) { this->concurrentThreadId = threadId; }

void ClearConcurrentThreadId() { this->concurrentThreadId = (DWORD)-1; }

DWORD GetConcurrentThreadId() { return this->concurrentThreadId; }

DWORD HasConcurrentThreadId() { return this->concurrentThreadId != -1; }

#endif

#if DBG\_DUMP

wchar\_t const \* debugName;

#endif

protected:

SegmentBase<TVirtualAlloc> \* AllocSegment(size\_t pageCount);

void ReleaseSegment(SegmentBase<TVirtualAlloc> \* segment);

template <bool doPageAlign>

char \* AllocInternal(size\_t \* pageCount, SegmentBase<TVirtualAlloc> \*\* segment);

template <bool notPageAligned>

char \* SnailAllocPages(uint pageCount, PageSegmentBase<TVirtualAlloc> \*\* pageSegment, PageHeapMode pageHeapFlags);

void OnAllocFromNewSegment(uint pageCount, \_\_in void\* pages, SegmentBase<TVirtualAlloc>\* segment);

template <bool notPageAligned>

char \* TryAllocFreePages(uint pageCount, PageSegmentBase<TVirtualAlloc> \*\* pageSegment, PageHeapMode pageHeapFlags);

template <bool notPageAligned>

char \* TryAllocDecommitedPages(uint pageCount, PageSegmentBase<TVirtualAlloc> \*\* pageSegment, PageHeapMode pageHeapFlags);

DListBase<PageSegmentBase<TVirtualAlloc>> \* GetSegmentList(PageSegmentBase<TVirtualAlloc> \* segment);

void TransferSegment(PageSegmentBase<TVirtualAlloc> \* segment, DListBase<PageSegmentBase<TVirtualAlloc>> \* fromSegmentList);

void FillAllocPages(\_\_in void \* address, uint pageCount);

void FillFreePages(\_\_in void \* address, uint pageCount);

struct FreePageEntry : public SLIST\_ENTRY

{

PageSegmentBase<TVirtualAlloc> \* segment;

uint pageCount;

};

bool IsPageSegment(SegmentBase<TVirtualAlloc>\* segment)

{

return segment->GetAvailablePageCount() <= maxAllocPageCount;

}

#if DBG\_DUMP

virtual void DumpStats() const;

#endif

virtual PageSegmentBase<TVirtualAlloc> \* AddPageSegment(DListBase<PageSegmentBase<TVirtualAlloc>>& segmentList);

static PageSegmentBase<TVirtualAlloc> \* AllocPageSegment(DListBase<PageSegmentBase<TVirtualAlloc>>& segmentList, PageAllocatorBase<TVirtualAlloc> \* pageAllocator, bool external);

// Zero Pages

void AddPageToZeroQueue(\_\_in void \* address, uint pageCount, \_\_in PageSegmentBase<TVirtualAlloc> \* pageSegment);

bool HasZeroPageQueue() const;

bool ZeroPages() const { return zeroPages; }

bool QueueZeroPages() const { return queueZeroPages; }

FreePageEntry \* PopPendingZeroPage();

#if DBG

void Check();

bool disableThreadAccessCheck;

#endif

protected:

// Data

DListBase<PageSegmentBase<TVirtualAlloc>> segments;

DListBase<PageSegmentBase<TVirtualAlloc>> fullSegments;

DListBase<PageSegmentBase<TVirtualAlloc>> emptySegments;

DListBase<PageSegmentBase<TVirtualAlloc>> decommitSegments;

DListBase<SegmentBase<TVirtualAlloc>> largeSegments;

uint maxAllocPageCount;

DWORD allocFlags;

uint maxFreePageCount;

size\_t freePageCount;

uint secondaryAllocPageCount;

bool isClosed;

bool stopAllocationOnOutOfMemory;

bool disableAllocationOutOfMemory;

bool excludeGuardPages;

AllocationPolicyManager \* policyManager;

TVirtualAlloc \* virtualAllocator;

#ifndef JD\_PRIVATE

Js::ConfigFlagsTable& pageAllocatorFlagTable;

#endif

// zero pages

BackgroundPageQueue \* backgroundPageQueue;

bool zeroPages;

bool queueZeroPages;

bool hasZeroQueuedPages;

// Idle Decommit

bool isUsed;

size\_t minFreePageCount;

uint idleDecommitEnterCount;

void UpdateMinFreePageCount();

void ResetMinFreePageCount();

void ClearMinFreePageCount();

void AddFreePageCount(uint pageCount);

static uint GetFreePageLimit() { return 0; }

#if DBG

size\_t debugMinFreePageCount;

ThreadContextId threadContextHandle;

DWORD concurrentThreadId;

#endif

#if DBG\_DUMP

size\_t decommitPageCount;

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

bool verifyEnabled;

#endif

#if defined(RECYCLER\_NO\_PAGE\_REUSE) || defined(ARENA\_MEMORY\_VERIFY)

bool disablePageReuse;

#endif

friend class SegmentBase<TVirtualAlloc>;

friend class PageSegmentBase<TVirtualAlloc>;

friend class IdleDecommit;

protected:

virtual bool CreateSecondaryAllocator(SegmentBase<TVirtualAlloc>\* segment, SecondaryAllocator\*\* allocator)

{

\*allocator = nullptr;

return true;

}

bool IsAddressInSegment(\_\_in void\* address, const PageSegmentBase<TVirtualAlloc>& segment);

bool IsAddressInSegment(\_\_in void\* address, const SegmentBase<TVirtualAlloc>& segment);

private:

uint GetSecondaryAllocPageCount() const { return this->secondaryAllocPageCount; }

void IntegrateSegments(DListBase<PageSegmentBase<TVirtualAlloc>>& segmentList, uint segmentCount, size\_t pageCount);

void QueuePages(void \* address, uint pageCount, PageSegmentBase<TVirtualAlloc> \* pageSegment);

template <bool notPageAligned>

char\* AllocPagesInternal(uint pageCount, PageSegmentBase<TVirtualAlloc> \*\* pageSegment, PageHeapMode pageHeapModeFlags = PageHeapMode::PageHeapModeOff);

#ifdef PROFILE\_MEM

PageMemoryData \* memoryData;

#endif

size\_t usedBytes;

PageAllocatorType type;

size\_t reservedBytes;

size\_t committedBytes;

size\_t numberOfSegments;

#ifdef PERF\_COUNTERS

PerfCounter::Counter& GetReservedSizeCounter() const

{

return PerfCounter::PageAllocatorCounterSet::GetReservedSizeCounter(type);

}

PerfCounter::Counter& GetCommittedSizeCounter() const

{

return PerfCounter::PageAllocatorCounterSet::GetCommittedSizeCounter(type);

}

PerfCounter::Counter& GetUsedSizeCounter() const

{

return PerfCounter::PageAllocatorCounterSet::GetUsedSizeCounter(type);

}

PerfCounter::Counter& GetTotalReservedSizeCounter() const

{

return PerfCounter::PageAllocatorCounterSet::GetTotalReservedSizeCounter();

}

PerfCounter::Counter& GetTotalCommittedSizeCounter() const

{

return PerfCounter::PageAllocatorCounterSet::GetTotalCommittedSizeCounter();

}

PerfCounter::Counter& GetTotalUsedSizeCounter() const

{

return PerfCounter::PageAllocatorCounterSet::GetTotalUsedSizeCounter();

}

#endif

void AddReservedBytes(size\_t bytes);

void SubReservedBytes(size\_t bytes);

void AddCommittedBytes(size\_t bytes);

void SubCommittedBytes(size\_t bytes);

void AddUsedBytes(size\_t bytes);

void SubUsedBytes(size\_t bytes);

void AddNumberOfSegments(size\_t segmentCount);

void SubNumberOfSegments(size\_t segmentCount);

bool RequestAlloc(size\_t byteCount)

{

if (disableAllocationOutOfMemory)

{

return false;

}

if (policyManager != nullptr)

{

return policyManager->RequestAlloc(byteCount);

}

return true;

}

void ReportFree(size\_t byteCount)

{

if (policyManager != nullptr)

{

policyManager->ReportFree(byteCount);

}

}

template <typename T>

void ReleaseSegmentList(DListBase<T> \* segmentList);

protected:

// Instrumentation

void LogAllocSegment(SegmentBase<TVirtualAlloc> \* segment);

void LogAllocSegment(uint segmentCount, size\_t pageCount);

void LogFreeSegment(SegmentBase<TVirtualAlloc> \* segment);

void LogFreeDecommittedSegment(SegmentBase<TVirtualAlloc> \* segment);

void LogFreePartiallyDecommitedPageSegment(PageSegmentBase<TVirtualAlloc> \* pageSegment);

void LogAllocPages(size\_t pageCount);

void LogFreePages(size\_t pageCount);

void LogCommitPages(size\_t pageCount);

void LogRecommitPages(size\_t pageCount);

void LogDecommitPages(size\_t pageCount);

void ReportFailure(size\_t byteCount)

{

if (this->stopAllocationOnOutOfMemory)

{

this->disableAllocationOutOfMemory = true;

}

if (policyManager != nullptr)

{

policyManager->ReportFailure(byteCount);

}

}

};

template<typename TVirtualAlloc>

class HeapPageAllocator : public PageAllocatorBase<TVirtualAlloc>

{

public:

HeapPageAllocator(AllocationPolicyManager \* policyManager, bool allocXdata, bool excludeGuardPages);

BOOL ProtectPages(\_\_in char\* address, size\_t pageCount, \_\_in void\* segment, DWORD dwVirtualProtectFlags, DWORD\* dwOldVirtualProtectFlags, DWORD desiredOldProtectFlag);

bool AllocSecondary(void\* segment, ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation);

void ReleaseSecondary(const SecondaryAllocation& allocation, void\* segment);

void TrackDecommitedPages(void \* address, uint pageCount, \_\_in void\* segment);

void DecommitPages(\_\_in char\* address, size\_t pageCount = 1);

// Release pages that has already been decommited

void ReleaseDecommited(void \* address, size\_t pageCount, \_\_in void \* segment);

bool IsAddressFromAllocator(\_\_in void\* address);

char \* InitPageSegment();

PageSegmentBase<TVirtualAlloc> \* AddPageSegment(DListBase<PageSegmentBase<TVirtualAlloc>>& segmentList);

private:

bool allocXdata;

void ReleaseDecommitedSegment(\_\_in SegmentBase<TVirtualAlloc>\* segment);

#if PDATA\_ENABLED

virtual bool CreateSecondaryAllocator(SegmentBase<TVirtualAlloc>\* segment, SecondaryAllocator\*\* allocator) override;

#endif

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#define PAGE\_ALLOCATOR\_TYPE(MACRO) \

MACRO(Thread) \

MACRO(Diag) \

MACRO(CustomHeap) \

MACRO(BGJIT) \

MACRO(GCThread) \

MACRO(Recycler) \

#define DEFINE\_PAGE\_ALLOCATOR\_TYPE\_ENUM(x) PageAllocatorType\_ ## x,

enum PageAllocatorType

{

PAGE\_ALLOCATOR\_TYPE(DEFINE\_PAGE\_ALLOCATOR\_TYPE\_ENUM)

PageAllocatorType\_Max

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef RECYCLER\_PAGE\_HEAP

namespace Memory

{

enum class PageHeapBlockTypeFilter

{

PageHeapBlockTypeFilterNormal = 0x01,

PageHeapBlockTypeFilterLeaf = 0x02,

PageHeapBlockTypeFilterFinalizable = 0x04,

PageHeapBlockTypeFilterNormalBarrier = 0x08,

PageHeapBlockTypeFilterFinalizableBarrier = 0x10,

PageHeapBlockTypeFilterLarge = 0x20,

PageHeapBlockTypeFilterMax = PageHeapBlockTypeFilterLarge,

PageHeapBlockTypeFilterAll = 0xff

};

};

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// PagePool caches freed pages in a pool for reuse, and more importantly,

// defers freeing them until ReleaseFreePages is called.

// This allows us to free the pages when we know it is multi-thread safe to do so,

// e.g. after all parallel marking is completed.

namespace Memory

{

class PagePoolPage

{

private:

PageAllocator \* pageAllocator;

PageSegment \* pageSegment;

bool isReserved;

public:

static PagePoolPage \* New(PageAllocator \* pageAllocator, bool isReserved = false)

{

PageSegment \* pageSegment;

PagePoolPage \* newPage = (PagePoolPage \*)pageAllocator->AllocPages(1, &pageSegment);

if (newPage == nullptr)

{

return nullptr;

}

newPage->pageAllocator = pageAllocator;

newPage->pageSegment = pageSegment;

newPage->isReserved = isReserved;

return newPage;

}

void Free()

{

this->pageAllocator->ReleasePages(this, 1, this->pageSegment);

}

bool IsReserved()

{

return isReserved;

}

};

class PagePool

{

private:

class PagePoolFreePage : public PagePoolPage

{

public:

PagePoolFreePage \* nextFreePage;

};

PageAllocator pageAllocator;

PagePoolFreePage \* freePageList;

// List of pre-allocated pages that are

// freed only when the page pool is destroyed

PagePoolFreePage \* reservedPageList;

public:

PagePool(Js::ConfigFlagsTable& flagsTable) :

pageAllocator(NULL, flagsTable, PageAllocatorType\_GCThread, PageAllocator::DefaultMaxFreePageCount, false, nullptr, PageAllocator::DefaultMaxAllocPageCount, 0, true),

freePageList(nullptr),

reservedPageList(nullptr)

{

}

void ReservePages(uint reservedPageCount)

{

for (uint i = 0; i < reservedPageCount; i++)

{

PagePoolPage\* page = PagePoolPage::New(&pageAllocator, true);

if (page == nullptr)

{

Js::Throw::OutOfMemory();

}

FreeReservedPage(page);

}

}

~PagePool()

{

Assert(freePageList == nullptr);

if (reservedPageList != nullptr)

{

while (reservedPageList != nullptr)

{

PagePoolFreePage \* page = reservedPageList;

Assert(page->IsReserved());

reservedPageList = reservedPageList->nextFreePage;

page->Free();

}

}

}

PageAllocator \* GetPageAllocator() { return &this->pageAllocator; }

PagePoolPage \* GetPage(bool useReservedPages = false)

{

if (freePageList != nullptr)

{

PagePoolPage \* page = freePageList;

freePageList = freePageList->nextFreePage;

Assert(!page->IsReserved());

return page;

}

if (useReservedPages && reservedPageList != nullptr)

{

PagePoolPage \* page = reservedPageList;

reservedPageList = reservedPageList->nextFreePage;

Assert(page->IsReserved());

return page;

}

return PagePoolPage::New(&pageAllocator);

}

void FreePage(PagePoolPage \* page)

{

PagePoolFreePage \* freePage = (PagePoolFreePage \*)page;

if (freePage->IsReserved())

{

return FreeReservedPage(page);

}

freePage->nextFreePage = freePageList;

freePageList = freePage;

}

void ReleaseFreePages()

{

while (freePageList != nullptr)

{

PagePoolFreePage \* page = freePageList;

Assert(!page->IsReserved());

freePageList = freePageList->nextFreePage;

page->Free();

}

}

void Decommit()

{

pageAllocator.DecommitNow();

}

#if DBG

bool IsEmpty() const { return (freePageList == nullptr); }

#endif

private:

void FreeReservedPage(PagePoolPage \* page)

{

Assert(page->IsReserved());

PagePoolFreePage \* freePage = (PagePoolFreePage \*)page;

freePage->nextFreePage = reservedPageList;

reservedPageList = freePage;

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef CONCURRENT\_GC\_ENABLED

#include <process.h>

#endif

#ifdef \_M\_AMD64

#include "amd64.h"

#endif

#ifdef \_M\_ARM

#include "arm.h"

#endif

#ifdef \_M\_ARM64

#include "arm64.h"

#endif

#include "core\BinaryFeatureControl.h"

#include "Common\ThreadService.h"

#include "Memory\AutoAllocatorObjectPtr.h"

DEFINE\_RECYCLER\_TRACKER\_PERF\_COUNTER(RecyclerWeakReferenceBase);

#ifdef PROFILE\_RECYCLER\_ALLOC

struct UnallocatedPortionOfBumpAllocatedBlock

{

};

struct ExplicitFreeListedObject

{

};

Recycler::TrackerData Recycler::TrackerData::EmptyData(&typeid(UnallocatedPortionOfBumpAllocatedBlock), false);

Recycler::TrackerData Recycler::TrackerData::ExplicitFreeListObjectData(&typeid(ExplicitFreeListedObject), false);

#endif

enum ETWEventGCActivationKind : unsigned

{

ETWEvent\_GarbageCollect = 0, // force in-thread GC

ETWEvent\_ThreadCollect = 1, // thread GC with wait

ETWEvent\_ConcurrentCollect = 2,

ETWEvent\_PartialCollect = 3,

ETWEvent\_ConcurrentMark = 11,

ETWEvent\_ConcurrentRescan = 12,

ETWEvent\_ConcurrentSweep = 13,

ETWEvent\_ConcurrentTransferSwept = 14,

ETWEvent\_ConcurrentFinishMark = 15,

};

DefaultRecyclerCollectionWrapper DefaultRecyclerCollectionWrapper::Instance;

\_\_inline bool

DefaultRecyclerCollectionWrapper::IsCollectionDisabled(Recycler \* recycler)

{

// GC shouldn't be triggered during heap enum, unless we missed a case where it allocate memory (which

// shouldn't happen during heap enum) or for the case we explicitly allow allocation

// REVIEW: isHeapEnumInProgress should have been a collection state and checked before to avoid a check here.

// Collection will be disabled in VarDispEx because it could be called from projection re-entrance as ASTA allows

// QI/AddRef/Release to come back.

bool collectionDisabled = recycler->IsCollectionDisabled();

#if DBG

if (collectionDisabled)

{

// disabled collection should only happen if we allowed allocation during heap enum

if (recycler->IsHeapEnumInProgress())

{

Assert(recycler->AllowAllocationDuringHeapEnum());

}

else

{

#ifdef ENABLE\_PROJECTION

Assert(recycler->IsInRefCountTrackingForProjection());

#else

Assert(false);

#endif

}

}

#endif

return collectionDisabled;

}

BOOL DefaultRecyclerCollectionWrapper::ExecuteRecyclerCollectionFunction(Recycler \* recycler, CollectionFunction function, CollectionFlags flags)

{

if (IsCollectionDisabled(recycler))

{

return FALSE;

}

BOOL ret = FALSE;

BEGIN\_NO\_EXCEPTION

{

ret = (recycler->\*(function))(flags);

}

END\_NO\_EXCEPTION;

return ret;

}

void

DefaultRecyclerCollectionWrapper::DisposeObjects(Recycler \* recycler)

{

if (IsCollectionDisabled(recycler))

{

return;

}

BEGIN\_NO\_EXCEPTION

{

recycler->DisposeObjects();

}

END\_NO\_EXCEPTION;

}

static void\* GetStackBase();

template \_\_forceinline char \* Recycler::AllocWithAttributesInlined<NoBit, false>(size\_t size);

template \_\_forceinline char\* Recycler::RealAlloc<NoBit, false>(HeapInfo\* heap, size\_t size);

template \_\_forceinline \_Ret\_notnull\_ void \* \_\_cdecl operator new<Recycler>(size\_t byteSize, Recycler \* alloc, char \* (Recycler::\*AllocFunc)(size\_t));

Recycler::Recycler(AllocationPolicyManager \* policyManager, IdleDecommitPageAllocator \* pageAllocator, void (\*outOfMemoryFunc)(), Js::ConfigFlagsTable& configFlagsTable) :

collectionState(CollectionStateNotCollecting),

recyclerFlagsTable(configFlagsTable),

recyclerPageAllocator(this, policyManager, configFlagsTable, RecyclerHeuristic::Instance.DefaultMaxFreePageCount, RecyclerHeuristic::Instance.DefaultMaxAllocPageCount),

recyclerLargeBlockPageAllocator(this, policyManager, configFlagsTable, RecyclerHeuristic::Instance.DefaultMaxFreePageCount),

threadService(nullptr),

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator(this, policyManager, configFlagsTable, RecyclerHeuristic::Instance.DefaultMaxFreePageCount),

#endif

threadPageAllocator(pageAllocator),

markPagePool(configFlagsTable),

parallelMarkPagePool1(configFlagsTable),

parallelMarkPagePool2(configFlagsTable),

parallelMarkPagePool3(configFlagsTable),

markContext(this, &this->markPagePool),

parallelMarkContext1(this, &this->parallelMarkPagePool1),

parallelMarkContext2(this, &this->parallelMarkPagePool2),

parallelMarkContext3(this, &this->parallelMarkPagePool3),

clientTrackedObjectAllocator(L"CTO-List", GetPageAllocator(), Js::Throw::OutOfMemory),

outOfMemoryFunc(outOfMemoryFunc),

#ifdef RECYCLER\_TEST\_SUPPORT

checkFn(NULL),

#endif

externalRootMarker(NULL),

externalRootMarkerContext(NULL),

recyclerSweep(nullptr),

inEndMarkOnLowMemory(false),

enableScanInteriorPointers(CUSTOM\_CONFIG\_FLAG(configFlagsTable, RecyclerForceMarkInterior)),

enableScanImplicitRoots(false),

disableCollectOnAllocationHeuristics(false),

#ifdef CONCURRENT\_GC\_ENABLED

backgroundFinishMarkCount(0),

hasPendingUnpinnedObject(false),

hasPendingConcurrentFindRoot(false),

queueTrackedObject(false),

enableConcurrentMark(false), // Default to non-concurrent

enableParallelMark(false),

enableConcurrentSweep(false),

concurrentThread(NULL),

concurrentWorkReadyEvent(NULL),

concurrentWorkDoneEvent(NULL),

mainThreadHandle(NULL),

parallelThread1(this, &Recycler::ParallelWorkFunc<0>),

parallelThread2(this, &Recycler::ParallelWorkFunc<1>),

priorityBoost(false),

skipStack(false),

isAborting(false),

#if DBG

concurrentThreadExited(true),

isProcessingTrackedObjects(false),

hasIncompletedDoCollect(false),

isConcurrentGCOnIdle(false),

isFinishGCOnIdle(false),

isExternalStackSkippingGC(false),

isProcessingRescan(false),

#endif

#ifdef IDLE\_DECOMMIT\_ENABLED

concurrentIdleDecommitEvent(nullptr),

#endif

#endif

#ifdef PARTIAL\_GC\_ENABLED

inPartialCollectMode(false),

scanPinnedObjectMap(false),

partialUncollectedAllocBytes(0),

uncollectedNewPageCountPartialCollect((size\_t)-1),

partialConcurrentNextCollection(false),

#ifdef RECYCLER\_STRESS

forcePartialScanStack(false),

#endif

#endif

#if defined(RECYCLER\_DUMP\_OBJECT\_GRAPH) || defined(LEAK\_REPORT) || defined(CHECK\_MEMORY\_LEAK)

isPrimaryMarkContextInitialized(false),

#endif

allowDispose(false),

hasDisposableObject(false),

tickCountNextDispose(0),

hasPendingTransferDisposedObjects(false),

transientPinnedObject(nullptr),

pinnedObjectMap(1024, HeapAllocator::GetNoMemProtectInstance()),

weakReferenceMap(1024, HeapAllocator::GetNoMemProtectInstance()),

weakReferenceCleanupId(0),

collectionWrapper(&DefaultRecyclerCollectionWrapper::Instance),

isScriptActive(false),

isInScript(false),

isShuttingDown(false),

inExhaustiveCollection(false),

hasExhaustiveCandidate(false),

inDecommitNowCollection(false),

inCacheCleanupCollection(false),

hasPendingDeleteGuestArena(false),

needOOMRescan(false),

hasBackgroundFinishPartial(false),

decommitOnFinish(false)

#ifdef PROFILE\_EXEC

, profiler(nullptr)

, backgroundProfiler(nullptr)

, backgroundProfilerPageAllocator(nullptr, configFlagsTable, PageAllocatorType\_GCThread)

, backgroundProfilerArena()

#endif

#ifdef PROFILE\_MEM

, memoryData(nullptr)

#endif

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

, objectGraphDumper(nullptr)

, dumpObjectOnceOnCollect(false)

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

, trackerDictionary(nullptr)

#endif

#ifdef HEAP\_ENUMERATION\_VALIDATION

,pfPostHeapEnumScanCallback(nullptr)

#endif

, telemetryBlock(&localTelemetryBlock)

#ifdef ENABLE\_JS\_ETW

,bulkFreeMemoryWrittenCount(0)

#endif

#ifdef RECYCLER\_PAGE\_HEAP

, isPageHeapEnabled(false)

, capturePageHeapAllocStack(false)

, capturePageHeapFreeStack(false)

#endif

, objectBeforeCollectCallbackMap(nullptr)

, objectBeforeCollectCallbackState(ObjectBeforeCollectCallback\_None)

{

#ifdef RECYCLER\_MARK\_TRACK

this->markMap = NoCheckHeapNew(MarkMap, &NoCheckHeapAllocator::Instance, 163, &markMapCriticalSection);

markContext.SetMarkMap(markMap);

parallelMarkContext1.SetMarkMap(markMap);

parallelMarkContext2.SetMarkMap(markMap);

parallelMarkContext3.SetMarkMap(markMap);

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

verifyPad = GetRecyclerFlagsTable().RecyclerVerifyPadSize;

verifyEnabled = GetRecyclerFlagsTable().IsEnabled(Js::RecyclerVerifyFlag);

if (verifyEnabled)

{

ForRecyclerPageAllocator(EnableVerify());

}

#endif

#ifdef RECYCLER\_NO\_PAGE\_REUSE

if (GetRecyclerFlagsTable().IsEnabled(Js::RecyclerNoPageReuseFlag))

{

ForRecyclerPageAllocator(DisablePageReuse());

}

#endif

this->inDispose = false;

#if DBG

this->heapBlockCount = 0;

this->collectionCount = 0;

this->disableThreadAccessCheck = false;

this->disableConcurentThreadExitedCheck = false;

#endif

#if DBG || defined RECYCLER\_TRACE

this->inResolveExternalWeakReferences = false;

#endif

#if DBG || defined(RECYCLER\_STATS)

isForceSweeping = false;

#endif

#ifdef RECYCLER\_FINALIZE\_CHECK

collectionStats.finalizeCount = 0;

#endif

RecyclerMemoryTracking::ReportRecyclerCreate(this);

#if DBG\_DUMP

forceTraceMark = false;

recyclerPageAllocator.debugName = L"Recycler";

recyclerLargeBlockPageAllocator.debugName = L"RecyclerLargeBlock";

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.debugName = L"RecyclerWithBarrier";

#endif

#endif

isHeapEnumInProgress = false;

isCollectionDisabled = false;

#if DBG

allowAllocationDuringRenentrance = false;

allowAllocationDuringHeapEnum = false;

#ifdef ENABLE\_PROJECTION

isInRefCountTrackingForProjection = false;

#endif

#endif

ScheduleNextCollection();

#if defined(RECYCLER\_DUMP\_OBJECT\_GRAPH) || defined(LEAK\_REPORT) || defined(CHECK\_MEMORY\_LEAK)

this->inDllCanUnloadNow = false;

this->inDetachProcess = false;

#endif

memset(&localTelemetryBlock, 0, sizeof(localTelemetryBlock));

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

// recycler requires at least Recycler::PrimaryMarkStackReservedPageCount to function properly for the main mark context

this->markContext.SetMaxPageCount(max<size\_t>(GetRecyclerFlagsTable().MaxMarkStackPageCount, Recycler::PrimaryMarkStackReservedPageCount));

this->parallelMarkContext1.SetMaxPageCount(GetRecyclerFlagsTable().MaxMarkStackPageCount);

this->parallelMarkContext2.SetMaxPageCount(GetRecyclerFlagsTable().MaxMarkStackPageCount);

this->parallelMarkContext3.SetMaxPageCount(GetRecyclerFlagsTable().MaxMarkStackPageCount);

if (GetRecyclerFlagsTable().IsEnabled(Js::GCMemoryThresholdFlag))

{

// Note, we can't do this in the constructor for RecyclerHeuristic::Instance because it runs before config is processed

RecyclerHeuristic::Instance.ConfigureBaseFactor(GetRecyclerFlagsTable().GCMemoryThreshold);

}

#endif

}

#if DBG

void

Recycler::SetDisableThreadAccessCheck()

{

recyclerPageAllocator.SetDisableThreadAccessCheck();

recyclerLargeBlockPageAllocator.SetDisableThreadAccessCheck();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.SetDisableThreadAccessCheck();

#endif

disableThreadAccessCheck = true;

}

#endif

void

Recycler::SetMemProtectMode()

{

this->enableScanInteriorPointers = true;

this->enableScanImplicitRoots = true;

this->disableCollectOnAllocationHeuristics = true;

#ifdef RECYCLER\_STRESS

this->recyclerStress = GetRecyclerFlagsTable().MemProtectHeapStress;

#ifdef CONCURRENT\_GC\_ENABLED

this->recyclerBackgroundStress = GetRecyclerFlagsTable().MemProtectHeapBackgroundStress;

this->recyclerConcurrentStress = GetRecyclerFlagsTable().MemProtectHeapConcurrentStress;

this->recyclerConcurrentRepeatStress = GetRecyclerFlagsTable().MemProtectHeapConcurrentRepeatStress;

#endif

#ifdef PARTIAL\_GC\_ENABLED

this->recyclerPartialStress = GetRecyclerFlagsTable().MemProtectHeapPartialStress;

#endif

#endif

}

void

Recycler::LogMemProtectHeapSize(bool fromGC)

{

Assert(IsMemProtectMode());

#ifdef ENABLE\_JS\_ETW

if (IS\_JS\_ETW(EventEnabledMEMPROTECT\_GC\_HEAP\_SIZE()))

{

IdleDecommitPageAllocator\* recyclerPageAllocator = GetRecyclerPageAllocator();

IdleDecommitPageAllocator\* recyclerLeafPageAllocator = GetRecyclerLeafPageAllocator();

IdleDecommitPageAllocator\* recyclerLargeBlockPageAllocator = GetRecyclerLargeBlockPageAllocator();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

IdleDecommitPageAllocator\* recyclerWithBarrierPageAllocator = GetRecyclerWithBarrierPageAllocator();

#endif

size\_t usedBytes = (recyclerPageAllocator->usedBytes + recyclerLeafPageAllocator->usedBytes +

recyclerLargeBlockPageAllocator->usedBytes);

size\_t reservedBytes = (recyclerPageAllocator->reservedBytes + recyclerLeafPageAllocator->reservedBytes +

recyclerLargeBlockPageAllocator->reservedBytes);

size\_t committedBytes = (recyclerPageAllocator->committedBytes + recyclerLeafPageAllocator->committedBytes +

recyclerLargeBlockPageAllocator->committedBytes);

size\_t numberOfSegments = (recyclerPageAllocator->numberOfSegments +

recyclerLeafPageAllocator->numberOfSegments +

recyclerLargeBlockPageAllocator->numberOfSegments);

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

usedBytes += recyclerWithBarrierPageAllocator->usedBytes;

reservedBytes += recyclerWithBarrierPageAllocator->reservedBytes;

committedBytes += recyclerWithBarrierPageAllocator->committedBytes;

numberOfSegments += recyclerWithBarrierPageAllocator->numberOfSegments;

#endif

JS\_ETW(EventWriteMEMPROTECT\_GC\_HEAP\_SIZE(this, usedBytes, reservedBytes, committedBytes, numberOfSegments, fromGC));

}

#endif

}

#if DBG

void

Recycler::SetDisableConcurentThreadExitedCheck()

{

disableConcurentThreadExitedCheck = true;

#ifdef RECYCLER\_STRESS

this->recyclerStress = false;

#ifdef CONCURRENT\_GC\_ENABLED

this->recyclerBackgroundStress = false;

this->recyclerConcurrentStress = false;

this->recyclerConcurrentRepeatStress = false;

#endif

#ifdef PARTIAL\_GC\_ENABLED

this->recyclerPartialStress = false;

#endif

#endif

}

#endif

#if DBG

void

Recycler::ResetThreadId()

{

// Transfer all the page allocator to the current thread id

ForRecyclerPageAllocator(ClearConcurrentThreadId());

#ifdef CONCURRENT\_GC\_ENABLED

if (this->IsConcurrentEnabled())

{

markContext.GetPageAllocator()->ClearConcurrentThreadId();

}

#endif

#if defined(DBG) && defined(PROFILE\_EXEC)

this->backgroundProfilerPageAllocator.ClearConcurrentThreadId();

#endif

}

#endif

Recycler::~Recycler()

{

Assert(!this->isAborting);

// Stop any further collection

this->isShuttingDown = true;

#if DBG

this->ResetThreadId();

#endif

#ifdef ENABLE\_JS\_ETW

FlushFreeRecord();

#endif

ClearObjectBeforeCollectCallbacks();

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

if (GetRecyclerFlagsTable().DumpObjectGraphOnExit)

{

// Always skip stack here, as we may be running the dtor on another thread.

RecyclerObjectGraphDumper::Param param = { 0 };

param.skipStack = true;

this->DumpObjectGraph(&param);

}

#endif

AUTO\_LEAK\_REPORT\_SECTION(this->GetRecyclerFlagsTable(), L"Recycler (%p): %s", this, this->IsInDllCanUnloadNow()? L"DllCanUnloadNow" :

this->IsInDetachProcess()? L"DetachProcess" : L"Destructor");

#ifdef LEAK\_REPORT

ReportLeaks();

#endif

#ifdef CHECK\_MEMORY\_LEAK

CheckLeaks(this->IsInDllCanUnloadNow()? L"DllCanUnloadNow" : this->IsInDetachProcess()? L"DetachProcess" : L"Destructor");

#endif

AUTO\_LEAK\_REPORT\_SECTION(this->GetRecyclerFlagsTable(), L"Skipped finalizers");

#ifdef CONCURRENT\_GC\_ENABLED

Assert(concurrentThread == nullptr);

// We only sometime clean up the state after abort concurrent to not collection

// Still need to delete heap block that is held by the recyclerSweep

if (recyclerSweep != nullptr)

{

recyclerSweep->ShutdownCleanup();

recyclerSweep = nullptr;

}

if (mainThreadHandle != nullptr)

{

CloseHandle(mainThreadHandle);

}

#endif

recyclerPageAllocator.Close();

recyclerLargeBlockPageAllocator.Close();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.Close();

#endif

markContext.Release();

parallelMarkContext1.Release();

parallelMarkContext2.Release();

parallelMarkContext3.Release();

// Clean up the weak reference map so that

// objects being finalized can safely refer to weak references

// (this could otherwise become a problem for weak references held

// to large objects since their block would be destroyed before

// the finalizer was run)

// When the recycler is shutting down, all objects are going to be reclaimed

// so null out the weak references so that anyone relying on weak

// references simply thinks the object has been reclaimed

weakReferenceMap.Map([](RecyclerWeakReferenceBase \* weakRef) -> bool

{

weakRef->strongRef = nullptr;

// Put in a dummy heap block so that we can still do the isPendingConcurrentSweep check first.

weakRef->strongRefHeapBlock = &CollectedRecyclerWeakRefHeapBlock::Instance;

// Remove

return false;

});

#ifdef PARTIAL\_GC\_ENABLED

clientTrackedObjectList.Clear(&this->clientTrackedObjectAllocator);

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

if (trackerDictionary != nullptr)

{

this->trackerDictionary->Map([](type\_info const \*, TrackerItem \* item)

{

NoCheckHeapDelete(item);

});

NoCheckHeapDelete(this->trackerDictionary);

this->trackerDictionary = nullptr;

::DeleteCriticalSection(&trackerCriticalSection);

}

#endif

#ifdef RECYCLER\_MARK\_TRACK

NoCheckHeapDelete(this->markMap);

this->markMap = nullptr;

#endif

#if DBG

// Disable idle decommit asserts

ForRecyclerPageAllocator(ShutdownIdleDecommit());

#endif

Assert(this->collectionState == CollectionStateExit || this->collectionState == CollectionStateNotCollecting);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(this->disableConcurentThreadExitedCheck || this->concurrentThreadExited == true);

#endif

}

void

Recycler::SetIsThreadBound()

{

Assert(mainThreadHandle == nullptr);

::DuplicateHandle(::GetCurrentProcess(), ::GetCurrentThread(), ::GetCurrentProcess(), &mainThreadHandle,

0, FALSE, DUPLICATE\_SAME\_ACCESS);

stackBase = GetStackBase();

}

void

Recycler::RootAddRef(void\* obj, uint \*count)

{

Assert(this->IsValidObject(obj));

if (transientPinnedObject)

{

PinRecord& refCount = pinnedObjectMap.GetReference(transientPinnedObject);

++refCount;

if (refCount == 1)

{

this->scanPinnedObjectMap = true;

RECYCLER\_PERF\_COUNTER\_INC(PinnedObject);

}

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

if (GetRecyclerFlagsTable().LeakStackTrace)

{

StackBackTraceNode::Prepend(&NoCheckHeapAllocator::Instance, refCount.stackBackTraces,

transientPinnedObjectStackBackTrace);

}

#endif

}

if (count != nullptr)

{

PinRecord\* refCount = pinnedObjectMap.TryGetReference(obj);

\*count = (refCount != nullptr) ? (\*refCount + 1) : 1;

}

transientPinnedObject = obj;

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

if (GetRecyclerFlagsTable().LeakStackTrace)

{

transientPinnedObjectStackBackTrace = StackBackTrace::Capture(&NoCheckHeapAllocator::Instance);

}

#endif

}

void

Recycler::RootRelease(void\* obj, uint \*count)

{

Assert(this->IsValidObject(obj));

if (transientPinnedObject == obj)

{

transientPinnedObject = nullptr;

if (count != nullptr)

{

PinRecord \*refCount = pinnedObjectMap.TryGetReference(obj);

\*count = (refCount != nullptr) ? \*refCount : 0;

}

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

if (GetRecyclerFlagsTable().LeakStackTrace)

{

transientPinnedObjectStackBackTrace->Delete(&NoCheckHeapAllocator::Instance);

}

#endif

}

else

{

PinRecord \*refCount = pinnedObjectMap.TryGetReference(obj);

if (refCount == nullptr)

{

if (count != nullptr)

{

\*count = (uint)-1;

}

// REVIEW: throw if not found

Assert(false);

return;

}

uint newRefCount = (--(\*refCount));

if (count != nullptr)

{

\*count = newRefCount;

}

if (newRefCount != 0)

{

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

if (GetRecyclerFlagsTable().LeakStackTrace)

{

StackBackTraceNode::Prepend(&NoCheckHeapAllocator::Instance, refCount->stackBackTraces,

StackBackTrace::Capture(&NoCheckHeapAllocator::Instance));

}

#endif

return;

}

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

StackBackTraceNode::DeleteAll(&NoCheckHeapAllocator::Instance, refCount->stackBackTraces);

refCount->stackBackTraces = nullptr;

#endif

// Don't delete the entry if we are in concurrent find root state

// We will delete it later on in-thread find root

if (this->hasPendingConcurrentFindRoot)

{

this->hasPendingUnpinnedObject = true;

}

else

{

pinnedObjectMap.Remove(obj);

}

RECYCLER\_PERF\_COUNTER\_DEC(PinnedObject);

}

// Not a real collection. This doesn't activate GC.

// This tell the GC that we have an exhaustive candidate, and should trigger

// another GC if there is an exhaustive GC going on.

this->CollectNow<CollectExhaustiveCandidate>();

}

void

Recycler::Initialize(const bool forceInThread, JsUtil::ThreadService \*threadService, const bool deferThreadStartup

#ifdef RECYCLER\_PAGE\_HEAP

, PageHeapMode pageheapmode

, bool captureAllocCallStack

, bool captureFreeCallStack

#endif

)

{

#ifdef PROFILE\_RECYCLER\_ALLOC

this->InitializeProfileAllocTracker();

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

this->disableCollection = CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::RecyclerPhase);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

this->skipStack = false;

#endif

#ifdef PARTIAL\_GC\_ENABLED

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

this->enablePartialCollect = !CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::PartialCollectPhase);

#else

this->enablePartialCollect = true;

#endif

#endif

#ifdef PROFILE\_MEM

this->memoryData = MemoryProfiler::GetRecyclerMemoryData();

#endif

#if DBG || DBG\_DUMP || defined(RECYCLER\_TRACE)

mainThreadId = GetCurrentThreadContextId();

#endif

#ifdef RECYCLER\_TRACE

collectionParam.domCollect = false;

#endif

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(RECYCLER\_MEMORY\_VERIFY) || defined(MEMSPECT\_TRACKING) || defined(ETW\_MEMORY\_TRACKING)

bool dontNeedDetailedTracking = false;

#if defined(PROFILE\_RECYCLER\_ALLOC)

dontNeedDetailedTracking = dontNeedDetailedTracking || this->trackerDictionary == nullptr;

#endif

#if defined(RECYCLER\_MEMORY\_VERIFY)

dontNeedDetailedTracking = dontNeedDetailedTracking || !this->verifyEnabled;

#endif

// If we need detailed tracking we force allocation fast path in the JIT to fail and go to the helper, so there is no

// need for the TrackNativeAllocatedMemoryBlock callback.

if (dontNeedDetailedTracking)

{

autoHeap.Initialize(this, TrackNativeAllocatedMemoryBlock

#ifdef RECYCLER\_PAGE\_HEAP

, pageheapmode

, captureAllocCallStack

, captureFreeCallStack

#endif

);

}

else

{

autoHeap.Initialize(this

#ifdef RECYCLER\_PAGE\_HEAP

, pageheapmode

, captureAllocCallStack

, captureFreeCallStack

#endif

);

}

#else

autoHeap.Initialize(this

#ifdef RECYCLER\_PAGE\_HEAP

, pageheapmode

, captureAllocCallStack

, captureFreeCallStack

#endif

);

#endif

markContext.Init(Recycler::PrimaryMarkStackReservedPageCount);

#if defined(RECYCLER\_DUMP\_OBJECT\_GRAPH) || defined(LEAK\_REPORT) || defined(CHECK\_MEMORY\_LEAK)

isPrimaryMarkContextInitialized = true;

#endif

#ifdef RECYCLER\_PAGE\_HEAP

isPageHeapEnabled = autoHeap.IsPageHeapEnabled();

if (IsPageHeapEnabled())

{

capturePageHeapAllocStack = autoHeap.captureAllocCallStack;

capturePageHeapFreeStack = autoHeap.captureFreeCallStack;

}

#endif

#ifdef RECYCLER\_STRESS

if (GetRecyclerFlagsTable().RecyclerTrackStress)

{

// Disable partial if we are doing track stress, since partial relies on ClientTracked processing

// and track stress doesn't support this.

this->enablePartialCollect = false;

}

this->recyclerStress = GetRecyclerFlagsTable().RecyclerStress;

#ifdef CONCURRENT\_GC\_ENABLED

this->recyclerBackgroundStress = GetRecyclerFlagsTable().RecyclerBackgroundStress;

this->recyclerConcurrentStress = GetRecyclerFlagsTable().RecyclerConcurrentStress;

this->recyclerConcurrentRepeatStress = GetRecyclerFlagsTable().RecyclerConcurrentRepeatStress;

#endif

#ifdef PARTIAL\_GC\_ENABLED

this->recyclerPartialStress = GetRecyclerFlagsTable().RecyclerPartialStress;

#endif

#endif

bool needWriteWatch = false;

#ifdef CONCURRENT\_GC\_ENABLED

// Default to non-concurrent

uint numProcs = (uint)AutoSystemInfo::Data.GetNumberOfPhysicalProcessors();

this->maxParallelism = (numProcs > 4) || CUSTOM\_PHASE\_FORCE1(GetRecyclerFlagsTable(), Js::ParallelMarkPhase) ? 4 : numProcs;

if (forceInThread)

{

// Requested a non-concurrent recycler

this->disableConcurrent = true;

}

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

else if (CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::ConcurrentCollectPhase))

{

// Concurrent collection disabled

this->disableConcurrent = true;

}

else if (CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::ConcurrentMarkPhase) &&

CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::ParallelMarkPhase) &&

CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::ConcurrentSweepPhase))

{

// All concurrent collection phases disabled

this->disableConcurrent = true;

}

#endif

else

{

this->disableConcurrent = false;

if (deferThreadStartup || EnableConcurrent(threadService, false))

{

needWriteWatch = true;

}

}

#endif

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

needWriteWatch = true;

}

#endif

if (needWriteWatch)

{

// need write watch to support concurrent and/or partial collection

recyclerPageAllocator.EnableWriteWatch();

recyclerLargeBlockPageAllocator.EnableWriteWatch();

}

}

#if DBG

BOOL

Recycler::IsFreeObject(void \* candidate)

{

HeapBlock \* heapBlock = this->FindHeapBlock(candidate);

if (heapBlock != NULL)

{

return heapBlock->IsFreeObject(candidate);

}

return false;

}

#endif

BOOL

Recycler::IsValidObject(void\* candidate, size\_t minimumSize)

{

HeapBlock \* heapBlock = this->FindHeapBlock(candidate);

if (heapBlock != NULL)

{

return heapBlock->IsValidObject(candidate) && (minimumSize == 0 || heapBlock->GetObjectSize(candidate) >= minimumSize);

}

return false;

}

void

Recycler::Prime()

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (GetRecyclerFlagsTable().IsEnabled(Js::ForceFragmentAddressSpaceFlag))

{

// Never prime the recycler if we are forced to fragment address space

return;

}

#endif

ForRecyclerPageAllocator(Prime(RecyclerPageAllocator::DefaultPrimePageCount));

}

void

Recycler::AddExternalMemoryUsage(size\_t size)

{

this->autoHeap.uncollectedAllocBytes += size;

this->autoHeap.uncollectedExternalBytes += size;

// Generally normal GC can cleanup the uncollectedAllocBytes. But if external components

// do fast large allocations in a row, normal GC might not kick in. Let's force the GC

// here if we need to collect anyhow.

CollectNow<CollectOnAllocation>();

}

BOOL Recycler::ReportExternalMemoryAllocation(size\_t size)

{

return recyclerPageAllocator.RequestAlloc(size);

}

void Recycler::ReportExternalMemoryFailure(size\_t size)

{

recyclerPageAllocator.ReportFailure(size);

}

void Recycler::ReportExternalMemoryFree(size\_t size)

{

recyclerPageAllocator.ReportFree(size);

}

/\*------------------------------------------------------------------------------------------------

\* Idle Decommit

\*------------------------------------------------------------------------------------------------\*/

void

Recycler::EnterIdleDecommit()

{

ForRecyclerPageAllocator(EnterIdleDecommit());

#ifdef IDLE\_DECOMMIT\_ENABLED

::InterlockedCompareExchange(&needIdleDecommitSignal, IdleDecommitSignal\_None, IdleDecommitSignal\_NeedTimer);

#endif

}

void

Recycler::LeaveIdleDecommit()

{

#ifdef IDLE\_DECOMMIT\_ENABLED

bool allowTimer = (this->concurrentIdleDecommitEvent != nullptr);

IdleDecommitSignal idleDecommitSignalRecycler = recyclerPageAllocator.LeaveIdleDecommit(allowTimer);

IdleDecommitSignal idleDecommitSignalRecyclerLargeBlock = recyclerLargeBlockPageAllocator.LeaveIdleDecommit(allowTimer);

IdleDecommitSignal idleDecommitSignal = max(idleDecommitSignalRecycler, idleDecommitSignalRecyclerLargeBlock);

IdleDecommitSignal idleDecommitSignalThread = threadPageAllocator->LeaveIdleDecommit(allowTimer);

idleDecommitSignal = max(idleDecommitSignal, idleDecommitSignalThread);

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

IdleDecommitSignal idleDecommitSignalRecyclerWithBarrier = recyclerWithBarrierPageAllocator.LeaveIdleDecommit(allowTimer);

idleDecommitSignal = max(idleDecommitSignal, idleDecommitSignalRecyclerWithBarrier);

#endif

if (idleDecommitSignal != IdleDecommitSignal\_None)

{

Assert(allowTimer);

// Reduce the number of times we need to signal the background thread

// by detecting whether the thread is waiting on a time out or not

if (idleDecommitSignal == IdleDecommitSignal\_NeedSignal ||

::InterlockedCompareExchange(&needIdleDecommitSignal, IdleDecommitSignal\_NeedTimer, IdleDecommitSignal\_None) == IdleDecommitSignal\_NeedSignal)

{

#if DBG

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::IdleDecommitPhase))

{

Output::Print(L"Recycler Thread IdleDecommit Need Signal\n");

Output::Flush();

}

#endif

SetEvent(this->concurrentIdleDecommitEvent);

}

}

#else

ForEachRecyclerPageAllocatorIn(this, LeaveIdleDecommit(false));

#endif

}

/\*------------------------------------------------------------------------------------------------

\* Freeing

\*------------------------------------------------------------------------------------------------\*/

bool Recycler::ExplicitFreeLeaf(void\* buffer, size\_t size)

{

return ExplicitFreeInternalWrapper<ObjectInfoBits::LeafBit>(buffer, size);

}

bool Recycler::ExplicitFreeNonLeaf(void\* buffer, size\_t size)

{

return ExplicitFreeInternalWrapper<ObjectInfoBits::NoBit>(buffer, size);

}

size\_t Recycler::GetAllocSize(size\_t size)

{

size\_t allocSize = size;

#ifdef RECYCLER\_MEMORY\_VERIFY

if (this->VerifyEnabled())

{

allocSize += verifyPad + sizeof(size\_t);

Assert(allocSize > size);

}

#endif

return allocSize;

}

template <typename TBlockAttributes>

void Recycler::SetExplicitFreeBitOnSmallBlock(HeapBlock\* heapBlock, size\_t sizeCat, void\* buffer, ObjectInfoBits attributes)

{

Assert(!heapBlock->IsLargeHeapBlock());

Assert(heapBlock->GetObjectSize(buffer) == sizeCat);

SmallHeapBlockT<TBlockAttributes>\* smallBlock = (SmallHeapBlockT<TBlockAttributes>\*)heapBlock;

if ((attributes & ObjectInfoBits::LeafBit) == LeafBit)

{

Assert(smallBlock->IsLeafBlock());

}

else

{

Assert(smallBlock->IsAnyNormalBlock());

}

#ifdef RECYCLER\_MEMORY\_VERIFY

smallBlock->SetExplicitFreeBitForObject(buffer);

#endif

}

template <ObjectInfoBits attributes>

bool Recycler::ExplicitFreeInternalWrapper(void\* buffer, size\_t size)

{

Assert(buffer != nullptr);

Assert(size > 0);

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::ExplicitFreePhase))

{

return false;

}

#endif

size\_t allocSize = GetAllocSize(size);

if (HeapInfo::IsSmallObject(allocSize))

{

return ExplicitFreeInternal<attributes, SmallAllocationBlockAttributes>(buffer, size, HeapInfo::GetAlignedSizeNoCheck(allocSize));

}

if (HeapInfo::IsMediumObject(allocSize))

{

return ExplicitFreeInternal<attributes, MediumAllocationBlockAttributes>(buffer, size, HeapInfo::GetMediumObjectAlignedSizeNoCheck(allocSize));

}

return false;

}

template <ObjectInfoBits attributes, typename TBlockAttributes>

bool Recycler::ExplicitFreeInternal(void\* buffer, size\_t size, size\_t sizeCat)

{

// If the GC is in sweep state while FreeInternal is called, we might be executing a finalizer

// which called Free, which would cause a "sweepable" buffer to be free-listed. Don't allow this.

// Also don't allow freeing while we're shutting down the recycler since finalizers get executed

// at this stage too

if (this->IsSweeping() || this->IsExiting())

{

return false;

}

// We shouldn't be freeing object when we are running GC in thread

Assert(this->IsConcurrentState() || !this->CollectionInProgress() || this->collectionState == CollectionStatePostCollectionCallback);

DebugOnly(RecyclerHeapObjectInfo info);

Assert(this->FindHeapObject(buffer, FindHeapObjectFlags\_NoFreeBitVerify, info));

Assert((info.GetAttributes() & ~ObjectInfoBits::LeafBit) == 0); // Only NoBit or LeafBit

#if DBG || defined(RECYCLER\_MEMORY\_VERIFY) || defined(RECYCLER\_PAGE\_HEAP)

// Either the mainThreadHandle is null (we're not thread bound)

// or we should be calling this function on the main script thread

Assert(this->mainThreadHandle == NULL ||

::GetCurrentThreadId() == ::GetThreadId(this->mainThreadHandle));

HeapBlock\* heapBlock = this->FindHeapBlock(buffer);

Assert(heapBlock != nullptr);

#ifdef RECYCLER\_PAGE\_HEAP

if (this->IsPageHeapEnabled() && this->ShouldCapturePageHeapFreeStack())

{

heapBlock->CapturePageHeapFreeStack();

// Don't do actual explicit free in page heap mode

return false;

}

#endif

SetExplicitFreeBitOnSmallBlock<TBlockAttributes>(heapBlock, sizeCat, buffer, attributes);

#endif

if (TBlockAttributes::IsMediumBlock)

{

autoHeap.FreeMediumObject<attributes>(buffer, sizeCat);

}

else

{

autoHeap.FreeSmallObject<attributes>(buffer, sizeCat);

}

if (size > sizeof(FreeObject) || TBlockAttributes::IsMediumBlock)

{

// Do this on the background somehow?

byte expectedFill = 0;

size\_t fillSize = size - sizeof(FreeObject);

#ifdef RECYCLER\_MEMORY\_VERIFY

if (this->VerifyEnabled())

{

expectedFill = Recycler::VerifyMemFill;

}

#endif

memset(((char\*)buffer) + sizeof(FreeObject), expectedFill, fillSize);

}

#ifdef PROFILE\_RECYCLER\_ALLOC

if (this->trackerDictionary != nullptr)

{

this->SetTrackerData(buffer, &TrackerData::ExplicitFreeListObjectData);

}

#endif

return true;

}

/\*------------------------------------------------------------------------------------------------

\* Allocation

\*------------------------------------------------------------------------------------------------\*/

char \*

Recycler::TryLargeAlloc(HeapInfo \* heap, size\_t size, ObjectInfoBits attributes, bool nothrow)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

Assert(size != 0);

size\_t sizeCat = HeapInfo::GetAlignedSizeNoCheck(size);

if (sizeCat == 0)

{

// overflow scenario

// if onthrow is false, throw out of memory

// otherwise, return null

if (nothrow == false)

{

this->OutOfMemory();

}

return nullptr;

}

char \* memBlock;

if (heap->largeObjectBucket.largeBlockList != nullptr)

{

memBlock = heap->largeObjectBucket.largeBlockList->Alloc(sizeCat, attributes);

if (memBlock != nullptr)

{

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

VerifyZeroFill(memBlock, sizeCat);

#endif

return memBlock;

}

}

// We don't care whether a GC happened here or not, because we are not reusing freed

// large objects. We might try to allocate from existing block if we implement

// large object reuse.

if (!this->disableCollectOnAllocationHeuristics)

{

CollectNow<CollectOnAllocation>();

}

#ifdef RECYCLER\_PAGE\_HEAP

if (IsPageHeapEnabled())

{

if (heap->largeObjectBucket.IsPageHeapEnabled())

{

memBlock = heap->largeObjectBucket.PageHeapAlloc(this, size, (ObjectInfoBits)attributes, autoHeap.pageHeapMode, nothrow);

if (memBlock != nullptr)

{

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

VerifyZeroFill(memBlock, sizeCat);

#endif

return memBlock;

}

}

}

#endif

LargeHeapBlock \* heapBlock = heap->AddLargeHeapBlock(sizeCat);

if (heapBlock == nullptr)

{

return nullptr;

}

memBlock = heapBlock->Alloc(sizeCat, attributes);

Assert(memBlock != nullptr);

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

VerifyZeroFill(memBlock, sizeCat);

#endif

return memBlock;

}

template <bool nothrow>

char\*

Recycler::LargeAlloc(HeapInfo\* heap, size\_t size, ObjectInfoBits attributes)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

char \* addr = TryLargeAlloc(heap, size, attributes, nothrow);

if (addr == nullptr)

{

// Force a collection and try to allocate again.

this->CollectNow<CollectNowForceInThread>();

addr = TryLargeAlloc(heap, size, attributes, nothrow);

if (addr == nullptr)

{

if (nothrow == false)

{

// Still fails, we are out of memory

// Since nothrow is false, it's okay to throw here

this->OutOfMemory();

}

else

{

return nullptr;

}

}

}

autoHeap.uncollectedAllocBytes += size;

return addr;

}

// Explicitly instantiate both versions of LargeAlloc

template char\* Recycler::LargeAlloc<true>(HeapInfo\* heap, size\_t size, ObjectInfoBits attributes);

template char\* Recycler::LargeAlloc<false>(HeapInfo\* heap, size\_t size, ObjectInfoBits attributes);

void

Recycler::OutOfMemory()

{

outOfMemoryFunc();

}

void Recycler::GetNormalHeapBlockAllocatorInfoForNativeAllocation(size\_t allocSize, void\*& allocatorAddress, uint32& endAddressOffset, uint32& freeListOffset)

{

Assert(HeapInfo::IsAlignedSize(allocSize));

Assert(HeapInfo::IsSmallObject(allocSize));

allocatorAddress = GetAddressOfAllocator<NoBit>(allocSize);

endAddressOffset = GetEndAddressOffset<NoBit>(allocSize);

freeListOffset = GetFreeObjectListOffset<NoBit>(allocSize);

if (!AllowNativeCodeBumpAllocation())

{

freeListOffset = endAddressOffset;

}

}

bool Recycler::AllowNativeCodeBumpAllocation()

{

// In debug builds, if we need to track allocation info, we pretend there is no pointer-bump-allocation space

// on this page, so that we always fail the check in native code and go to helper, which does the tracking.

#ifdef PROFILE\_RECYCLER\_ALLOC

if (this->trackerDictionary != nullptr)

{

return false;

}

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

if (this->verifyEnabled)

{

return false;

}

#endif

#ifdef RECYCLER\_PAGE\_HEAP

// Don't allow bump allocation in the JIT when page heap is turned on

if (this->IsPageHeapEnabled())

{

return false;

}

#endif

return true;

}

void Recycler::TrackNativeAllocatedMemoryBlock(Recycler \* recycler, void \* memBlock, size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedSize(sizeCat));

Assert(HeapInfo::IsSmallObject(sizeCat));

#ifdef PROFILE\_RECYCLER\_ALLOC

AssertMsg(!Recycler::DoProfileAllocTracker(), "Why did we register allocation tracking callback if all allocations are forced to slow path?");

#endif

RecyclerMemoryTracking::ReportAllocation(recycler, memBlock, sizeCat);

RECYCLER\_PERF\_COUNTER\_INC(LiveObject);

RECYCLER\_PERF\_COUNTER\_ADD(LiveObjectSize, sizeCat);

RECYCLER\_PERF\_COUNTER\_SUB(FreeObjectSize, sizeCat);

#ifdef RECYCLER\_MEMORY\_VERIFY

AssertMsg(!recycler->VerifyEnabled(), "Why did we register allocation tracking callback if all allocations are forced to slow path?");

#endif

}

/\*------------------------------------------------------------------------------------------------

\* FindRoots

\*------------------------------------------------------------------------------------------------\*/

static void\* GetStackBase()

{

return ((NT\_TIB \*)NtCurrentTeb())->StackBase;

}

#if \_M\_IX86

// REVIEW: For x86, do we care about scanning esp/ebp?

// At GC time, they shouldn't be pointing to GC memory.

#define SAVE\_THREAD\_CONTEXT() \

void\*\* targetBuffer = this->savedThreadContext.GetRegisters(); \

\_\_asm { push eax } \

\_\_asm { mov eax, targetBuffer } \

\_\_asm { mov [eax], esp} \

\_\_asm { mov [eax+0x4], eax} \

\_\_asm { mov [eax+0x8], ebx} \

\_\_asm { mov [eax+0xc], ecx} \

\_\_asm { mov [eax+0x10], edx} \

\_\_asm { mov [eax+0x14], ebp} \

\_\_asm { mov [eax+0x18], esi} \

\_\_asm { mov [eax+0x1c], edi} \

\_\_asm { pop eax }

#elif \_M\_ARM

#define SAVE\_THREAD\_CONTEXT() arm\_SAVE\_REGISTERS(this->savedThreadContext.GetRegisters());

#elif \_M\_ARM64

#define SAVE\_THREAD\_CONTEXT() arm64\_SAVE\_REGISTERS(this->savedThreadContext.GetRegisters());

#elif \_M\_AMD64

#define SAVE\_THREAD\_CONTEXT() amd64\_SAVE\_REGISTERS(this->savedThreadContext.GetRegisters());

#else

#error Unexpected architecture

#endif

size\_t

Recycler::ScanArena(ArenaData \* alloc, bool background)

{

#if DBG\_DUMP

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase)

|| GetRecyclerFlagsTable().Trace.IsEnabled(Js::FindRootPhase))

{

this->forceTraceMark = true;

Output::Print(L"Scanning Guest Arena %p: ", alloc);

}

#endif

size\_t scanRootBytes = 0;

BEGIN\_DUMP\_OBJECT\_ADDRESS(L"Guest Arena", alloc);

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

// The new write watch batching logic broke the write watch handling here.

// For now, just disable write watch for guest arenas.

// Re-enable this in the future.

#if FALSE

// Note, guest arenas are allocated out of the large block page allocator.

bool writeWatch = alloc->GetPageAllocator() == &this->recyclerLargeBlockPageAllocator;

// Only use write watch when we are doing rescan (Partial collect or finish concurrent)

if (writeWatch && this->collectionState == CollectionStateRescanFindRoots)

{

scanRootBytes += TryMarkBigBlockListWithWriteWatch(alloc->GetBigBlocks(background));

scanRootBytes += TryMarkBigBlockListWithWriteWatch(alloc->GetFullBlocks());

}

else

#endif

#endif

{

scanRootBytes += TryMarkBigBlockList(alloc->GetBigBlocks(background));

scanRootBytes += TryMarkBigBlockList(alloc->GetFullBlocks());

}

scanRootBytes += TryMarkArenaMemoryBlockList(alloc->GetMemoryBlocks());

END\_DUMP\_OBJECT(this);

#if DBG\_DUMP

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase)

|| GetRecyclerFlagsTable().Trace.IsEnabled(Js::FindRootPhase))

{

this->forceTraceMark = false;

Output::Print(L"\n");

Output::Flush();

}

#endif

// The arena has been scanned so the full blocks can be rearranged at this point

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (background || !GetRecyclerFlagsTable().RecyclerProtectPagesOnRescan)

#endif

{

alloc->SetLockBlockList(false);

}

return scanRootBytes;

}

#if DBG

bool

Recycler::ExpectStackSkip() const

{

// Okay to skip the stack scan if we're in leak check mode

bool expectStackSkip = false;

#ifdef LEAK\_REPORT

expectStackSkip = expectStackSkip || GetRecyclerFlagsTable().IsEnabled(Js::LeakReportFlag);

#endif

#ifdef CHECK\_MEMORY\_LEAK

expectStackSkip = expectStackSkip || GetRecyclerFlagsTable().CheckMemoryLeak;

#endif

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

expectStackSkip = expectStackSkip || (this->objectGraphDumper != nullptr);

#endif

#if defined(INTERNAL\_MEM\_PROTECT\_HEAP\_ALLOC)

expectStackSkip = expectStackSkip || GetRecyclerFlagsTable().MemProtectHeap;

#endif

return expectStackSkip || isExternalStackSkippingGC;

}

#endif

#pragma warning(push)

#pragma warning(disable:4731) // 'pointer' : frame pointer register 'register' modified by inline assembly code

size\_t

Recycler::ScanStack()

{

#ifdef CONCURRENT\_GC\_ENABLED

if (this->skipStack)

{

#ifdef RECYCLER\_TRACE

CUSTOM\_PHASE\_PRINT\_VERBOSE\_TRACE1(GetRecyclerFlagsTable(), Js::ScanStackPhase, L"[%04X] Skipping the stack scan\n", ::GetCurrentThreadId());

#endif

Assert(this->isFinishGCOnIdle || this->isConcurrentGCOnIdle || this->ExpectStackSkip());

return 0;

}

#endif

#ifdef RECYCLER\_STATS

size\_t lastMarkCount = this->collectionStats.markData.markCount;

#endif

GCETW(GC\_SCANSTACK\_START, (this));

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::ScanStackPhase);

SAVE\_THREAD\_CONTEXT();

void \* stackTop = this->savedThreadContext.GetStackTop();

void \* stackStart = GetStackBase();

Assert(stackStart > stackTop);

size\_t stackScanned = (size\_t)((char \*)stackStart - (char \*)stackTop);

#if DBG\_DUMP

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase)

|| GetRecyclerFlagsTable().Trace.IsEnabled(Js::ScanStackPhase))

{

this->forceTraceMark = true;

Output::Print(L"Scanning Stack %p(%8d): ", stackTop, (char \*)stackStart - (char \*)stackTop);

}

#endif

BEGIN\_DUMP\_OBJECT(this, L"Registers");

ScanMemoryInline(this->savedThreadContext.GetRegisters(), sizeof(void\*) \* SavedRegisterState::NumRegistersToSave);

END\_DUMP\_OBJECT(this);

BEGIN\_DUMP\_OBJECT(this, L"Stack");

ScanMemoryInline((void\*\*) stackTop, stackScanned);

END\_DUMP\_OBJECT(this);

#if DBG\_DUMP

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase)

|| GetRecyclerFlagsTable().Trace.IsEnabled(Js::ScanStackPhase))

{

this->forceTraceMark = false;

Output::Print(L"\n");

Output::Flush();

}

#endif

RECYCLER\_PROFILE\_EXEC\_END(this, Js::ScanStackPhase);

RECYCLER\_STATS\_ADD(this, stackCount, this->collectionStats.markData.markCount - lastMarkCount);

GCETW(GC\_SCANSTACK\_STOP, (this));

return stackScanned;

}

#pragma warning(pop)

template <bool background>

size\_t Recycler::ScanPinnedObjects()

{

size\_t scanRootBytes = 0;

BEGIN\_DUMP\_OBJECT(this, L"Pinned");

{

this->TryMarkNonInterior(transientPinnedObject, &transientPinnedObject /\* parentReference \*/);

if (this->scanPinnedObjectMap)

{

// We are scanning the pinned object map now, we don't need to rescan unless

// we reset mark or we add stuff to the map in Recycler::AddRef

this->scanPinnedObjectMap = false;

pinnedObjectMap.MapAndRemoveIf([this, &scanRootBytes](void \* obj, PinRecord const& refCount)

{

if (refCount == 0)

{

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

Assert(refCount.stackBackTraces == nullptr);

#endif

// Only remove if we are not doing this in the background.

return !background;

}

this->TryMarkNonInterior(obj, static\_cast<void\*>(const\_cast<PinRecord\*>(&refCount)) /\* parentReference \*/);

scanRootBytes += sizeof(void \*);

return false;

});

if (!background)

{

this->hasPendingUnpinnedObject = false;

}

}

}

END\_DUMP\_OBJECT(this);

if (background)

{

// Re-enable resize now that we are done

pinnedObjectMap.EnableResize();

}

return scanRootBytes;

}

void

RecyclerScanMemoryCallback::operator()(void\*\* obj, size\_t byteCount)

{

this->recycler->ScanMemoryInline(obj, byteCount);

}

size\_t

Recycler::FindRoots()

{

size\_t scanRootBytes = 0;

#ifdef RECYCLER\_STATS

size\_t lastMarkCount = this->collectionStats.markData.markCount;

#endif

GCETW(GC\_SCANROOTS\_START, (this));

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::FindRootPhase);

#ifdef ENABLE\_PROJECTION

{

AUTO\_TIMESTAMP(externalWeakReferenceObjectResolve);

BEGIN\_DUMP\_OBJECT(this, L"External Weak Referenced Roots");

Assert(!this->IsInRefCountTrackingForProjection());

#if DBG

AutoIsInRefCountTrackingForProjection autoIsInRefCountTrackingForProjection(this);

#endif

collectionWrapper->MarkExternalWeakReferencedObjects(this->inPartialCollectMode);

END\_DUMP\_OBJECT(this);

}

#endif

// go through ITracker\* stuff. Don't need to do it if we are doing a partial collection

// as we keep track and mark all trackable objects.

// Do this first because the host might unpin stuff in the process

if (externalRootMarker != NULL)

{

#ifdef PARTIAL\_GC\_ENABLED

if (!this->inPartialCollectMode)

#endif

{

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::FindRootExtPhase);

#if DBG\_DUMP

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase)

|| GetRecyclerFlagsTable().Trace.IsEnabled(Js::FindRootPhase))

{

this->forceTraceMark = true;

Output::Print(L"Scanning External Roots: ");

}

#endif

BEGIN\_DUMP\_OBJECT(this, L"External Roots");

// PARTIALGC-TODO: How do we count external roots?

externalRootMarker(externalRootMarkerContext);

END\_DUMP\_OBJECT(this);

#if DBG\_DUMP

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase)

|| GetRecyclerFlagsTable().Trace.IsEnabled(Js::FindRootPhase))

{

this->forceTraceMark = false;

Output::Print(L"\n");

Output::Flush();

}

#endif

RECYCLER\_PROFILE\_EXEC\_END(this, Js::FindRootExtPhase);

}

}

#if DBG\_DUMP

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase)

|| GetRecyclerFlagsTable().Trace.IsEnabled(Js::FindRootPhase))

{

this->forceTraceMark = true;

Output::Print(L"Scanning Pinned Objects: ");

}

#endif

scanRootBytes += this->ScanPinnedObjects</\*background = \*/false>();

#if DBG\_DUMP

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::MarkPhase)

|| GetRecyclerFlagsTable().Trace.IsEnabled(Js::FindRootPhase))

{

this->forceTraceMark = false;

Output::Print(L"\n");

Output::Flush();

}

#endif

Assert(!this->hasPendingConcurrentFindRoot);

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::FindRootArenaPhase);

DListBase<GuestArenaAllocator>::EditingIterator guestArenaIter(&guestArenaList);

while (guestArenaIter.Next())

{

GuestArenaAllocator& allocator = guestArenaIter.Data();

if (allocator.pendingDelete)

{

Assert(this->hasPendingDeleteGuestArena);

allocator.SetLockBlockList(false);

guestArenaIter.RemoveCurrent(&HeapAllocator::Instance);

}

else if (this->backgroundFinishMarkCount == 0)

{

// Only scan arena if we haven't finished mark in the background

scanRootBytes += ScanArena(&allocator, false);

}

}

this->hasPendingDeleteGuestArena = false;

DList<ArenaData \*, HeapAllocator>::Iterator externalGuestArenaIter(&externalGuestArenaList);

while (externalGuestArenaIter.Next())

{

scanRootBytes += ScanArena(externalGuestArenaIter.Data(), false);

}

RECYCLER\_PROFILE\_EXEC\_END(this, Js::FindRootArenaPhase);

this->ScanImplicitRoots();

RECYCLER\_PROFILE\_EXEC\_END(this, Js::FindRootPhase);

GCETW(GC\_SCANROOTS\_STOP, (this));

RECYCLER\_STATS\_ADD(this, rootCount, this->collectionStats.markData.markCount - lastMarkCount);

return scanRootBytes;

}

void

Recycler::ScanImplicitRoots()

{

if (this->enableScanImplicitRoots)

{

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::FindImplicitRootPhase);

if (!this->hasScannedInitialImplicitRoots)

{

this->ScanInitialImplicitRoots();

this->hasScannedInitialImplicitRoots = true;

}

else

{

this->ScanNewImplicitRoots();

}

RECYCLER\_PROFILE\_EXEC\_END(this, Js::FindImplicitRootPhase);

}

}

size\_t

Recycler::TryMarkArenaMemoryBlockList(ArenaMemoryBlock \* memoryBlocks)

{

size\_t scanRootBytes = 0;

ArenaMemoryBlock \*blockp = memoryBlocks;

while (blockp != NULL)

{

void\*\* base=(void\*\*)blockp->GetBytes();

size\_t byteCount = blockp->nbytes;

scanRootBytes += byteCount;

this->ScanMemory(base, byteCount);

blockp = blockp->next;

}

return scanRootBytes;

}

size\_t

Recycler::TryMarkBigBlockListWithWriteWatch(BigBlock \* memoryBlocks)

{

DWORD pageSize = AutoSystemInfo::PageSize;

size\_t scanRootBytes = 0;

BigBlock \*blockp = memoryBlocks;

// Reset the write watch bit if we are scanning this in the background thread

DWORD const writeWatchFlags = this->IsConcurrentFindRootState()? WRITE\_WATCH\_FLAG\_RESET : 0;

while (blockp != NULL)

{

char \* currentAddress = (char \*)blockp->GetBytes();

char \* endAddress = currentAddress + blockp->currentByte;

char \* currentPageStart = (char \*)blockp->allocation;

while (currentAddress < endAddress)

{

void \* written;

ULONG\_PTR count = 1;

if (::GetWriteWatch(writeWatchFlags, currentPageStart, AutoSystemInfo::PageSize, &written, &count, &pageSize) != 0 || count == 1)

{

char \* currentEnd = min(currentPageStart + pageSize, endAddress);

size\_t byteCount = (size\_t)(currentEnd - currentAddress);

scanRootBytes += byteCount;

this->ScanMemory((void \*\*)currentAddress, byteCount);

}

currentPageStart += pageSize;

currentAddress = currentPageStart;

}

blockp = blockp->nextBigBlock;

}

return scanRootBytes;

}

size\_t

Recycler::TryMarkBigBlockList(BigBlock \* memoryBlocks)

{

size\_t scanRootBytes = 0;

BigBlock \*blockp = memoryBlocks;

while (blockp != NULL)

{

void\*\* base = (void\*\*)blockp->GetBytes();

size\_t byteCount = blockp->currentByte;

scanRootBytes += byteCount;

this->ScanMemory(base, byteCount);

blockp = blockp->nextBigBlock;

}

return scanRootBytes;

}

void

Recycler::ScanInitialImplicitRoots()

{

autoHeap.ScanInitialImplicitRoots();

}

void

Recycler::ScanNewImplicitRoots()

{

autoHeap.ScanNewImplicitRoots();

}

/\*------------------------------------------------------------------------------------------------

\* Mark

\*------------------------------------------------------------------------------------------------\*/

void

Recycler::ResetMarks(ResetMarkFlags flags)

{

Assert(!this->CollectionInProgress());

collectionState = CollectionStateResetMarks;

RecyclerVerboseTrace(GetRecyclerFlagsTable(), L"Reset marks\n");

GCETW(GC\_RESETMARKS\_START, (this));

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::ResetMarksPhase);

Assert(IsMarkStackEmpty());

this->scanPinnedObjectMap = true;

this->hasScannedInitialImplicitRoots = false;

heapBlockMap.ResetMarks();

autoHeap.ResetMarks(flags);

RECYCLER\_PROFILE\_EXEC\_END(this, Js::ResetMarksPhase);

GCETW(GC\_RESETMARKS\_STOP, (this));

#ifdef RECYCLER\_MARK\_TRACK

this->ClearMarkMap();

#endif

}

#ifdef RECYCLER\_MARK\_TRACK

void Recycler::ClearMarkMap()

{

this->markMap->Clear();

}

void Recycler::PrintMarkMap()

{

this->markMap->Map([](void\* key, void\* value)

{

Output::Print(L"0x%P => 0x%P\n", key, value);

});

}

#endif

#if DBG

void

Recycler::CheckAllocExternalMark() const

{

Assert(!disableThreadAccessCheck);

Assert(GetCurrentThreadContextId() == mainThreadId);

#ifdef CONCURRENT\_GC\_ENABLED

#ifdef HEAP\_ENUMERATION\_VALIDATION

Assert((this->IsMarkState() || this->IsPostEnumHeapValidationInProgress()) && collectionState != CollectionStateConcurrentMark);

#else

Assert(this->IsMarkState() && collectionState != CollectionStateConcurrentMark);

#endif

#else

Assert(this->IsMarkState());

#endif

}

#endif

void

Recycler::TryMarkNonInterior(void\* candidate, void\* parentReference)

{

#ifdef HEAP\_ENUMERATION\_VALIDATION

Assert(!isHeapEnumInProgress || this->IsPostEnumHeapValidationInProgress());

#else

Assert(!isHeapEnumInProgress);

#endif

Assert(this->collectionState != CollectionStateParallelMark);

markContext.Mark</\*parallel \*/ false, /\* interior \*/ false>(candidate, parentReference);

}

void

Recycler::TryMarkInterior(void\* candidate, void\* parentReference)

{

#ifdef HEAP\_ENUMERATION\_VALIDATION

Assert(!isHeapEnumInProgress || this->IsPostEnumHeapValidationInProgress());

#else

Assert(!isHeapEnumInProgress);

#endif

Assert(this->collectionState != CollectionStateParallelMark);

markContext.Mark</\*parallel \*/ false, /\* interior \*/ true>(candidate, parentReference);

}

template <bool parallel, bool interior>

void

Recycler::ProcessMarkContext(MarkContext \* markContext)

{

// Copying the markContext onto the stack messes up tracked object handling, because

// the tracked object will call TryMark[Non]Interior to report its references.

// These functions implicitly use the main markContext on the Recycler, but this will

// be overridden if we're processing the main markContext here.

// So, don't do this if we are going to process tracked objects.

// (This will be the case if we're not queuing and we're not in partial mode, which ignores tracked objects.)

// In this case we shouldn't be parallel anyway, so we don't need to worry about cache behavior.

// We should revisit how we manage markContexts in general in the future, and clean this up

// by passing the MarkContext through to the tracked object's Mark method.

if (this->inPartialCollectMode || DoQueueTrackedObject())

{

// The markContext as passed is one of the markContexts that lives on the Recycler.

// Copy it locally for processing.

// This serves two purposes:

// (1) Allow for better codegen because the markContext is local and we don't need to track the this pointer separately

// (because all the key processing is inlined into this function).

// (2) Ensure we don't have weird cache behavior because we're accidentally writing to the same cache line from

// multiple threads during parallel marking.

MarkContext localMarkContext = \*markContext;

// Do the actual marking.

localMarkContext.ProcessMark<parallel, interior>();

// Copy back to the original location.

\*markContext = localMarkContext;

// Clear the local mark context.

localMarkContext.Clear();

}

else

{

Assert(!parallel);

markContext->ProcessMark<parallel, interior>();

}

}

void

Recycler::ProcessMark(bool background)

{

if (background)

{

GCETW(GC\_BACKGROUNDMARK\_START, (this, backgroundRescanCount));

}

else

{

GCETW(GC\_MARK\_START, (this));

}

RECYCLER\_PROFILE\_EXEC\_THREAD\_BEGIN(background, this, Js::MarkPhase);

if (this->enableScanInteriorPointers)

{

this->ProcessMarkContext</\* parallel \*/ false, /\* interior \*/ true>(&markContext);

}

else

{

this->ProcessMarkContext</\* parallel \*/ false, /\* interior \*/ false>(&markContext);

}

RECYCLER\_PROFILE\_EXEC\_THREAD\_END(background, this, Js::MarkPhase);

if (background)

{

GCETW(GC\_BACKGROUNDMARK\_STOP, (this, backgroundRescanCount));

}

else

{

GCETW(GC\_MARK\_STOP, (this));

}

DebugOnly(this->markContext.VerifyPostMarkState());

}

void

Recycler::ProcessParallelMark(bool background, MarkContext \* markContext)

{

if (background)

{

GCETW(GC\_BACKGROUNDPARALLELMARK\_START, (this, backgroundRescanCount));

}

else

{

GCETW(GC\_PARALLELMARK\_START, (this));

}

RECYCLER\_PROFILE\_EXEC\_THREAD\_BEGIN(background, this, Js::MarkPhase);

if (this->enableScanInteriorPointers)

{

this->ProcessMarkContext</\* parallel \*/ true, /\* interior \*/ true>(markContext);

}

else

{

this->ProcessMarkContext</\* parallel \*/ true, /\* interior \*/ false>(markContext);

}

RECYCLER\_PROFILE\_EXEC\_THREAD\_END(background, this, Js::MarkPhase);

if (background)

{

GCETW(GC\_BACKGROUNDPARALLELMARK\_STOP, (this, backgroundRescanCount));

}

else

{

GCETW(GC\_PARALLELMARK\_STOP, (this));

}

}

void

Recycler::Mark()

{

// Marking in thread, we can just pre-mark them

ResetMarks(this->enableScanImplicitRoots ? ResetMarkFlags\_InThreadImplicitRoots : ResetMarkFlags\_InThread);

collectionState = CollectionStateFindRoots;

RootMark(CollectionStateMark);

}

#ifdef CONCURRENT\_GC\_ENABLED

void

Recycler::StartQueueTrackedObject()

{

Assert(!this->queueTrackedObject);

Assert(!this->HasPendingTrackObjects());

Assert(this->clientTrackedObjectList.Empty());

Assert(!this->inPartialCollectMode);

this->queueTrackedObject = true;

}

bool

Recycler::DoQueueTrackedObject() const

{

Assert(this->queueTrackedObject || !this->IsConcurrentMarkState());

Assert(this->queueTrackedObject || this->inPartialCollectMode || !(this->collectionState == CollectionStateParallelMark));

Assert(this->queueTrackedObject || this->isProcessingTrackedObjects || !this->HasPendingTrackObjects());

#ifdef PARTIAL\_GC\_ENABLED

Assert(!this->queueTrackedObject || (this->clientTrackedObjectList.Empty() && !this->inPartialCollectMode));

#endif

return this->queueTrackedObject;

}

#endif

void

Recycler::ResetCollectionState()

{

Assert(IsMarkStackEmpty());

this->collectionState = CollectionStateNotCollecting;

this->backgroundFinishMarkCount = 0;

this->inExhaustiveCollection = false;

this->inDecommitNowCollection = false;

CleanupPendingUnroot();

#ifdef PARTIAL\_GC\_ENABLED

if (inPartialCollectMode)

{

FinishPartialCollect();

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

Assert(!this->DoQueueTrackedObject());

#endif

#ifdef RECYCLER\_FINALIZE\_CHECK

// Reset the collection stats.

this->collectionStats.finalizeCount = this->autoHeap.liveFinalizableObjectCount - this->autoHeap.newFinalizableObjectCount - this->autoHeap.pendingDisposableObjectCount;

#endif

}

void

Recycler::ResetMarkCollectionState()

{

// If we aborted after doing a background Rescan, there will be entries in the markContext.

// Abort these entries and reset the markContext state.

markContext.Abort();

// If we aborted after doing a background parallel Mark, we wouldn't have cleaned up the

// parallel markContexts yet. Clean these up now.

// Note parallelMarkContext1 is not used in background parallel (see DoBackgroundParallelMark)

parallelMarkContext2.Cleanup();

parallelMarkContext3.Cleanup();

this->ClearNeedOOMRescan();

DebugOnly(this->isProcessingRescan = false);

#ifdef CONCURRENT\_GC\_ENABLED

// If we're reseting the mark collection state, we need to unlock the block list

DListBase<GuestArenaAllocator>::EditingIterator guestArenaIter(&guestArenaList);

while (guestArenaIter.Next())

{

GuestArenaAllocator& allocator = guestArenaIter.Data();

allocator.SetLockBlockList(false);

}

this->queueTrackedObject = false;

#endif

ResetCollectionState();

}

void

Recycler::ResetHeuristicCounters()

{

autoHeap.lastUncollectedAllocBytes = autoHeap.uncollectedAllocBytes;

autoHeap.uncollectedAllocBytes = 0;

autoHeap.uncollectedExternalBytes = 0;

ResetPartialHeuristicCounters();

}

void Recycler::ResetPartialHeuristicCounters()

{

#ifdef PARTIAL\_GC\_ENABLED

autoHeap.uncollectedNewPageCount = 0;

#endif

}

void

Recycler::ScheduleNextCollection()

{

this->tickCountNextCollection = ::GetTickCount() + RecyclerHeuristic::TickCountCollection;

this->tickCountNextFinishCollection = ::GetTickCount() + RecyclerHeuristic::TickCountFinishCollection;

}

#ifdef CONCURRENT\_GC\_ENABLED

void

Recycler::PrepareSweep()

{

autoHeap.PrepareSweep();

}

#endif

size\_t

Recycler::RescanMark(DWORD waitTime)

{

bool const onLowMemory = this->NeedOOMRescan();

Assert(this->inPartialCollectMode || DoQueueTrackedObject());

{

// We are about to do a rescan mark, which for consistency requires the runtime to stop any additional mutator threads

AUTO\_NO\_EXCEPTION\_REGION;

collectionWrapper->PreRescanMarkCallback();

}

// Always called in-thread

Assert(collectionState == CollectionStateRescanFindRoots);

#ifdef CONCURRENT\_GC\_ENABLED

if (!onLowMemory && // Don't do background finish mark if we are low on memory

// Only do background finish mark if we have a time limit or it is forced

(CUSTOM\_PHASE\_FORCE1(GetRecyclerFlagsTable(), Js::BackgroundFinishMarkPhase) || waitTime != INFINITE) &&

// Don't do background finish mark if we failed to finish mark too many times

(this->backgroundFinishMarkCount < RecyclerHeuristic::MaxBackgroundFinishMarkCount(this->GetRecyclerFlagsTable())))

{

this->PrepareBackgroundFindRoots();

if (StartConcurrent(CollectionStateConcurrentFinishMark))

{

this->backgroundFinishMarkCount++;

this->PrepareSweep();

GCETW(GC\_RESCANMARKWAIT\_START, (this, waitTime));

const BOOL waited = WaitForConcurrentThread(waitTime);

GCETW(GC\_RESCANMARKWAIT\_STOP, (this, !waited));

if (!waited)

{

CUSTOM\_PHASE\_PRINT\_TRACE1(GetRecyclerFlagsTable(), Js::BackgroundFinishMarkPhase, L"Finish mark timed out\n");

{

// We timed out doing the finish mark, notify the runtime

AUTO\_NO\_EXCEPTION\_REGION;

collectionWrapper->RescanMarkTimeoutCallback();

}

return Recycler::InvalidScanRootBytes;

}

Assert(collectionState == CollectionStateRescanWait);

collectionState = CollectionStateRescanFindRoots;

Assert(recyclerPageAllocator.GetWriteWatchPageCount() == 0);

Assert(recyclerLargeBlockPageAllocator.GetWriteWatchPageCount() == 0);

return this->backgroundRescanRootBytes;

}

this->RevertPrepareBackgroundFindRoots();

}

#endif

this->backgroundFinishMarkCount = 0;

return FinishMarkRescan(false) \* AutoSystemInfo::PageSize;

}

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t

Recycler::FinishMark(DWORD waitTime)

{

size\_t scannedRootBytes = RescanMark(waitTime);

Assert(waitTime != INFINITE || scannedRootBytes != Recycler::InvalidScanRootBytes);

if (scannedRootBytes != Recycler::InvalidScanRootBytes)

{

#if DBG

RecyclerVerboseTrace(GetRecyclerFlagsTable(), L"CTO: %d\n", this->clientTrackedObjectList.Count());

#endif

#ifdef PARTIAL\_GC\_ENABLED

if (this->inPartialCollectMode)

{

RecyclerVerboseTrace(GetRecyclerFlagsTable(), L"Processing client tracked objects\n");

ProcessClientTrackedObjects();

}

else

#endif

if (DoQueueTrackedObject())

{

RecyclerVerboseTrace(GetRecyclerFlagsTable(), L"Processing regular tracked objects\n");

ProcessTrackedObjects();

Assert(this->backgroundFinishMarkCount == 0 ||

(this->recyclerPageAllocator.GetWriteWatchPageCount() == 0 &&

this->recyclerLargeBlockPageAllocator.GetWriteWatchPageCount() == 0));

}

// Continue to mark from root one more time

scannedRootBytes += RootMark(CollectionStateRescanMark);

}

return scannedRootBytes;

}

#endif

void

Recycler::DoParallelMark()

{

Assert(this->enableParallelMark);

Assert(this->maxParallelism > 1 && this->maxParallelism <= 4);

// Split the mark stack into [this->maxParallelism] equal pieces.

// The actual # of splits is returned, in case the stack was too small to split that many ways.

MarkContext \* splitContexts[3] = { &parallelMarkContext1, &parallelMarkContext2, &parallelMarkContext3 };

uint actualSplitCount = markContext.Split(this->maxParallelism - 1, splitContexts);

Assert(actualSplitCount <= 3);

// If we failed to split at all, just mark in thread with no parallelism.

if (actualSplitCount == 0)

{

this->ProcessMark(false);

return;

}

// We need to queue tracked objects while we mark in parallel.

// (Unless it's a partial collect, in which case we don't process tracked objects at all)

if (!this->inPartialCollectMode)

{

StartQueueTrackedObject();

}

// Kick off marking on the background thread

bool concurrentSuccess = StartConcurrent(CollectionStateParallelMark);

// If there's enough work to split, then kick off marking on parallel threads too.

// If the threads haven't been created yet, this will create them (or fail).

bool parallelSuccess1 = false;

bool parallelSuccess2 = false;

if (concurrentSuccess && actualSplitCount >= 2)

{

parallelSuccess1 = parallelThread1.StartConcurrent();

if (parallelSuccess1 && actualSplitCount == 3)

{

parallelSuccess2 = parallelThread2.StartConcurrent();

}

}

// Process our portion of the split.

this->ProcessParallelMark(false, &parallelMarkContext1);

// If we successfully launched parallel work, wait for it to complete.

// If we failed, then process the work in-thread now.

if (concurrentSuccess)

{

WaitForConcurrentThread(INFINITE);

}

else

{

this->ProcessParallelMark(false, &markContext);

}

if (actualSplitCount >= 2)

{

if (parallelSuccess1)

{

parallelThread1.WaitForConcurrent();

}

else

{

this->ProcessParallelMark(false, &parallelMarkContext2);

}

if (actualSplitCount == 3)

{

if (parallelSuccess2)

{

parallelThread2.WaitForConcurrent();

}

else

{

this->ProcessParallelMark(false, &parallelMarkContext3);

}

}

}

this->collectionState = CollectionStateMark;

// Process tracked objects, if any, then do one final mark phase in case they marked any new objects.

// (Unless it's a partial collect, in which case we don't process tracked objects at all)

if (!this->inPartialCollectMode)

{

this->ProcessTrackedObjects();

this->ProcessMark(false);

}

else

{

Assert(!this->HasPendingTrackObjects());

}

}

void

Recycler::DoBackgroundParallelMark()

{

// Split the mark stack into [this->maxParallelism - 1] equal pieces (thus, "- 2" below).

// The actual # of splits is returned, in case the stack was too small to split that many ways.

// The parallel threads are hardwired to use parallelMarkContext2/3, so we split using those.

uint actualSplitCount = 0;

MarkContext \* splitContexts[2] = { &parallelMarkContext2, &parallelMarkContext3 };

if (this->enableParallelMark)

{

Assert(this->maxParallelism > 1 && this->maxParallelism <= 4);

if (this->maxParallelism > 2)

{

actualSplitCount = markContext.Split(this->maxParallelism - 2, splitContexts);

}

}

Assert(actualSplitCount <= 2);

// If we failed to split at all, just mark in thread with no parallelism.

if (actualSplitCount == 0)

{

this->ProcessMark(true);

return;

}

// We should already be set up to queue tracked objects, unless this is a partial collect

Assert(this->DoQueueTrackedObject() || this->inPartialCollectMode);

this->collectionState = CollectionStateBackgroundParallelMark;

// Kick off marking on parallel threads too, if there is work for them

// If the threads haven't been created yet, this will create them (or fail).

bool parallelSuccess1 = false;

bool parallelSuccess2 = false;

parallelSuccess1 = parallelThread1.StartConcurrent();

if (parallelSuccess1 && actualSplitCount == 2)

{

parallelSuccess2 = parallelThread2.StartConcurrent();

}

// Process our portion of the split.

this->ProcessParallelMark(true, &markContext);

// If we successfully launched parallel work, wait for it to complete.

// If we failed, then process the work in-thread now.

if (parallelSuccess1)

{

parallelThread1.WaitForConcurrent();

}

else

{

this->ProcessParallelMark(true, &parallelMarkContext2);

}

if (actualSplitCount == 2)

{

if (parallelSuccess2)

{

parallelThread2.WaitForConcurrent();

}

else

{

this->ProcessParallelMark(true, &parallelMarkContext3);

}

}

this->collectionState = CollectionStateConcurrentMark;

}

size\_t

Recycler::RootMark(CollectionState markState)

{

size\_t scannedRootBytes = 0;

Assert(!this->NeedOOMRescan() || markState == CollectionStateRescanMark);

RecyclerVerboseTrace(GetRecyclerFlagsTable(), L"PreMark done, partial collect: %d\n", this->inPartialCollectMode);

Assert(collectionState == (markState == CollectionStateMark? CollectionStateFindRoots : CollectionStateRescanFindRoots));

BOOL stacksScannedByRuntime = FALSE;

{

// We are about to scan roots in thread, notify the runtime first so it can stop threads if necessary and also provide additional roots

AUTO\_NO\_EXCEPTION\_REGION;

RecyclerScanMemoryCallback scanMemory(this);

scannedRootBytes += collectionWrapper->RootMarkCallback(scanMemory, &stacksScannedByRuntime);

}

scannedRootBytes += FindRoots();

if (!stacksScannedByRuntime)

{

// The runtime did not scan the stack(s) for us, so we use the normal Recycler code.

scannedRootBytes += ScanStack();

}

this->collectionState = markState;

if (this->enableParallelMark)

{

this->DoParallelMark();

}

else

{

this->ProcessMark(false);

}

if (this->EndMark())

{

// return large root scanned byte to not get into partial mode if we are low on memory

scannedRootBytes = RecyclerSweep::MaxPartialCollectRescanRootBytes + 1;

}

return scannedRootBytes;

}

bool

Recycler::EndMarkCheckOOMRescan()

{

bool oomRescan = false;

if (this->NeedOOMRescan())

{

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

if (this->objectGraphDumper)

{

// Do not complete the mark if we are just dumping the object graph

// Just report out of memory

this->objectGraphDumper->isOutOfMemory = true;

this->ClearNeedOOMRescan();

}

else

#endif

{

EndMarkOnLowMemory();

oomRescan = true;

}

}

// Done with the mark stack, it should be empty.

// Release pages it is holding.

Assert(!HasPendingMarkObjects());

Assert(!HasPendingTrackObjects());

return oomRescan;

}

bool

Recycler::EndMark()

{

Assert(!this->DoQueueTrackedObject());

Assert(this->clientTrackedObjectList.Empty());

{

// We have finished marking

AUTO\_NO\_EXCEPTION\_REGION;

collectionWrapper->EndMarkCallback();

}

bool oomRescan = EndMarkCheckOOMRescan();

if (ProcessObjectBeforeCollectCallbacks())

{

// callbacks may trigger additional marking, need to check OOMRescan again

oomRescan |= EndMarkCheckOOMRescan();

}

// GC-CONSIDER: Consider keeping some page around

GCETW(GC\_DECOMMIT\_CONCURRENT\_COLLECT\_PAGE\_ALLOCATOR\_START, (this));

// Clean up mark contexts, which will release held free pages

// Do this for all contexts before we decommit, to make sure all pages are freed

markContext.Cleanup();

parallelMarkContext1.Cleanup();

parallelMarkContext2.Cleanup();

parallelMarkContext3.Cleanup();

// Decommit all pages

markContext.DecommitPages();

parallelMarkContext1.DecommitPages();

parallelMarkContext2.DecommitPages();

parallelMarkContext3.DecommitPages();

GCETW(GC\_DECOMMIT\_CONCURRENT\_COLLECT\_PAGE\_ALLOCATOR\_STOP, (this));

return oomRescan;

}

void

Recycler::EndMarkOnLowMemory()

{

GCETW(GC\_ENDMARKONLOWMEMORY\_START, (this));

Assert(this->NeedOOMRescan());

this->inEndMarkOnLowMemory = true;

// Treat this as a concurrent mark reset so that we don't invalidate the allocators

RecyclerVerboseTrace(GetRecyclerFlagsTable(), L"OOM during mark- rerunning mark\n");

// Try to release as much memory as possible

ForRecyclerPageAllocator(DecommitNow());

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

uint iterations = 0;

#endif

do

{

Assert(this->clientTrackedObjectList.Empty());

// Always queue tracked objects during rescan, to avoid changes to mark state.

// (Unless we're in a partial, in which case we ignore tracked objects)

Assert(!this->DoQueueTrackedObject());

#ifdef PARTIAL\_GC\_ENABLED

if (!this->inPartialCollectMode)

#endif

{

this->StartQueueTrackedObject();

}

this->collectionState = CollectionStateRescanFindRoots;

this->ClearNeedOOMRescan();

#if DBG

Assert(!this->isProcessingRescan);

this->isProcessingRescan = true;

#endif

if (!heapBlockMap.OOMRescan(this))

{

// Kill the process- we couldn't even rescan a single block

// We are in pretty low memory state at this point

// The fail-fast is present for two reasons:

// 1) Defense-in-depth for cases we hadn't thought about

// 2) Deal with cases like -MaxMarkStackPageCount:1 which can still hang without the fail-fast

MarkStack\_OOM\_fatal\_error();

}

autoHeap.Rescan(RescanFlags\_None);

DebugOnly(this->isProcessingRescan = false);

this->ProcessMark(false);

// Process any tracked objects we found

#ifdef PARTIAL\_GC\_ENABLED

if (!this->inPartialCollectMode)

#endif

{

ProcessTrackedObjects();

}

// Drain the mark stack

ProcessMark(false);

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

iterations++;

#endif

}

while (this->NeedOOMRescan());

Assert(!markContext.GetPageAllocator()->DisableAllocationOutOfMemory());

Assert(!parallelMarkContext1.GetPageAllocator()->DisableAllocationOutOfMemory());

Assert(!parallelMarkContext2.GetPageAllocator()->DisableAllocationOutOfMemory());

Assert(!parallelMarkContext3.GetPageAllocator()->DisableAllocationOutOfMemory());

CUSTOM\_PHASE\_PRINT\_TRACE1(GetRecyclerFlagsTable(), Js::RecyclerPhase, L"EndMarkOnLowMemory iterations: %d\n", iterations);

Assert(this->clientTrackedObjectList.Empty());

Assert(!this->DoQueueTrackedObject());

this->inEndMarkOnLowMemory = false;

#ifdef PARTIAL\_GC\_ENABLED

if (this->inPartialCollectMode)

{

this->FinishPartialCollect();

}

#endif

GCETW(GC\_ENDMARKONLOWMEMORY\_STOP, (this));

}

#if DBG

bool

Recycler::IsMarkStackEmpty()

{

return (markContext.IsEmpty() && parallelMarkContext1.IsEmpty() && parallelMarkContext2.IsEmpty() && parallelMarkContext3.IsEmpty());

}

#endif

#ifdef HEAP\_ENUMERATION\_VALIDATION

void

Recycler::PostHeapEnumScan(PostHeapEnumScanCallback callback, void \*data)

{

this->pfPostHeapEnumScanCallback = callback;

this->postHeapEnunScanData = data;

FindRoots();

ProcessMark(false);

this->pfPostHeapEnumScanCallback = NULL;

this->postHeapEnunScanData = NULL;

}

#endif

bool

Recycler::QueueTrackedObject(FinalizableObject \* trackableObject)

{

return markContext.AddTrackedObject(trackableObject);

}

bool

Recycler::FindImplicitRootObject(void\* candidate, RecyclerHeapObjectInfo& heapObject)

{

HeapBlock\* heapBlock = FindHeapBlock(candidate);

if (heapBlock == nullptr)

{

return false;

}

if (heapBlock->GetHeapBlockType() < HeapBlock::HeapBlockType::SmallAllocBlockTypeCount)

{

return ((SmallHeapBlock\*)heapBlock)->FindImplicitRootObject(candidate, this, heapObject);

}

else if (!heapBlock->IsLargeHeapBlock())

{

return ((MediumHeapBlock\*)heapBlock)->FindImplicitRootObject(candidate, this, heapObject);

}

else

{

return ((LargeHeapBlock\*)heapBlock)->FindImplicitRootObject(candidate, this, heapObject);

}

}

bool

Recycler::FindHeapObject(void\* candidate, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject)

{

HeapBlock\* heapBlock = FindHeapBlock(candidate);

return heapBlock && heapBlock->FindHeapObject(candidate, this, flags, heapObject);

}

bool

Recycler::FindHeapObjectWithClearedAllocators(void\* candidate, RecyclerHeapObjectInfo& heapObject)

{

// Heap enum has some case where it allocates, so we can't assert

Assert(autoHeap.AllocatorsAreEmpty() || this->isHeapEnumInProgress);

return FindHeapObject(candidate, FindHeapObjectFlags\_ClearedAllocators, heapObject);

}

void\*

Recycler::GetRealAddressFromInterior(void\* candidate)

{

HeapBlock \* heapBlock = heapBlockMap.GetHeapBlock(candidate);

if (heapBlock == NULL)

{

return NULL;

}

return heapBlock->GetRealAddressFromInterior(candidate);

}

/\*------------------------------------------------------------------------------------------------

\* Sweep

\*------------------------------------------------------------------------------------------------\*/

bool

Recycler::Sweep(size\_t rescanRootBytes, bool concurrent, bool adjustPartialHeuristics)

{

Assert(!this->hasBackgroundFinishPartial);

#ifdef CONCURRENT\_GC\_ENABLED

if (!this->enableConcurrentSweep)

#endif

{

concurrent = false;

}

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, concurrent? Js::ConcurrentSweepPhase : Js::SweepPhase);

recyclerSweepInstance.BeginSweep(this, rescanRootBytes, adjustPartialHeuristics);

this->SweepHeap(concurrent, \*recyclerSweep);

if (concurrent)

{

// If we finished mark in the background, all the relevant write watches should already be reset

// Only reset write watch if we didn't finish mark in the background

if (this->backgroundFinishMarkCount == 0)

{

if (this->inPartialCollectMode)

{

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::ResetWriteWatchPhase);

if (!recyclerPageAllocator.ResetWriteWatch() || !recyclerLargeBlockPageAllocator.ResetWriteWatch())

{

// Shouldn't happen

Assert(false);

// Disable partial collect

this->enablePartialCollect = false;

// We haven't done any partial collection yet, just get out of partial collect mode

this->inPartialCollectMode = false;

}

RECYCLER\_PROFILE\_EXEC\_END(this, Js::ResetWriteWatchPhase);

}

}

}

else

{

recyclerSweep->FinishSweep();

recyclerSweep->EndSweep();

}

RECYCLER\_PROFILE\_EXEC\_END(this, concurrent? Js::ConcurrentSweepPhase : Js::SweepPhase);

#ifdef CONCURRENT\_GC\_ENABLED

if (concurrent)

{

if (!StartConcurrent(CollectionStateConcurrentSweep))

{

// Failed to spawn the concurrent sweep.

// Instead, force the concurrent sweep to happen right here in thread.

this->collectionState = CollectionStateConcurrentSweep;

DoBackgroundWork(true);

// Continue as if the concurrent sweep were executing

// Next time we check for completion, we will finish the sweep just as if it had happened out of thread.

}

return true;

}

#endif

return false;

}

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

void Recycler::DisplayMemStats()

{

#ifdef PERF\_COUNTERS

#if DBG\_DUMP

printf("Recycler Live Object Count %u\n", PerfCounter::RecyclerCounterSet::GetLiveObjectCounter().GetValue());

printf("Recycler Live Object Size %u\n", PerfCounter::RecyclerCounterSet::GetLiveObjectSizeCounter().GetValue());

#endif

printf("Recycler Used Page Size %u\n", PerfCounter::PageAllocatorCounterSet::GetUsedSizeCounter(PageAllocatorType::PageAllocatorType\_Recycler).GetValue());

#endif

}

#endif

CollectedRecyclerWeakRefHeapBlock CollectedRecyclerWeakRefHeapBlock::Instance;

void

Recycler::SweepWeakReference()

{

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::SweepWeakPhase);

GCETW(GC\_SWEEP\_WEAKREF\_START, (this));

// REVIEW: Clean up the weak reference map concurrently?

bool hasCleanup = false;

weakReferenceMap.Map([&hasCleanup](RecyclerWeakReferenceBase \* weakRef) -> bool

{

if (!weakRef->weakRefHeapBlock->TestObjectMarkedBit(weakRef))

{

hasCleanup = true;

// Remove

return false;

}

if (!weakRef->strongRefHeapBlock->TestObjectMarkedBit(weakRef->strongRef))

{

hasCleanup = true;

weakRef->strongRef = nullptr;

// Put in a dummy heap block so that we can still do the isPendingConcurrentSweep check first.

weakRef->strongRefHeapBlock = &CollectedRecyclerWeakRefHeapBlock::Instance;

// Remove

return false;

}

// Keep

return true;

});

this->weakReferenceCleanupId += hasCleanup;

GCETW(GC\_SWEEP\_WEAKREF\_STOP, (this));

RECYCLER\_PROFILE\_EXEC\_END(this, Js::SweepWeakPhase);

}

void

Recycler::SweepHeap(bool concurrent, RecyclerSweep& recyclerSweep)

{

Assert(!this->hasPendingDeleteGuestArena);

Assert(!this->isHeapEnumInProgress);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(!this->DoQueueTrackedObject());

if (concurrent)

{

collectionState = CollectionStateSetupConcurrentSweep;

// Only queue up non-leaf pages- leaf pages don't need to be zeroed out

recyclerPageAllocator.StartQueueZeroPage();

recyclerLargeBlockPageAllocator.StartQueueZeroPage();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.StartQueueZeroPage();

#endif

}

else

#endif

{

Assert(!concurrent);

collectionState = CollectionStateSweep;

}

this->SweepWeakReference();

#ifdef CONCURRENT\_GC\_ENABLED

if (concurrent)

{

GCETW(GC\_SETUPBACKGROUNDSWEEP\_START, (this));

}

else

#endif

{

GCETW(GC\_SWEEP\_START, (this));

}

recyclerPageAllocator.SuspendIdleDecommit();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.SuspendIdleDecommit();

#endif

recyclerLargeBlockPageAllocator.SuspendIdleDecommit();

autoHeap.Sweep(recyclerSweep, concurrent);

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.ResumeIdleDecommit();

#endif

recyclerPageAllocator.ResumeIdleDecommit();

recyclerLargeBlockPageAllocator.ResumeIdleDecommit();

#ifdef CONCURRENT\_GC\_ENABLED

if (concurrent)

{

recyclerPageAllocator.StopQueueZeroPage();

recyclerLargeBlockPageAllocator.StopQueueZeroPage();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.StopQueueZeroPage();

#endif

GCETW(GC\_SETUPBACKGROUNDSWEEP\_STOP, (this));

}

else

{

Assert(!recyclerPageAllocator.HasZeroQueuedPages());

Assert(!recyclerLargeBlockPageAllocator.HasZeroQueuedPages());

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

Assert(!recyclerWithBarrierPageAllocator.HasZeroQueuedPages());

#endif

uint sweptBytes = 0;

#ifdef RECYCLER\_STATS

sweptBytes = (uint)collectionStats.objectSweptBytes;

#endif

GCETW(GC\_SWEEP\_STOP, (this, sweptBytes));

}

#endif

}

void

Recycler::BackgroundFinishPartialCollect(RecyclerSweep \* recyclerSweep)

{

Assert(this->inPartialCollectMode);

Assert(recyclerSweep != nullptr && recyclerSweep->IsBackground());

this->hasBackgroundFinishPartial = true;

this->autoHeap.FinishPartialCollect(recyclerSweep);

this->inPartialCollectMode = false;

}

void

Recycler::DisposeObjects()

{

Assert(this->allowDispose && this->hasDisposableObject && !this->inDispose);

Assert(!isHeapEnumInProgress);

GCETW(GC\_DISPOSE\_START, (this));

ASYNC\_HOST\_OPERATION\_START(collectionWrapper);

this->inDispose = true;

#ifdef PROFILE\_RECYCLER\_ALLOC

// finalizer may allocate memory and dispose object can happen in the middle of allocation

// save and restore the tracked object info

TrackAllocData oldAllocData = { 0 };

if (trackerDictionary != nullptr)

{

oldAllocData = nextAllocData;

nextAllocData.Clear();

}

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::RecyclerPhase))

{

Output::Print(L"Disposing objects\n");

}

#endif

// Disable dispose within this method, restore it when we're done

AutoRestoreValue<bool> disableDispose(&this->allowDispose, false);

#ifdef FAULT\_INJECTION

this->collectionWrapper->DisposeScriptContextByFaultInjectionCallBack();

#endif

// Scope timestamp to just dispose

{

AUTO\_TIMESTAMP(dispose);

autoHeap.DisposeObjects();

}

#ifdef PROFILE\_RECYCLER\_ALLOC

if (trackerDictionary != nullptr)

{

Assert(nextAllocData.IsEmpty());

nextAllocData = oldAllocData;

}

#endif

#ifdef ENABLE\_PROJECTION

{

Assert(!this->inResolveExternalWeakReferences);

Assert(!this->allowDispose);

#if DBG || defined RECYCLER\_TRACE

AutoRestoreValue<bool> inResolveExternalWeakReferencedObjects(&this->inResolveExternalWeakReferences, true);

#endif

AUTO\_TIMESTAMP(externalWeakReferenceObjectResolve);

// This is where it is safe to resolve external weak references as they can lead to new script entry

collectionWrapper->ResolveExternalWeakReferencedObjects();

}

#endif

Assert(!this->inResolveExternalWeakReferences);

Assert(this->inDispose);

this->inDispose = false;

ASYNC\_HOST\_OPERATION\_END(collectionWrapper);

uint sweptBytes = 0;

#ifdef RECYCLER\_STATS

sweptBytes = (uint)collectionStats.objectSweptBytes;

#endif

GCETW(GC\_DISPOSE\_STOP, (this, sweptBytes));

}

bool

Recycler::FinishDisposeObjects()

{

CUSTOM\_PHASE\_PRINT\_TRACE1(GetRecyclerFlagsTable(), Js::DisposePhase, L"[Dispose] AllowDispose in FinishDisposeObject: %d\n", this->allowDispose);

if (this->hasDisposableObject && this->allowDispose)

{

CUSTOM\_PHASE\_PRINT\_TRACE1(GetRecyclerFlagsTable(), Js::DisposePhase, L"[Dispose] FinishDisposeObject, calling Dispose: %d\n", this->allowDispose);

#ifdef RECYCLER\_TRACE

CollectionParam savedCollectionParam = collectionParam;

#endif

DisposeObjects();

#ifdef RECYCLER\_TRACE

collectionParam = savedCollectionParam;

#endif

// FinishDisposeObjects is always called either during a collection,

// or we will check the NeedExhaustiveRepeatCollect(), so no need to check it here

return true;

}

#ifdef RECYCLER\_TRACE

if (!this->inDispose && this->hasDisposableObject

&& GetRecyclerFlagsTable().Trace.IsEnabled(Js::RecyclerPhase))

{

Output::Print(L"%04X> RC(%p): %s\n", this->mainThreadId, this, L"Dispose object delayed");

}

#endif

return false;

}

template bool Recycler::FinishDisposeObjectsNow<FinishDispose>();

template bool Recycler::FinishDisposeObjectsNow<FinishDisposeTimed>();

template <CollectionFlags flags>

bool

Recycler::FinishDisposeObjectsNow()

{

if (inDisposeWrapper)

{

return false;

}

return FinishDisposeObjectsWrapped<flags>();

}

template <CollectionFlags flags>

\_\_inline

bool

Recycler::FinishDisposeObjectsWrapped()

{

const BOOL allowDisposeFlag = flags & CollectOverride\_AllowDispose;

if (allowDisposeFlag && this->NeedDispose())

{

if ((flags & CollectHeuristic\_TimeIfScriptActive) == CollectHeuristic\_TimeIfScriptActive)

{

if (!this->NeedDisposeTimed())

{

return false;

}

}

this->allowDispose = true;

this->inDisposeWrapper = true;

#ifdef RECYCLER\_TRACE

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::RecyclerPhase))

{

Output::Print(L"%04X> RC(%p): %s\n", this->mainThreadId, this, L"Process delayed dispose object");

}

#endif

collectionWrapper->DisposeObjects(this);

// Dispose may get into message loop and cause a reentrant GC. If those don't allow reentrant

// it will get added to a pending collect request.

// FinishDipsosedObjectsWrapped/DisposeObjectsWrapped is called at a place that might not be during a collection

// and won't check NeedExhaustiveRepeatCollect(), need to check it here to honor those requests

if (!this->CollectionInProgress() && NeedExhaustiveRepeatCollect() && ((flags & CollectOverride\_NoExhaustiveCollect) != CollectOverride\_NoExhaustiveCollect))

{

#ifdef RECYCLER\_TRACE

CaptureCollectionParam((CollectionFlags)(flags & ~CollectMode\_Partial), true);

#endif

DoCollectWrapped((CollectionFlags)(flags & ~CollectMode\_Partial));

}

this->inDisposeWrapper = false;

return true;

}

return false;

}

/\*------------------------------------------------------------------------------------------------

\* Collect

\*------------------------------------------------------------------------------------------------\*/

BOOL

Recycler::CollectOnAllocatorThread()

{

#ifdef PARTIAL\_GC\_ENABLED

Assert(!inPartialCollectMode);

#endif

#ifdef RECYCLER\_TRACE

PrintCollectTrace(Js::GarbageCollectPhase);

#endif

this->CollectionBegin<Js::GarbageCollectPhase>();

this->Mark();

// Partial collect mode is not re-enabled after a non-partial in-thread GC because partial GC heuristics are not adjusted

// after a full in-thread GC. Enabling partial collect mode causes partial GC heuristics to be reset before the next full

// in-thread GC, thereby allowing partial GC to kick in more easily without being able to adjust heuristics after the full

// GCs. Until we have a way of adjusting partial GC heuristics after a full in-thread GC, once partial collect mode is

// turned off, it will remain off until a concurrent GC happens

this->Sweep();

this->CollectionEnd<Js::GarbageCollectPhase>();

FinishCollection();

return true;

}

// Explicitly instantiate all possible modes

template BOOL Recycler::CollectNow<CollectOnScriptIdle>();

template BOOL Recycler::CollectNow<CollectOnScriptExit>();

template BOOL Recycler::CollectNow<CollectOnAllocation>();

template BOOL Recycler::CollectNow<CollectOnTypedArrayAllocation>();

template BOOL Recycler::CollectNow<CollectOnScriptCloseNonPrimary>();

template BOOL Recycler::CollectNow<CollectExhaustiveCandidate>();

template BOOL Recycler::CollectNow<CollectNowConcurrent>();

template BOOL Recycler::CollectNow<CollectNowExhaustive>();

template BOOL Recycler::CollectNow<CollectNowDecommitNowExplicit>();

template BOOL Recycler::CollectNow<CollectNowPartial>();

template BOOL Recycler::CollectNow<CollectNowConcurrentPartial>();

template BOOL Recycler::CollectNow<CollectNowForceInThread>();

template BOOL Recycler::CollectNow<CollectNowForceInThreadExternal>();

template BOOL Recycler::CollectNow<CollectNowForceInThreadExternalNoStack>();

template BOOL Recycler::CollectNow<CollectOnRecoverFromOutOfMemory>();

template BOOL Recycler::CollectNow<CollectNowDefault>();

template BOOL Recycler::CollectNow<CollectOnSuspendCleanup>();

template BOOL Recycler::CollectNow<CollectNowDefaultLSCleanup>();

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

template BOOL Recycler::CollectNow<CollectNowFinalGC>();

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

template BOOL Recycler::CollectNow<CollectNowExhaustiveSkipStack>();

#endif

template <CollectionFlags flags>

BOOL

Recycler::CollectNow()

{

// Force-in-thread cannot be concurrent or partial

CompileAssert((flags & CollectOverride\_ForceInThread) == 0 || (flags & (CollectMode\_Concurrent | CollectMode\_Partial)) == 0);

// Collections not allowed when the recycler is currently executing the PostCollectionCallback

if (collectionState == CollectionStatePostCollectionCallback)

{

return false;

}

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

if ((disableCollection && (flags & CollectOverride\_Explicit) == 0) || isShuttingDown)

#else

if (isShuttingDown)

#endif

{

Assert(collectionState == CollectionStateNotCollecting

|| collectionState == CollectionStateExit

|| this->isShuttingDown);

return false;

}

if (flags & CollectOverride\_ExhaustiveCandidate)

{

return CollectWithExhaustiveCandidate<flags>();

}

return CollectInternal<flags>();

}

template <CollectionFlags flags>

BOOL

Recycler::GetPartialFlag()

{

#ifdef PARTIAL\_GC\_ENABLED

#pragma prefast(suppress:6313, "flags is a template parameter and can be 0")

return(flags & CollectMode\_Partial) && inPartialCollectMode;

#else

return false;

#endif

}

template <CollectionFlags flags>

BOOL

Recycler::CollectWithExhaustiveCandidate()

{

Assert(flags & CollectOverride\_ExhaustiveCandidate);

// Currently we don't have any exhaustive candidate that has heuristic.

Assert((flags & CollectHeuristic\_Mask & ~CollectHeuristic\_Never) == 0);

this->hasExhaustiveCandidate = true;

if (flags & CollectHeuristic\_Never)

{

// This is just an exhaustive candidate notification. Don't trigger a GC.

return false;

}

// Continue with the GC heuristic

return CollectInternal<flags>();

}

template <CollectionFlags flags>

BOOL

Recycler::CollectInternal()

{

// CollectHeuristic\_Never flag should only be used with exhaustive candidate

Assert((flags & CollectHeuristic\_Never) == 0);

// If we're in a re-entrant state, we want to allow GC to be triggered only

// from allocation (or trigger points with AllowReentrant). This is to minimize

// the number of reentrant GCs

if ((flags & CollectOverride\_AllowReentrant) == 0 && this->inDispose)

{

return false;

}

#ifdef RECYCLER\_TRACE

CaptureCollectionParam(flags);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

const BOOL concurrent = flags & CollectMode\_Concurrent;

const BOOL finishConcurrent = flags & CollectOverride\_FinishConcurrent;

// If we priority boosted, we should try to finish it every chance we get

// Otherwise, we should finishing it if we are not doing a concurrent GC,

// or the flags tell us to always try to finish a concurrent GC (CollectOverride\_FinishConcurrent)

if ((!concurrent || finishConcurrent || priorityBoost) && this->CollectionInProgress())

{

return TryFinishConcurrentCollect<flags>();

}

#endif

if (flags & CollectHeuristic\_Mask)

{

// Check some heuristics first before starting a collection

return CollectWithHeuristic<flags>();

}

// Start a collection now.

return Collect<flags>();

}

template <CollectionFlags flags>

BOOL

Recycler::CollectWithHeuristic()

{

// CollectHeuristic\_Never flag should only be used with exhaustive candidate

Assert((flags & CollectHeuristic\_Never) == 0);

const BOOL allocSize = flags & CollectHeuristic\_AllocSize;

const BOOL timedIfScriptActive = flags & CollectHeuristic\_TimeIfScriptActive;

const BOOL timedIfInScript = flags & CollectHeuristic\_TimeIfInScript;

const BOOL timed = (timedIfScriptActive && isScriptActive) || (timedIfInScript && isInScript) || (flags & CollectHeuristic\_Time);

#ifdef PARTIAL\_GC\_ENABLED

if (GetPartialFlag<flags>())

{

Assert(enablePartialCollect);

Assert(allocSize);

Assert(this->uncollectedNewPageCountPartialCollect >= RecyclerSweep::MinPartialUncollectedNewPageCount

&& this->uncollectedNewPageCountPartialCollect <= RecyclerHeuristic::Instance.MaxPartialUncollectedNewPageCount);

// PARTIAL-GC-REVIEW: For now, we have only alloc size heuristic

// Maybe improve this heuristic by looking at how many free pages are in the page allocator.

if (autoHeap.uncollectedNewPageCount > this->uncollectedNewPageCountPartialCollect)

{

return Collect<flags>();

}

}

#endif

// allocation byte count heuristic, collect every 1 MB allocated

if (allocSize && (autoHeap.uncollectedAllocBytes < RecyclerHeuristic::UncollectedAllocBytesCollection()))

{

return FinishDisposeObjectsWrapped<flags>();

}

// time heuristic, allocate every 1000 clock tick, or 64 MB is allocated in a short time

if (timed && (autoHeap.uncollectedAllocBytes < RecyclerHeuristic::Instance.MaxUncollectedAllocBytes))

{

uint currentTickCount = GetTickCount();

#ifdef RECYCLER\_TRACE

collectionParam.timeDiff = currentTickCount - tickCountNextCollection;

#endif

if ((int)(tickCountNextCollection - currentTickCount) >= 0)

{

return FinishDisposeObjectsWrapped<flags>();

}

}

#ifdef RECYCLER\_TRACE

else

{

uint currentTickCount = GetTickCount();

collectionParam.timeDiff = currentTickCount - tickCountNextCollection;

}

#endif

// Passed all the heuristic, do some GC work, maybe

return Collect<(CollectionFlags)(flags & ~CollectMode\_Partial)>();

}

template <CollectionFlags flags>

BOOL

Recycler::Collect()

{

#ifdef CONCURRENT\_GC\_ENABLED

if (this->CollectionInProgress())

{

// If we are forced in thread, we can't be concurrent

// If we are not concurrent we should have been handled before in CollectInternal and we shouldn't be here

Assert((flags & CollectOverride\_ForceInThread) == 0);

Assert((flags & CollectMode\_Concurrent) != 0);

return TryFinishConcurrentCollect<flags>();

}

#endif

SetupPostCollectionFlags<flags>();

const BOOL partial = GetPartialFlag<flags>();

CollectionFlags finalFlags = flags;

if (!partial)

{

finalFlags = (CollectionFlags)(flags & ~CollectMode\_Partial);

}

// ExecuteRecyclerCollectionFunction may cause exception. In which case, we may trigger the assert

// in SetupPostCollectionFlags because we didn't reset the inExhausitvECollection variable if

// an exception. Use this flag to disable it the assertion if exception occur

DebugOnly(this->hasIncompletedDoCollect = true);

{

RECORD\_TIMESTAMP(initialCollectionStartTime);

this->telemetryBlock->initialCollectionStartProcessUsedBytes = PageAllocator::GetProcessUsedBytes();

this->telemetryBlock->exhaustiveRepeatedCount = 0;

return DoCollectWrapped(finalFlags);

}

}

template <CollectionFlags flags>

void Recycler::SetupPostCollectionFlags()

{

// If we are not in a collection (collection in progress or in dispose), inExhastivecollection should not be set

// Otherwise, we have missed an exhaustive collection.

Assert(this->hasIncompletedDoCollect ||

this->CollectionInProgress() || this->inDispose || (!this->inExhaustiveCollection && !this->inDecommitNowCollection));

// Record whether we want to start exhaustive detection or do decommit now after GC

const BOOL exhaustive = flags & CollectMode\_Exhaustive;

const BOOL decommitNow = flags & CollectMode\_DecommitNow;

const BOOL cacheCleanup = flags & CollectMode\_CacheCleanup;

if (decommitNow)

{

this->inDecommitNowCollection = true;

}

if (exhaustive)

{

this->inExhaustiveCollection = true;

}

if (cacheCleanup)

{

this->inCacheCleanupCollection = true;

}

}

BOOL

Recycler::DoCollectWrapped(CollectionFlags flags)

{

#ifdef CONCURRENT\_GC\_ENABLED

this->skipStack = ((flags & CollectOverride\_SkipStack) != 0);

DebugOnly(this->isConcurrentGCOnIdle = (flags == CollectOnScriptIdle));

#endif

this->allowDispose = (flags & CollectOverride\_AllowDispose) == CollectOverride\_AllowDispose;

BOOL collected = collectionWrapper->ExecuteRecyclerCollectionFunction(this, &Recycler::DoCollect, flags);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(IsConcurrentExecutingState() || IsConcurrentFinishedState() || !CollectionInProgress());

#else

Assert(!CollectionInProgress());

#endif

return collected;

}

bool

Recycler::NeedExhaustiveRepeatCollect() const

{

return this->inExhaustiveCollection && this->hasExhaustiveCandidate;

}

BOOL

Recycler::DoCollect(CollectionFlags flags)

{

// ExecuteRecyclerCollectionFunction may cause exception. In which case, we may trigger the assert

// in SetupPostCollectionFlags because we didn't reset the inExhausitvECollection variable if

// an exception. We are not in DoCollect, there shouldn't be any more exception. Reset the flag

DebugOnly(this->hasIncompletedDoCollect = false);

#ifdef RECYCLER\_MEMORY\_VERIFY

this->Verify(Js::RecyclerPhase);

#endif

#ifdef RECYCLER\_FINALIZE\_CHECK

autoHeap.VerifyFinalize();

#endif

#ifdef PARTIAL\_GC\_ENABLED

BOOL partial = flags & CollectMode\_Partial;

#if DBG && defined(RECYCLER\_DUMP\_OBJECT\_GRAPH)

// Can't pass in RecyclerPartialStress and DumpObjectGraphOnCollect or call CollectGarbage with DumpObjectGraph

if (GetRecyclerFlagsTable().RecyclerPartialStress) {

Assert(!GetRecyclerFlagsTable().DumpObjectGraphOnCollect && !this->dumpObjectOnceOnCollect);

} else if (GetRecyclerFlagsTable().DumpObjectGraphOnCollect || this->dumpObjectOnceOnCollect) {

Assert(!GetRecyclerFlagsTable().RecyclerPartialStress);

}

#endif

#ifdef RECYCLER\_STRESS

if (partial && GetRecyclerFlagsTable().RecyclerPartialStress)

{

this->inPartialCollectMode = true;

this->forcePartialScanStack = true;

}

#endif

#endif

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

if (dumpObjectOnceOnCollect || GetRecyclerFlagsTable().DumpObjectGraphOnCollect)

{

DumpObjectGraph();

dumpObjectOnceOnCollect = false;

// Can't do a partial collect if DumpObjectGraph is set since it'll call FinishPartial

// which will set inPartialCollectMode to false.

partial = false;

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

const bool concurrent = (flags & CollectMode\_Concurrent) != 0;

const BOOL forceInThread = flags & CollectOverride\_ForceInThread;

#endif

// Flush the pending dispose objects first if dispose is allowed

Assert(!this->CollectionInProgress());

Assert(this->backgroundFinishMarkCount == 0);

bool collected = FinishDisposeObjects();

do

{

INC\_TIMESTAMP\_FIELD(exhaustiveRepeatedCount);

RECORD\_TIMESTAMP(currentCollectionStartTime);

this->telemetryBlock->currentCollectionStartProcessUsedBytes = PageAllocator::GetProcessUsedBytes();

#ifdef CONCURRENT\_GC\_ENABLED

// DisposeObject may call script again and start another GC, so we may still be in concurrent GC state

if (this->CollectionInProgress())

{

Assert(this->IsConcurrentState());

Assert(collected);

if (forceInThread)

{

return this->FinishConcurrentCollect(flags);

}

return true;

}

#endif

Assert(this->backgroundFinishMarkCount == 0);

#if DBG

collectionCount++;

#endif

collectionState = Collection\_PreCollection;

collectionWrapper->PreCollectionCallBack(flags);

collectionState = CollectionStateNotCollecting;

hasExhaustiveCandidate = false; // reset the candidate detection

#ifdef RECYCLER\_STATS

#ifdef PARTIAL\_GC\_ENABLED

RecyclerCollectionStats oldCollectionStats = collectionStats;

#endif

memset(&collectionStats, 0, sizeof(RecyclerCollectionStats));

this->collectionStats.startCollectAllocBytes = autoHeap.uncollectedAllocBytes;

#ifdef PARTIAL\_GC\_ENABLED

this->collectionStats.startCollectNewPageCount = autoHeap.uncollectedNewPageCount;

this->collectionStats.uncollectedNewPageCountPartialCollect = this->uncollectedNewPageCountPartialCollect;

#endif

#endif

#ifdef PARTIAL\_GC\_ENABLED

if (partial)

{

#ifdef CONCURRENT\_GC\_ENABLED

Assert(!forceInThread);

#endif

#ifdef RECYCLER\_STATS

// We are only doing a partial GC, copy some old stats

collectionStats.finalizeCount = oldCollectionStats.finalizeCount;

memcpy(collectionStats.heapBlockCount, oldCollectionStats.smallNonLeafHeapBlockPartialUnusedCount,

sizeof(oldCollectionStats.smallNonLeafHeapBlockPartialUnusedCount));

memcpy(collectionStats.heapBlockFreeByteCount, oldCollectionStats.smallNonLeafHeapBlockPartialUnusedBytes,

sizeof(oldCollectionStats.smallNonLeafHeapBlockPartialUnusedBytes));

memcpy(collectionStats.smallNonLeafHeapBlockPartialUnusedCount, oldCollectionStats.smallNonLeafHeapBlockPartialUnusedCount,

sizeof(oldCollectionStats.smallNonLeafHeapBlockPartialUnusedCount));

memcpy(collectionStats.smallNonLeafHeapBlockPartialUnusedBytes, oldCollectionStats.smallNonLeafHeapBlockPartialUnusedBytes,

sizeof(oldCollectionStats.smallNonLeafHeapBlockPartialUnusedBytes));

#endif

Assert(enablePartialCollect && inPartialCollectMode);

if (!this->PartialCollect(concurrent))

{

return collected;

}

// This disable partial if we do a repeated exhaustive GC

partial = false;

collected = true;

continue;

}

// Not doing partial collect, we should decommit on finish collect

decommitOnFinish = true;

if (inPartialCollectMode)

{

// finish the partial collect first

FinishPartialCollect();

// Old heap block with free object is made available, count that as being collected

collected = true;

// PARTIAL-GC-CONSIDER: should we just pretend we did a GC, since we have made the free listed object

// available to be used, instead of starting off another GC?

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

bool skipConcurrent = false;

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

// If the below flag is passed in, skip doing a non-blocking concurrent collect. Instead,

// we will do a blocking concurrent collect, which is basically an in-thread GC

skipConcurrent = GetRecyclerFlagsTable().ForceBlockingConcurrentCollect;

#endif

// We are about to start a collection. Reset our heuristic counters now, so that

// any allocations that occur during concurrent collection count toward the next collection's threshold.

ResetHeuristicCounters();

if (concurrent && !skipConcurrent)

{

Assert(!forceInThread);

if (enableConcurrentMark)

{

if (StartBackgroundMarkCollect())

{

// Tell the caller whether we have finish a collection and there maybe free object to reuse

return collected;

}

// Either ResetWriteWatch failed or the thread service failed

// So concurrent mark is disabled, at least for now

}

if (enableConcurrentSweep)

{

if (StartConcurrentSweepCollect())

{

collected = true;

continue;

}

// out of memory during collection

return collected;

}

// concurrent collection failed, default back to non-concurrent collection

}

if (!forceInThread && enableConcurrentMark)

{

if (!CollectOnConcurrentThread())

{

// time out or out of memory during collection

return collected;

}

}

else

#endif

{

if (!CollectOnAllocatorThread())

{

// out of memory during collection

return collected;

}

}

collected = true;

#ifdef RECYCLER\_TRACE

collectionParam.repeat = true;

#endif

}

while (this->NeedExhaustiveRepeatCollect());

#ifdef CONCURRENT\_GC\_ENABLED

// DisposeObject may call script again and start another GC, so we may still be in concurrent GC state

if (this->CollectionInProgress())

{

Assert(this->IsConcurrentState());

Assert(collected);

return true;

}

#endif

EndCollection();

// Tell the caller whether we have finish a collection and there maybe free object to reuse

return collected;

}

void

Recycler::EndCollection()

{

Assert(this->backgroundFinishMarkCount == 0);

Assert(!this->CollectionInProgress());

// no more collection is requested, we can turn exhaustive back off

this->inExhaustiveCollection = false;

if (this->inDecommitNowCollection || CUSTOM\_CONFIG\_FLAG(GetRecyclerFlagsTable(), ForceDecommitOnCollect))

{

#ifdef RECYCLER\_TRACE

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::RecyclerPhase))

{

Output::Print(L"%04X> RC(%p): %s\n", this->mainThreadId, this, L"Decommit now");

}

#endif

ForRecyclerPageAllocator(DecommitNow());

this->inDecommitNowCollection = false;

}

RECORD\_TIMESTAMP(lastCollectionEndTime);

}

#ifdef PARTIAL\_GC\_ENABLED

bool

Recycler::PartialCollect(bool concurrent)

{

Assert(IsMarkStackEmpty());

Assert(this->inPartialCollectMode);

Assert(collectionState == CollectionStateNotCollecting);

// Rescan again

collectionState = CollectionStateRescanFindRoots;

if (concurrent && enableConcurrentMark && this->partialConcurrentNextCollection)

{

this->PrepareBackgroundFindRoots();

if (StartConcurrent(CollectionStateConcurrentFinishMark))

{

#ifdef RECYCLER\_TRACE

PrintCollectTrace(Js::ConcurrentPartialCollectPhase);

#endif

return false;

}

this->RevertPrepareBackgroundFindRoots();

}

#ifdef RECYCLER\_STRESS

if (forcePartialScanStack)

{

// Mark the roots since they need not have been marked

// in RecyclerPartialStress mode

this->RootMark(collectionState);

}

#endif

#ifdef RECYCLER\_TRACE

PrintCollectTrace(Js::PartialCollectPhase);

#endif

bool needConcurrentSweep = false;

this->CollectionBegin<Js::PartialCollectPhase>();

size\_t rescanRootBytes = FinishMark(INFINITE);

Assert(rescanRootBytes != Recycler::InvalidScanRootBytes);

needConcurrentSweep = this->Sweep(rescanRootBytes, concurrent, true);

this->CollectionEnd<Js::PartialCollectPhase>();

// Only reset the new page counter

autoHeap.uncollectedNewPageCount = 0;

// Finish collection

FinishCollection(needConcurrentSweep);

return true;

}

void

Recycler::ProcessClientTrackedObjects()

{

GCETW(GC\_PROCESS\_CLIENT\_TRACKED\_OBJECT\_START, (this));

Assert(this->inPartialCollectMode);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(!this->DoQueueTrackedObject());

#endif

if (!this->clientTrackedObjectList.Empty())

{

SListBase<void \*>::Iterator iter(&this->clientTrackedObjectList);

while (iter.Next())

{

auto& reference = iter.Data();

this->TryMarkNonInterior(reference, &reference /\* parentReference \*/); // Reference to inside the node

RECYCLER\_STATS\_INC(this, clientTrackedObjectCount);

}

this->clientTrackedObjectList.Clear(&this->clientTrackedObjectAllocator);

}

GCETW(GC\_PROCESS\_CLIENT\_TRACKED\_OBJECT\_STOP, (this));

}

void

Recycler::ClearPartialCollect()

{

Assert(!this->DoQueueTrackedObject());

this->autoHeap.unusedPartialCollectFreeBytes = 0;

this->partialUncollectedAllocBytes = 0;

this->clientTrackedObjectList.Clear(&this->clientTrackedObjectAllocator);

this->uncollectedNewPageCountPartialCollect = (size\_t)-1;

}

void

Recycler::FinishPartialCollect(RecyclerSweep \* recyclerSweep)

{

Assert(recyclerSweep == nullptr || !recyclerSweep->IsBackground());

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::FinishPartialPhase);

Assert(inPartialCollectMode);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(!this->DoQueueTrackedObject());

#endif

autoHeap.FinishPartialCollect(recyclerSweep);

this->inPartialCollectMode = false;

ClearPartialCollect();

RECYCLER\_PROFILE\_EXEC\_END(this, Js::FinishPartialPhase);

}

#endif

void

Recycler::EnsureNotCollecting()

{

#ifdef CONCURRENT\_GC\_ENABLED

FinishConcurrent<ForceFinishCollection>();

#endif

Assert(!this->CollectionInProgress());

}

void Recycler::EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size))

{

// Make sure we are not collecting

EnsureNotCollecting();

#ifdef PARTIAL\_GC\_ENABLED

// We are updating the free bit vector, messing up the partial collection state.

// Just get out of partial collect mode

// GC-CONSIDER: consider adding an option in FinishConcurrent to not get into partial collect mode during sweep.

if (inPartialCollectMode)

{

FinishPartialCollect();

}

#endif

autoHeap.EnumerateObjects(infoBits, CallBackFunction);

// GC-TODO: Explicit heap?

}

BOOL

Recycler::IsMarkState() const

{

return (collectionState & Collection\_Mark);

}

BOOL

Recycler::IsFindRootsState() const

{

return (collectionState & Collection\_FindRoots);

}

#if DBG

BOOL

Recycler::IsReentrantState() const

{

#ifdef CONCURRENT\_GC\_ENABLED

return !this->CollectionInProgress() || this->IsConcurrentState();

#else

return !this->CollectionInProgress();

#endif

}

#endif

#ifdef ENABLE\_JS\_ETW

template <Js::Phase phase> static ETWEventGCActivationKind GetETWEventGCActivationKind();

template <> static ETWEventGCActivationKind GetETWEventGCActivationKind<Js::GarbageCollectPhase>() { return ETWEvent\_GarbageCollect; }

template <> static ETWEventGCActivationKind GetETWEventGCActivationKind<Js::ThreadCollectPhase>() { return ETWEvent\_ThreadCollect; }

template <> static ETWEventGCActivationKind GetETWEventGCActivationKind<Js::ConcurrentCollectPhase>() { return ETWEvent\_ConcurrentCollect; }

template <> static ETWEventGCActivationKind GetETWEventGCActivationKind<Js::PartialCollectPhase>() { return ETWEvent\_PartialCollect; }

#endif

template <Js::Phase phase>

void

Recycler::CollectionBegin()

{

RECYCLER\_PROFILE\_EXEC\_BEGIN2(this, Js::RecyclerPhase, phase);

GCETW(GC\_START, (this, GetETWEventGCActivationKind<phase>()));

#ifdef ENABLE\_BASIC\_TELEMETRY

if (this->IsMemProtectMode() == false)

{

gcTel.LogGCPauseStartTime();

}

#endif

}

template <Js::Phase phase>

void

Recycler::CollectionEnd()

{

GCETW(GC\_STOP, (this, GetETWEventGCActivationKind<phase>()));

#ifdef ENABLE\_BASIC\_TELEMETRY

if (GetCurrentThreadContextId() == mainThreadId && IsMemProtectMode() == false)

{

gcTel.LogGCPauseEndTime();

}

#endif

RECYCLER\_PROFILE\_EXEC\_END2(this, phase, Js::RecyclerPhase);

}

#ifdef CONCURRENT\_GC\_ENABLED

size\_t

Recycler::BackgroundRescan(RescanFlags rescanFlags)

{

Assert(!this->isProcessingRescan);

DebugOnly(this->isProcessingRescan = true);

GCETW(GC\_BACKGROUNDRESCAN\_START, (this, backgroundRescanCount));

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(this, Js::BackgroundRescanPhase);

size\_t rescannedPageCount = heapBlockMap.Rescan(this, ((rescanFlags & RescanFlags\_ResetWriteWatch) != 0));

rescannedPageCount += autoHeap.Rescan(rescanFlags);

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, Js::BackgroundRescanPhase);

GCETW(GC\_BACKGROUNDRESCAN\_STOP, (this, backgroundRescanCount));

this->backgroundRescanCount++;

if (!this->NeedOOMRescan())

{

if ((rescanFlags & RescanFlags\_ResetWriteWatch) != 0)

{

DebugOnly(this->isProcessingRescan = false);

}

return rescannedPageCount;

}

DebugOnly(this->isProcessingRescan = false);

return Recycler::InvalidScanRootBytes;

}

void

Recycler::BackgroundResetWriteWatchAll()

{

GCETW(GC\_BACKGROUNDRESETWRITEWATCH\_START, (this, -1));

heapBlockMap.ResetWriteWatch(this);

GCETW(GC\_BACKGROUNDRESETWRITEWATCH\_STOP, (this, -1));

}

#endif

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t

Recycler::FinishMarkRescan(bool background)

{

if (background)

{

GCETW(GC\_BACKGROUNDRESCAN\_START, (this, 0));

}

else

{

GCETW(GC\_RESCAN\_START, (this));

}

RECYCLER\_PROFILE\_EXEC\_THREAD\_BEGIN(background, this, Js::RescanPhase);

#ifdef CONCURRENT\_GC\_ENABLED

RescanFlags const flags = (background ? RescanFlags\_ResetWriteWatch : RescanFlags\_None);

#else

Assert(!background);

RescanFlags const flags = RescanFlags\_None;

#endif

#if DBG

Assert(!this->isProcessingRescan);

this->isProcessingRescan = true;

#endif

size\_t scannedPageCount = heapBlockMap.Rescan(this, ((flags & RescanFlags\_ResetWriteWatch) != 0));

scannedPageCount += autoHeap.Rescan(flags);

DebugOnly(this->isProcessingRescan = false);

RECYCLER\_PROFILE\_EXEC\_THREAD\_END(background, this, Js::RescanPhase);

if (background)

{

GCETW(GC\_BACKGROUNDRESCAN\_STOP, (this, 0));

}

else

{

GCETW(GC\_RESCAN\_STOP, (this));

}

return scannedPageCount;

}

void

Recycler::ProcessTrackedObjects()

{

GCETW(GC\_PROCESS\_TRACKED\_OBJECT\_START, (this));

#ifdef PARTIAL\_GC\_ENABLED

Assert(this->clientTrackedObjectList.Empty());

Assert(!this->inPartialCollectMode);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

Assert(this->DoQueueTrackedObject());

this->queueTrackedObject = false;

DebugOnly(this->isProcessingTrackedObjects = true);

markContext.ProcessTracked();

// If we did a parallel mark, we need to process any queued tracked objects from the parallel mark stack as well.

// If we didn't, this will do nothing.

parallelMarkContext1.ProcessTracked();

parallelMarkContext2.ProcessTracked();

parallelMarkContext3.ProcessTracked();

DebugOnly(this->isProcessingTrackedObjects = false);

#endif

GCETW(GC\_PROCESS\_TRACKED\_OBJECT\_STOP, (this));

}

#endif // defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

BOOL

Recycler::RequestConcurrentWrapperCallback()

{

#ifdef CONCURRENT\_GC\_ENABLED

Assert(!IsConcurrentExecutingState());

// Save the original collection state

CollectionState oldState = this->collectionState;

// Get the background thread to start the callback

if (StartConcurrent(CollectionStateConcurrentWrapperCallback))

{

// Wait for the callback to complete

WaitForConcurrentThread(INFINITE);

// The state must not change back until we restore the original state

Assert(collectionState == CollectionStateConcurrentWrapperCallback);

this->collectionState = oldState;

return true;

}

#endif

return false;

}

#ifdef CONCURRENT\_GC\_ENABLED

/\*------------------------------------------------------------------------------------------------

\* Concurrent

\*------------------------------------------------------------------------------------------------\*/

BOOL

Recycler::CollectOnConcurrentThread()

{

#ifdef PARTIAL\_GC\_ENABLED

Assert(!inPartialCollectMode);

#endif

#ifdef RECYCLER\_TRACE

PrintCollectTrace(Js::ThreadCollectPhase);

#endif

this->CollectionBegin<Js::ThreadCollectPhase>();

// Synchronous concurrent mark

if (!StartSynchronousBackgroundMark())

{

this->CollectionEnd<Js::ThreadCollectPhase>();

return false;

}

const DWORD waitTime = RecyclerHeuristic::FinishConcurrentCollectWaitTime(this->GetRecyclerFlagsTable());

GCETW(GC\_SYNCHRONOUSMARKWAIT\_START, (this, waitTime));

const BOOL waited = WaitForConcurrentThread(waitTime);

GCETW(GC\_SYNCHRONOUSMARKWAIT\_STOP, (this, !waited));

if (!waited)

{

#ifdef RECYCLER\_TRACE

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::RecyclerPhase)

|| GetRecyclerFlagsTable().Trace.IsEnabled(Js::ThreadCollectPhase))

{

Output::Print(L"%04X> RC(%p): %s: %s\n", this->mainThreadId, this, Js::PhaseNames[Js::ThreadCollectPhase], L"Timeout");

}

#endif

this->CollectionEnd<Js::ThreadCollectPhase>();

return false;

}

// If the concurrent thread was done within the time limit, there shouldn't be

// any object needs to be rescanned

// CONCURRENT-TODO: Optimize it so we don't rescan in the background if we are still waiting

// GC-TODO: Unfortunately we can't assert this, as the background code gen thread may still

// touch GC memory (e.g. FunctionBody), causing write watch and rescan

// in the background.

// Assert(markContext.Empty());

DebugOnly(this->isProcessingRescan = false);

this->collectionState = CollectionStateMark;

this->ProcessTrackedObjects();

this->ProcessMark(false);

this->EndMark();

// Partial collect mode is not re-enabled after a non-partial in-thread GC because partial GC heuristics are not adjusted

// after a full in-thread GC. Enabling partial collect mode causes partial GC heuristics to be reset before the next full

// in-thread GC, thereby allowing partial GC to kick in more easily without being able to adjust heuristics after the full

// GCs. Until we have a way of adjusting partial GC heuristics after a full in-thread GC, once partial collect mode is

// turned off, it will remain off until a concurrent GC happens

this->Sweep();

this->CollectionEnd<Js::ThreadCollectPhase>();

FinishCollection();

return true;

}

// explicit instantiation

template BOOL Recycler::FinishConcurrent<FinishConcurrentOnIdle>();

template BOOL Recycler::FinishConcurrent<FinishConcurrentOnIdleAtRoot>();

template BOOL Recycler::FinishConcurrent<FinishConcurrentOnExitScript>();

template BOOL Recycler::FinishConcurrent<FinishConcurrentOnEnterScript>();

template BOOL Recycler::FinishConcurrent<ForceFinishCollection>();

template <CollectionFlags flags>

BOOL

Recycler::FinishConcurrent()

{

CompileAssert((flags & ~(CollectOverride\_AllowDispose | CollectOverride\_ForceFinish | CollectOverride\_ForceInThread

| CollectMode\_Concurrent | CollectOverride\_DisableIdleFinish | CollectOverride\_BackgroundFinishMark

| CollectOverride\_SkipStack | CollectOverride\_FinishConcurrentTimeout)) == 0);

if (this->CollectionInProgress())

{

Assert(this->IsConcurrentEnabled());

Assert(IsConcurrentState());

const BOOL forceFinish = flags & CollectOverride\_ForceFinish;

if (forceFinish || !IsConcurrentExecutingState())

{

if (this->collectionState == CollectionStateConcurrentSweep)

{

// Help with the background thread to zero and flush zero pages

// if we are going to wait anyways.

recyclerPageAllocator.ZeroQueuedPages();

recyclerLargeBlockPageAllocator.ZeroQueuedPages();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.ZeroQueuedPages();

#endif

this->FlushBackgroundPages();

}

#ifdef RECYCLER\_TRACE

collectionParam.finishOnly = true;

collectionParam.flags = flags;

#endif

#ifdef CONCURRENT\_GC\_ENABLED

// If SkipStack is provided, and we're not forcing the finish (i.e we're not in concurrent executing state)

// then, it's fine to set the skipStack flag to true, so that during the in-thread find-roots, we'll skip

// the stack scan

this->skipStack = ((flags & CollectOverride\_SkipStack) != 0) && !forceFinish;

#if DBG

this->isFinishGCOnIdle = (flags == FinishConcurrentOnIdleAtRoot);

#endif

#endif

return FinishConcurrentCollectWrapped(flags);

}

}

return false;

}

template <CollectionFlags flags>

BOOL

Recycler::TryFinishConcurrentCollect()

{

Assert(this->CollectionInProgress());

RECYCLER\_STATS\_INC(this, finishCollectTryCount);

SetupPostCollectionFlags<flags>();

const BOOL concurrent = flags & CollectMode\_Concurrent;

const BOOL forceInThread = flags & CollectOverride\_ForceInThread;

Assert(this->IsConcurrentEnabled());

Assert(IsConcurrentState() || IsCollectionDisabled());

Assert(!concurrent || !forceInThread);

if (concurrent && concurrentThread != NULL)

{

if (IsConcurrentExecutingState())

{

if (!this->priorityBoost)

{

uint tickCount = GetTickCount();

if ((autoHeap.uncollectedAllocBytes > RecyclerHeuristic::Instance.UncollectedAllocBytesConcurrentPriorityBoost)

|| (tickCount - this->tickCountStartConcurrent > RecyclerHeuristic::PriorityBoostTimeout(this->GetRecyclerFlagsTable())))

{

#ifdef RECYCLER\_TRACE

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::RecyclerPhase))

{

Output::Print(L"%04X> RC(%p): %s: ", this->mainThreadId, this, L"Set priority normal");

if (autoHeap.uncollectedAllocBytes > RecyclerHeuristic::Instance.UncollectedAllocBytesConcurrentPriorityBoost)

{

Output::Print(L"AllocBytes=%d (Time=%d)\n", autoHeap.uncollectedAllocBytes, tickCount - this->tickCountStartConcurrent);

}

else

{

Output::Print(L"Time=%d (AllocBytes=%d\n", tickCount - this->tickCountStartConcurrent, autoHeap.uncollectedAllocBytes);

}

}

#endif

// Set it to a large number so we don't set the thread priority again

this->priorityBoost = true;

// The recycler thread hasn't come back in 5 seconds

// It either has a large object graph, or it is starving.

// Set the priority back to normal

SetThreadPriority(this->concurrentThread, THREAD\_PRIORITY\_NORMAL);

}

}

return FinishDisposeObjectsWrapped<flags>();

}

else if ((flags & CollectOverride\_FinishConcurrentTimeout) != 0)

{

uint tickCount = GetTickCount();

// If we haven't gone past the time to call finish collection,

// simply call FinishDisposeObjects and return

// Otherwise, actually go ahead and call FinishConcurrentCollectWrapped

// We do this only if this is a collection that allows finish concurrent to timeout

// If not, by default, we finish the collection

if (tickCount <= this->tickCountNextFinishCollection)

{

return FinishDisposeObjectsWrapped<flags>();

}

}

}

return FinishConcurrentCollectWrapped(flags);

}

BOOL

Recycler::IsConcurrentMarkState() const

{

return (collectionState & Collection\_ConcurrentMark) == Collection\_ConcurrentMark;

}

BOOL

Recycler::IsConcurrentMarkExecutingState() const

{

return (collectionState & (Collection\_ConcurrentMark | Collection\_ExecutingConcurrent)) == (Collection\_ConcurrentMark | Collection\_ExecutingConcurrent);

}

BOOL

Recycler::IsConcurrentResetMarksState() const

{

return collectionState == CollectionStateConcurrentResetMarks;

}

BOOL

Recycler::IsInThreadFindRootsState() const

{

CollectionState currentCollectionState = collectionState;

return (currentCollectionState & Collection\_FindRoots) && (currentCollectionState != CollectionStateConcurrentFindRoots);

}

BOOL

Recycler::IsConcurrentFindRootState() const

{

return collectionState == CollectionStateConcurrentFindRoots;

}

BOOL

Recycler::IsConcurrentExecutingState() const

{

return (collectionState & Collection\_ExecutingConcurrent);

}

BOOL

Recycler::IsConcurrentSweepExecutingState() const

{

return (collectionState & (Collection\_ConcurrentSweep | Collection\_ExecutingConcurrent)) == (Collection\_ConcurrentSweep | Collection\_ExecutingConcurrent);

}

BOOL

Recycler::IsConcurrentState() const

{

return (collectionState & Collection\_Concurrent);

}

#if DBG

BOOL

Recycler::IsConcurrentFinishedState() const

{

return (collectionState & Collection\_FinishConcurrent);

}

#endif

bool

Recycler::InitializeConcurrent(JsUtil::ThreadService \*threadService)

{

try

{

AUTO\_NESTED\_HANDLED\_EXCEPTION\_TYPE(ExceptionType\_OutOfMemory);

concurrentWorkDoneEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

if (concurrentWorkDoneEvent == nullptr)

{

throw Js::OutOfMemoryException();

}

#if DBG\_DUMP

markContext.GetPageAllocator()->debugName = L"ConcurrentCollect";

#endif

if (!threadService->HasCallback())

{

#ifdef IDLE\_DECOMMIT\_ENABLED

concurrentIdleDecommitEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

if (concurrentIdleDecommitEvent == nullptr)

{

throw Js::OutOfMemoryException();

}

#endif

concurrentWorkReadyEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

if (concurrentWorkReadyEvent == nullptr)

{

throw Js::OutOfMemoryException();

}

}

}

catch (Js::OutOfMemoryException)

{

Assert(concurrentWorkReadyEvent == nullptr);

if (concurrentWorkDoneEvent)

{

CloseHandle(concurrentWorkDoneEvent);

concurrentWorkDoneEvent = nullptr;

}

#ifdef IDLE\_DECOMMIT\_ENABLED

if (concurrentIdleDecommitEvent)

{

CloseHandle(concurrentIdleDecommitEvent);

concurrentIdleDecommitEvent = nullptr;

}

#endif

return false;

}

return true;

}

#pragma prefast(suppress:6262, "Where this function is call should have ample of stack space")

bool Recycler::AbortConcurrent(bool restoreState)

{

Assert(!this->CollectionInProgress() || this->IsConcurrentState());

// In case the thread already died, wait for that too

HANDLE handle[2] = { concurrentWorkDoneEvent, concurrentThread };

// Note, concurrentThread will be null if we have a threadService.

Assert(concurrentThread != NULL || threadService->HasCallback());

DWORD handleCount = (concurrentThread == NULL ? 1 : 2);

DWORD ret = WAIT\_OBJECT\_0;

if (this->IsConcurrentState())

{

this->isAborting = true;

if (this->concurrentThread != NULL)

{

SetThreadPriority(this->concurrentThread, THREAD\_PRIORITY\_NORMAL);

}

ret = WaitForMultipleObjectsEx(handleCount, handle, FALSE, INFINITE, FALSE);

this->isAborting = false;

Assert(this->IsConcurrentFinishedState() || ret == WAIT\_OBJECT\_0 + 1);

if (ret == WAIT\_OBJECT\_0 && restoreState)

{

if (collectionState == CollectionStateRescanWait)

{

this->ResetMarkCollectionState();

}

else if (collectionState == CollectionStateTransferSweptWait)

{

// Make sure we don't do another GC after finishing this one.

this->inExhaustiveCollection = false;

// Let's just finish the sweep so that GC is in a consistent state, but don't run dispose

// AbortConcurrent already consumed the event from the concurrent thread, just signal it so

// FinishConcurrentCollect can wait for it again.

SetEvent(this->concurrentWorkDoneEvent);

EnsureNotCollecting();

}

else

{

Assert(UNREACHED);

}

Assert(collectionState == CollectionStateNotCollecting);

Assert(this->isProcessingRescan == false);

}

else

{

// Even if we weren't asked to restore states, we need to clean up the pending guest arena

CleanupPendingUnroot();

// Also need to release any pages held by the mark stack, if we abandoned it

markContext.Abort();

}

}

Assert(!this->hasPendingDeleteGuestArena);

return ret == WAIT\_OBJECT\_0;

}

void

Recycler::CleanupPendingUnroot()

{

Assert(!this->hasPendingConcurrentFindRoot);

if (hasPendingUnpinnedObject)

{

pinnedObjectMap.MapAndRemoveIf([](void \* obj, PinRecord const &refCount)

{

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

Assert(refCount != 0 || refCount.stackBackTraces == nullptr);

#endif

return refCount == 0;

});

hasPendingUnpinnedObject = false;

}

if (hasPendingDeleteGuestArena)

{

DebugOnly(bool foundPendingDelete = false);

DListBase<GuestArenaAllocator>::EditingIterator guestArenaIter(&guestArenaList);

while (guestArenaIter.Next())

{

GuestArenaAllocator& allocator = guestArenaIter.Data();

if (allocator.pendingDelete)

{

allocator.SetLockBlockList(false);

guestArenaIter.RemoveCurrent(&HeapAllocator::Instance);

DebugOnly(foundPendingDelete = true);

}

}

hasPendingDeleteGuestArena = false;

Assert(foundPendingDelete);

}

#if DBG

else

{

DListBase<GuestArenaAllocator>::Iterator guestArenaIter(&guestArenaList);

while (guestArenaIter.Next())

{

GuestArenaAllocator& allocator = guestArenaIter.Data();

Assert(!allocator.pendingDelete);

}

}

#endif

}

void

Recycler::FinalizeConcurrent(bool restoreState)

{

bool needCleanExitState = restoreState;

#if defined(RECYCLER\_DUMP\_OBJECT\_GRAPH)

needCleanExitState = needCleanExitState || GetRecyclerFlagsTable().DumpObjectGraphOnExit;

#endif

#ifdef LEAK\_REPORT

needCleanExitState = needCleanExitState || GetRecyclerFlagsTable().IsEnabled(Js::LeakReportFlag);

#endif

#ifdef CHECK\_MEMORY\_LEAK

needCleanExitState = needCleanExitState || GetRecyclerFlagsTable().CheckMemoryLeak;

#endif

bool aborted = AbortConcurrent(needCleanExitState);

collectionState = CollectionStateExit;

if (aborted && this->concurrentThread != NULL)

{

// In case the thread already died, wait for that too

HANDLE handle[2] = { concurrentWorkDoneEvent, concurrentThread };

SetEvent(concurrentWorkReadyEvent);

SetThreadPriority(this->concurrentThread, THREAD\_PRIORITY\_NORMAL);

// In case the thread already died, wait for that too

DWORD fRet = WaitForMultipleObjectsEx(2, handle, FALSE, INFINITE, FALSE);

AssertMsg(fRet != WAIT\_FAILED, "Check handles passed to WaitForMultipleObjectsEx.");

}

// Shutdown parallel threads and return the handle for them so the caller can

// close it.

parallelThread1.Shutdown();

parallelThread2.Shutdown();

#ifdef IDLE\_DECOMMIT\_ENABLED

if (concurrentIdleDecommitEvent != nullptr)

{

CloseHandle(concurrentIdleDecommitEvent);

concurrentIdleDecommitEvent = nullptr;

}

#endif

CloseHandle(concurrentWorkDoneEvent);

concurrentWorkDoneEvent = nullptr;

if (concurrentWorkReadyEvent != NULL)

{

CloseHandle(concurrentWorkReadyEvent);

concurrentWorkReadyEvent = nullptr;

}

if (needCleanExitState)

{

// We may do another marking pass to look for memory leaks;

// Since we have shut down the concurrent thread, don't do a parallel mark.

this->enableConcurrentMark = false;

this->enableParallelMark = false;

this->enableConcurrentSweep = false;

}

this->threadService = nullptr;

this->concurrentThread = nullptr;

}

bool

Recycler::EnableConcurrent(JsUtil::ThreadService \*threadService, bool startAllThreads)

{

if (this->disableConcurrent)

{

return false;

}

if (!this->InitializeConcurrent(threadService))

{

return false;

}

#if ENABLE\_DEBUG\_CONFIG\_OPTIONS

this->enableConcurrentMark = !CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::ConcurrentMarkPhase);

this->enableParallelMark = !CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::ParallelMarkPhase);

this->enableConcurrentSweep = !CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::ConcurrentSweepPhase);

#else

this->enableConcurrentMark = true;

this->enableParallelMark = true;

this->enableConcurrentSweep = true;

#endif

if (this->enableParallelMark && this->maxParallelism == 1)

{

// Disable parallel mark if only 1 CPU

this->enableParallelMark = false;

}

if (threadService->HasCallback())

{

this->threadService = threadService;

return true;

}

else

{

bool startConcurrentThread = true;

bool startedParallelThread1 = false;

bool startedParallelThread2 = false;

if (startAllThreads)

{

if (this->enableParallelMark && this->maxParallelism > 2)

{

if (!parallelThread1.EnableConcurrent(true))

{

startConcurrentThread = false;

}

else

{

startedParallelThread1 = true;

if (this->maxParallelism > 3)

{

if (!parallelThread2.EnableConcurrent(true))

{

startConcurrentThread = false;

}

else

{

startedParallelThread2 = true;

}

}

}

}

}

if (startConcurrentThread)

{

HANDLE concurrentThread = (HANDLE)\_beginthreadex(NULL, Recycler::ConcurrentThreadStackSize, &Recycler::StaticThreadProc, this, STACK\_SIZE\_PARAM\_IS\_A\_RESERVATION, NULL);

if (concurrentThread != nullptr)

{

// Wait for recycler thread to initialize

HANDLE handle[2] = { this->concurrentWorkDoneEvent, concurrentThread };

DWORD ret = WaitForMultipleObjectsEx(2, handle, FALSE, INFINITE, FALSE);

if (ret == WAIT\_OBJECT\_0)

{

this->threadService = threadService;

this->concurrentThread = concurrentThread;

return true;

}

CloseHandle(concurrentThread);

}

}

if (startedParallelThread1)

{

parallelThread1.Shutdown();

if (startedParallelThread2)

{

parallelThread2.Shutdown();

}

}

}

// We failed to start a concurrent thread so we set these back to false and clean up

this->enableConcurrentMark = false;

this->enableParallelMark = false;

this->enableConcurrentSweep = false;

if (concurrentWorkReadyEvent)

{

CloseHandle(concurrentWorkReadyEvent);

concurrentWorkReadyEvent = nullptr;

}

if (concurrentWorkDoneEvent)

{

CloseHandle(concurrentWorkDoneEvent);

concurrentWorkDoneEvent = nullptr;

}

#ifdef IDLE\_DECOMMIT\_ENABLED

if (concurrentIdleDecommitEvent)

{

CloseHandle(concurrentIdleDecommitEvent);

concurrentIdleDecommitEvent = nullptr;

}

#endif

return false;

}

void

Recycler::ShutdownThread()

{

if (this->IsConcurrentEnabled())

{

Assert(concurrentThread != NULL || threadService->HasCallback());

FinalizeConcurrent(false);

if (concurrentThread)

{

CloseHandle(concurrentThread);

}

}

}

void

Recycler::DisableConcurrent()

{

if (this->IsConcurrentEnabled())

{

Assert(concurrentThread != NULL || threadService->HasCallback());

FinalizeConcurrent(true);

if (concurrentThread)

{

CloseHandle(concurrentThread);

}

this->collectionState = CollectionStateNotCollecting;

}

}

bool

Recycler::StartConcurrent(CollectionState const state)

{

// Reset the tick count to detect if the concurrent thread is taking too long

tickCountStartConcurrent = GetTickCount();

CollectionState oldState = this->collectionState;

this->collectionState = state;

if (threadService->HasCallback())

{

Assert(concurrentThread == NULL);

Assert(concurrentWorkReadyEvent == NULL);

if (!threadService->Invoke(Recycler::StaticBackgroundWorkCallback, this))

{

this->collectionState = oldState;

return false;

}

return true;

}

else

{

Assert(concurrentThread != NULL);

Assert(concurrentWorkReadyEvent != NULL);

SetEvent(concurrentWorkReadyEvent);

return true;

}

}

BOOL

Recycler::StartBackgroundMarkCollect()

{

#ifdef RECYCLER\_TRACE

PrintCollectTrace(Js::ConcurrentMarkPhase);

#endif

this->CollectionBegin<Js::ConcurrentCollectPhase>();

// Asynchronous concurrent mark

BOOL success = StartAsynchronousBackgroundMark();

this->CollectionEnd<Js::ConcurrentCollectPhase>();

return success;

}

BOOL

Recycler::StartBackgroundMark(bool foregroundResetMark, bool foregroundFindRoots)

{

Assert(!this->CollectionInProgress());

CollectionState backgroundState = CollectionStateConcurrentResetMarks;

bool doBackgroundFindRoots = true;

if (foregroundResetMark || foregroundFindRoots)

{

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::ResetWriteWatchPhase);

bool hasWriteWatch = (recyclerPageAllocator.ResetWriteWatch() && recyclerLargeBlockPageAllocator.ResetWriteWatch());

RECYCLER\_PROFILE\_EXEC\_END(this, Js::ResetWriteWatchPhase);

if (!hasWriteWatch)

{

// Disable concurrent mark

this->enableConcurrentMark = false;

return false;

}

// In-thread synchronized GC on the concurrent thread

ResetMarks(this->enableScanImplicitRoots ? ResetMarkFlags\_SynchronizedImplicitRoots : ResetMarkFlags\_Synchronized);

if (foregroundFindRoots)

{

this->collectionState = CollectionStateFindRoots;

FindRoots();

ScanStack();

Assert(collectionState == CollectionStateFindRoots);

backgroundState = CollectionStateConcurrentMark;

doBackgroundFindRoots = false;

}

else

{

// Do find roots in the background

backgroundState = CollectionStateConcurrentFindRoots;

}

}

if (doBackgroundFindRoots)

{

this->PrepareBackgroundFindRoots();

}

if (!StartConcurrent(backgroundState))

{

if (doBackgroundFindRoots)

{

this->RevertPrepareBackgroundFindRoots();

}

this->collectionState = CollectionStateNotCollecting;

return false;

}

return true;

}

BOOL

Recycler::StartAsynchronousBackgroundMark()

{

// Debug flags to turn off background reset mark or background find roots, default to doing every concurrently

return StartBackgroundMark(CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::BackgroundResetMarksPhase), CUSTOM\_PHASE\_OFF1(GetRecyclerFlagsTable(), Js::BackgroundFindRootsPhase));

}

BOOL

Recycler::StartSynchronousBackgroundMark()

{

return StartBackgroundMark(true, true);

}

BOOL

Recycler::StartConcurrentSweepCollect()

{

Assert(collectionState == CollectionStateNotCollecting);

#ifdef RECYCLER\_TRACE

PrintCollectTrace(Js::ConcurrentSweepPhase);

#endif

this->CollectionBegin<Js::ConcurrentCollectPhase>();

this->Mark();

// We don't have rescan data if we disabled concurrent mark, assume the worst

// (which means it is harder to get into partial collect mode)

bool needConcurrentSweep = this->Sweep(RecyclerSweep::MaxPartialCollectRescanRootBytes, true, true);

this->CollectionEnd<Js::ConcurrentCollectPhase>();

FinishCollection(needConcurrentSweep);

return true;

}

size\_t

Recycler::BackgroundRepeatMark()

{

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(this, Js::BackgroundRepeatMarkPhase);

Assert(this->backgroundRescanCount <= RecyclerHeuristic::MaxBackgroundRepeatMarkCount - 1);

size\_t rescannedPageCount = this->BackgroundRescan(RescanFlags\_ResetWriteWatch);

if (this->NeedOOMRescan() || this->isAborting)

{

// OOM'ed. Let's not continue

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, Js::BackgroundRepeatMarkPhase);

return Recycler::InvalidScanRootBytes;

}

// Rescan the stack

this->BackgroundScanStack();

// Process mark stack

this->DoBackgroundParallelMark();

if (this->NeedOOMRescan())

{

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, Js::BackgroundRepeatMarkPhase);

return Recycler::InvalidScanRootBytes;

}

#ifdef RECYCLER\_STATS

Assert(this->backgroundRescanCount >= 1 && this->backgroundRescanCount <= RecyclerHeuristic::MaxBackgroundRepeatMarkCount);

this->collectionStats.backgroundMarkData[this->backgroundRescanCount - 1] = this->collectionStats.markData;

#endif

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, Js::BackgroundRepeatMarkPhase);

return rescannedPageCount;

}

char\* Recycler::GetScriptThreadStackTop()

{

// We should have already checked if the recycler is thread bound or not

Assert(mainThreadHandle != NULL);

return (char\*) savedThreadContext.GetStackTop();

}

size\_t

Recycler::BackgroundScanStack()

{

if (this->skipStack)

{

#ifdef RECYCLER\_TRACE

CUSTOM\_PHASE\_PRINT\_VERBOSE\_TRACE1(GetRecyclerFlagsTable(), Js::ScanStackPhase, L"[%04X] Skipping the stack scan\n", ::GetCurrentThreadId());

#endif

Output::Print(Js::ScanStackPhase, L"[%04X] Skipping the stack scan\n", ::GetCurrentThreadId());

return 0;

}

if (!this->isInScript || mainThreadHandle == nullptr)

{

// No point in scanning the main thread's stack if we are not in script

// We also can't scan the main thread's stack if we are not thread bounded, and didn't create the main thread's handle

return 0;

}

char\* stackTop = this->GetScriptThreadStackTop();

if (stackTop != nullptr)

{

size\_t size = (char \*)stackBase - stackTop;

ScanMemoryInline((void \*\*)stackTop, size);

return size;

}

return 0;

}

void

Recycler::BackgroundMark()

{

Assert(this->DoQueueTrackedObject());

this->backgroundRescanCount = 0;

this->DoBackgroundParallelMark();

if (this->NeedOOMRescan() || this->isAborting)

{

return;

}

#ifdef RECYCLER\_STATS

this->collectionStats.backgroundMarkData[0] = this->collectionStats.markData;

#endif

if (PHASE\_OFF1(Js::BackgroundRepeatMarkPhase))

{

return;

}

// We always do one repeat mark pass.

size\_t rescannedPageCount = this->BackgroundRepeatMark();

if (this->NeedOOMRescan() || this->isAborting)

{

// OOM'ed. Let's not continue

return;

}

Assert(rescannedPageCount != Recycler::InvalidScanRootBytes);

// If we rescanned enough pages in the previous repeat mark pass, then do one more

// to try to reduce the amount of work we need to do in-thread

if (rescannedPageCount >= RecyclerHeuristic::BackgroundSecondRepeatMarkThreshold)

{

this->BackgroundRepeatMark();

if (this->NeedOOMRescan() || this->isAborting)

{

// OOM'ed. Let's not continue

return;

}

}

}

void

Recycler::BackgroundResetMarks()

{

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(this, Js::BackgroundResetMarksPhase);

GCETW(GC\_BACKGROUNDRESETMARKS\_START, (this));

Assert(IsMarkStackEmpty());

this->scanPinnedObjectMap = true;

this->hasScannedInitialImplicitRoots = false;

heapBlockMap.ResetMarks();

autoHeap.ResetMarks(this->enableScanImplicitRoots ? ResetMarkFlags\_InBackgroundThreadImplicitRoots : ResetMarkFlags\_InBackgroundThread);

GCETW(GC\_BACKGROUNDRESETMARKS\_STOP, (this));

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, Js::BackgroundResetMarksPhase);

}

void

Recycler::PrepareBackgroundFindRoots()

{

Assert(!this->hasPendingConcurrentFindRoot);

this->hasPendingConcurrentFindRoot = true;

// Save the thread context here. The background thread

// will use this saved context for the marking instead of

// trying to get the live thread context of the thread

SAVE\_THREAD\_CONTEXT();

// Temporarily disable resize so the background can scan without

// the memory being freed from under it

pinnedObjectMap.DisableResize();

// Update the cached info for big blocks in the guest arena

DListBase<GuestArenaAllocator>::EditingIterator guestArenaIter(&guestArenaList);

while (guestArenaIter.Next())

{

GuestArenaAllocator& allocator = guestArenaIter.Data();

allocator.SetLockBlockList(true);

if (allocator.pendingDelete)

{

Assert(this->hasPendingDeleteGuestArena);

allocator.SetLockBlockList(false);

guestArenaIter.RemoveCurrent(&HeapAllocator::Instance);

}

else if (this->backgroundFinishMarkCount == 0)

{

// Update the cached info for big block

allocator.GetBigBlocks(false);

}

}

this->hasPendingDeleteGuestArena = false;

}

void

Recycler::RevertPrepareBackgroundFindRoots()

{

Assert(this->hasPendingConcurrentFindRoot);

this->hasPendingConcurrentFindRoot = false;

pinnedObjectMap.EnableResize();

}

size\_t

Recycler::BackgroundFindRoots()

{

#ifdef RECYCLER\_STATS

size\_t lastMarkCount = this->collectionStats.markData.markCount;

#endif

size\_t scanRootBytes = 0;

Assert(this->IsConcurrentFindRootState());

Assert(this->hasPendingConcurrentFindRoot);

Assert(this->inPartialCollectMode || this->DoQueueTrackedObject());

// Only mark pinned object and guest arenas, which is where most of the roots are.

// When we go back to the main thread to rescan, we will scan the rest of the root.

// NOTE: purposefully not marking the transientPinnedObject there. as it is transient :)

// background mark the pinned object. Since we are in concurrent find root state

// the main thread won't delete any entries from the map, so concurrent read

// to the map safe.

GCETW(GC\_BACKGROUNDSCANROOTS\_START, (this));

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(this, Js::BackgroundFindRootsPhase);

scanRootBytes += this->ScanPinnedObjects</\*background = \*/true>();

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(this, Js::FindRootArenaPhase);

// background mark the guest arenas. Since we are in concurrent find root state

// the main thread won't delete any arena, so concurrent reads to them are ok.

DListBase<GuestArenaAllocator>::EditingIterator guestArenaIter(&guestArenaList);

while (guestArenaIter.Next())

{

GuestArenaAllocator& allocator = guestArenaIter.Data();

if (allocator.pendingDelete)

{

// Skip guest arena that are already marked for delete

Assert(this->hasPendingDeleteGuestArena);

continue;

}

scanRootBytes += ScanArena(&allocator, true);

}

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, Js::FindRootArenaPhase);

this->ScanImplicitRoots();

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, Js::BackgroundFindRootsPhase);

this->hasPendingConcurrentFindRoot = false;

this->collectionState = CollectionStateConcurrentMark;

GCETW(GC\_BACKGROUNDSCANROOTS\_STOP, (this));

RECYCLER\_STATS\_ADD(this, rootCount, this->collectionStats.markData.markCount - lastMarkCount);

return scanRootBytes;

}

size\_t

Recycler::BackgroundFinishMark()

{

Assert(this->inPartialCollectMode || this->DoQueueTrackedObject());

Assert(collectionState == CollectionStateConcurrentFinishMark);

size\_t rescannedRootBytes = FinishMarkRescan(true) \* AutoSystemInfo::PageSize;

this->collectionState = CollectionStateConcurrentFindRoots;

rescannedRootBytes += this->BackgroundFindRoots();

this->collectionState = CollectionStateConcurrentFinishMark;

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(this, Js::MarkPhase);

ProcessMark(true);

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, Js::MarkPhase);

return rescannedRootBytes;

}

void

Recycler::SweepPendingObjects(RecyclerSweep& recyclerSweep)

{

autoHeap.SweepPendingObjects(recyclerSweep);

}

void

Recycler::ConcurrentTransferSweptObjects(RecyclerSweep& recyclerSweep)

{

Assert(!recyclerSweep.IsBackground());

Assert((this->collectionState & Collection\_TransferSwept) == Collection\_TransferSwept);

if (this->hasBackgroundFinishPartial)

{

this->hasBackgroundFinishPartial = false;

this->ClearPartialCollect();

}

autoHeap.ConcurrentTransferSweptObjects(recyclerSweep);

}

#ifdef PARTIAL\_GC\_ENABLED

void

Recycler::ConcurrentPartialTransferSweptObjects(RecyclerSweep& recyclerSweep)

{

Assert(!recyclerSweep.IsBackground());

Assert(!this->hasBackgroundFinishPartial);

autoHeap.ConcurrentPartialTransferSweptObjects(recyclerSweep);

}

#endif

BOOL

Recycler::FinishConcurrentCollectWrapped(CollectionFlags flags)

{

this->allowDispose = (flags & CollectOverride\_AllowDispose) == CollectOverride\_AllowDispose;

#ifdef CONCURRENT\_GC\_ENABLED

this->skipStack = ((flags & CollectOverride\_SkipStack) != 0);

DebugOnly(this->isConcurrentGCOnIdle = (flags == CollectOnScriptIdle));

#endif

BOOL collected = collectionWrapper->ExecuteRecyclerCollectionFunction(this, &Recycler::FinishConcurrentCollect, flags);

return collected;

}

BOOL

Recycler::WaitForConcurrentThread(DWORD waitTime)

{

Assert(this->IsConcurrentState() || this->collectionState == CollectionStateParallelMark);

RECYCLER\_PROFILE\_EXEC\_BEGIN(this, Js::ConcurrentWaitPhase);

if (concurrentThread != NULL)

{

// Set the priority back to normal before we wait to ensure it doesn't starve

SetThreadPriority(this->concurrentThread, THREAD\_PRIORITY\_NORMAL);

}

DWORD ret = WaitForSingleObject(concurrentWorkDoneEvent, waitTime);

if (concurrentThread != NULL)

{

if (ret == WAIT\_TIMEOUT)

{

// Keep the priority boost.

priorityBoost = true;

}

else

{

Assert(ret == WAIT\_OBJECT\_0);

// Back to below normal

SetThreadPriority(this->concurrentThread, THREAD\_PRIORITY\_BELOW\_NORMAL);

priorityBoost = false;

}

}

RECYCLER\_PROFILE\_EXEC\_END(this, Js::ConcurrentWaitPhase);

return (ret == WAIT\_OBJECT\_0);

}

void

Recycler::FlushBackgroundPages()

{

recyclerPageAllocator.SuspendIdleDecommit();

recyclerPageAllocator.FlushBackgroundPages();

recyclerPageAllocator.ResumeIdleDecommit();

recyclerLargeBlockPageAllocator.SuspendIdleDecommit();

recyclerLargeBlockPageAllocator.FlushBackgroundPages();

recyclerLargeBlockPageAllocator.ResumeIdleDecommit();

this->threadPageAllocator->SuspendIdleDecommit();

this->threadPageAllocator->FlushBackgroundPages();

this->threadPageAllocator->ResumeIdleDecommit();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.SuspendIdleDecommit();

recyclerWithBarrierPageAllocator.FlushBackgroundPages();

recyclerWithBarrierPageAllocator.ResumeIdleDecommit();

#endif

}

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

AutoProtectPages::AutoProtectPages(Recycler\* recycler, bool protectEnabled) :

isReadOnly(false),

recycler(recycler)

{

if (protectEnabled)

{

recycler->heapBlockMap.MakeAllPagesReadOnly(recycler);

isReadOnly = true;

}

}

AutoProtectPages::~AutoProtectPages()

{

Unprotect();

}

void AutoProtectPages::Unprotect()

{

if (isReadOnly)

{

recycler->heapBlockMap.MakeAllPagesReadWrite(recycler);

isReadOnly = false;

}

}

#endif

BOOL

Recycler::FinishConcurrentCollect(CollectionFlags flags)

{

if (!this->IsConcurrentState())

{

Assert(false);

return false;

}

#ifdef PROFILE\_EXEC

Js::Phase concurrentPhase = Js::ConcurrentCollectPhase;

#endif

RECYCLER\_PROFILE\_EXEC\_BEGIN2(this, Js::RecyclerPhase,

(concurrentPhase = ((this->inPartialCollectMode && this->IsConcurrentMarkState())?

Js::ConcurrentPartialCollectPhase : Js::ConcurrentCollectPhase)));

// Don't do concurrent sweep if we have priority boosted.

const BOOL forceInThread = flags & CollectOverride\_ForceInThread;

bool concurrent = (flags & CollectMode\_Concurrent) != 0;

concurrent = concurrent && (!priorityBoost || this->backgroundRescanCount != 1);

#ifdef RECYCLER\_TRACE

collectionParam.priorityBoostConcurentSweepOverride = priorityBoost;

#endif

const DWORD waitTime = forceInThread? INFINITE : RecyclerHeuristic::FinishConcurrentCollectWaitTime(this->GetRecyclerFlagsTable());

GCETW(GC\_FINISHCONCURRENTWAIT\_START, (this, waitTime));

const BOOL waited = WaitForConcurrentThread(waitTime);

GCETW(GC\_FINISHCONCURRENTWAIT\_STOP, (this, !waited));

if (!waited)

{

RECYCLER\_PROFILE\_EXEC\_END2(this, concurrentPhase, Js::RecyclerPhase);

return false;

}

bool needConcurrentSweep = false;

if (collectionState == CollectionStateRescanWait)

{

GCETW(GC\_START, (this, ETWEvent\_ConcurrentRescan));

#ifdef ENABLE\_BASIC\_TELEMETRY

if(GetCurrentThreadContextId()==mainThreadId && IsMemProtectMode()==false)

{

gcTel.LogGCPauseStartTime();

}

#endif

#ifdef RECYCLER\_TRACE

PrintCollectTrace(this->inPartialCollectMode ? Js::ConcurrentPartialCollectPhase : Js::ConcurrentMarkPhase, true);

#endif

collectionState = CollectionStateRescanFindRoots;

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

// TODO: Change this behavior

// ProtectPagesOnRescan is not supported in PageHeap mode because the page protection is changed

// outside the PageAllocator in PageHeap mode and so pages are not in the state that the

// PageAllocator expects when it goes to change the page protection

// One viable fix is to move the guard page protection logic outside of the heap blocks

// and into the page allocator

AssertMsg(!(IsPageHeapEnabled() && GetRecyclerFlagsTable().RecyclerProtectPagesOnRescan), "ProtectPagesOnRescan not supported in page heap mode");

AutoProtectPages protectPages(this, GetRecyclerFlagsTable().RecyclerProtectPagesOnRescan);

#endif

const bool backgroundFinishMark = !forceInThread && concurrent && ((flags & CollectOverride\_BackgroundFinishMark) != 0);

const DWORD finishMarkWaitTime = RecyclerHeuristic::BackgroundFinishMarkWaitTime(backgroundFinishMark, GetRecyclerFlagsTable());

size\_t rescanRootBytes = FinishMark(finishMarkWaitTime);

if (rescanRootBytes == Recycler::InvalidScanRootBytes)

{

Assert(this->IsMarkState());

RECYCLER\_PROFILE\_EXEC\_END2(this, concurrentPhase, Js::RecyclerPhase);

GCETW(GC\_STOP, (this, ETWEvent\_ConcurrentRescan));

#ifdef ENABLE\_BASIC\_TELEMETRY

if (GetCurrentThreadContextId() == mainThreadId && IsMemProtectMode() == false)

{

gcTel.LogGCPauseEndTime();

}

#endif

// we timeout trying to mark.

return false;

}

#ifdef RECYCLER\_STATS

collectionStats.continueCollectAllocBytes = autoHeap.uncollectedAllocBytes;

#endif

#ifdef RECYCLER\_VERIFY\_MARK

if (GetRecyclerFlagsTable().RecyclerVerifyMark)

{

this->VerifyMark();

}

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

protectPages.Unprotect();

#endif

needConcurrentSweep = this->Sweep(rescanRootBytes, concurrent, true);

GCETW(GC\_STOP, (this, ETWEvent\_ConcurrentRescan));

}

else

{

GCETW(GC\_START, (this, ETWEvent\_ConcurrentTransferSwept));

#ifdef ENABLE\_BASIC\_TELEMETRY

if(GetCurrentThreadContextId()==mainThreadId && IsMemProtectMode()==false)

{

gcTel.LogGCPauseStartTime();

}

#endif

GCETW(GC\_FLUSHZEROPAGE\_START, (this));

Assert(collectionState == CollectionStateTransferSweptWait);

#ifdef RECYCLER\_TRACE

PrintCollectTrace(Js::ConcurrentSweepPhase, true);

#endif

collectionState = CollectionStateTransferSwept;

// We should have zeroed all the pages in the background thread

Assert(!recyclerPageAllocator.HasZeroQueuedPages());

Assert(!recyclerLargeBlockPageAllocator.HasZeroQueuedPages());

this->FlushBackgroundPages();

GCETW(GC\_FLUSHZEROPAGE\_STOP, (this));

GCETW(GC\_TRANSFERSWEPTOBJECTS\_START, (this));

Assert(this->recyclerSweep != nullptr);

Assert(!this->recyclerSweep->IsBackground());

#ifdef PARTIAL\_GC\_ENABLED

if (this->inPartialCollectMode)

{

ConcurrentPartialTransferSweptObjects(\*this->recyclerSweep);

}

else

#endif

{

ConcurrentTransferSweptObjects(\*this->recyclerSweep);

}

recyclerSweep->EndSweep();

GCETW(GC\_TRANSFERSWEPTOBJECTS\_STOP, (this));

GCETW(GC\_STOP, (this, ETWEvent\_ConcurrentTransferSwept));

#ifdef ENABLE\_BASIC\_TELEMETRY

if (GetCurrentThreadContextId() == mainThreadId && IsMemProtectMode() == false)

{

gcTel.LogGCPauseEndTime();

}

#endif

}

RECYCLER\_PROFILE\_EXEC\_END2(this, concurrentPhase, Js::RecyclerPhase);

FinishCollection(needConcurrentSweep);

if (!this->CollectionInProgress())

{

if (NeedExhaustiveRepeatCollect())

{

DoCollect((CollectionFlags)(flags & ~CollectMode\_Partial));

}

else

{

EndCollection();

}

}

return true;

}

int

Recycler::ExceptFilter(LPEXCEPTION\_POINTERS pEP)

{

#if DBG

// Assert exception code

if (pEP->ExceptionRecord->ExceptionCode == STATUS\_ASSERTION\_FAILURE)

{

return EXCEPTION\_CONTINUE\_SEARCH;

}

#endif

#ifdef GENERATE\_DUMP

if (Js::Configuration::Global.flags.IsEnabled(Js::DumpOnCrashFlag))

{

Js::Throw::GenerateDump(pEP, Js::Configuration::Global.flags.DumpOnCrash);

}

#endif

#if DBG && \_M\_IX86

int callerEBP = \*((int\*)pEP->ContextRecord->Ebp);

Output::Print(L"Recycler Concurrent Thread: Uncaught exception: EIP: 0x%X ExceptionCode: 0x%X EBP: 0x%X ReturnAddress: 0x%X ReturnAddress2: 0x%X\n",

pEP->ExceptionRecord->ExceptionAddress, pEP->ExceptionRecord->ExceptionCode, pEP->ContextRecord->Eip,

pEP->ContextRecord->Ebp, \*((int\*)pEP->ContextRecord->Ebp + 1), \*((int\*) callerEBP + 1));

#endif

Output::Flush();

return EXCEPTION\_CONTINUE\_SEARCH;

}

unsigned int

Recycler::StaticThreadProc(LPVOID lpParameter)

{

DWORD ret = (DWORD)-1;

\_\_try

{

Recycler \* recycler = (Recycler \*)lpParameter;

#if DBG

recycler->concurrentThreadExited = false;

#endif

ret = recycler->ThreadProc();

}

\_\_except(Recycler::ExceptFilter(GetExceptionInformation()))

{

Assert(false);

}

return ret;

}

void

Recycler::StaticBackgroundWorkCallback(void \* callbackData)

{

Recycler \* recycler = (Recycler \*) callbackData;

recycler->DoBackgroundWork(true);

}

#ifdef ENABLE\_JS\_ETW

static ETWEventGCActivationKind

BackgroundMarkETWEventGCActivationKind(CollectionState collectionState)

{

return collectionState == CollectionStateConcurrentFinishMark?

ETWEvent\_ConcurrentFinishMark : ETWEvent\_ConcurrentMark;

}

#endif

void

Recycler::DoBackgroundWork(bool forceForeground)

{

if (this->collectionState == CollectionStateConcurrentWrapperCallback)

{

this->collectionWrapper->ConcurrentCallback();

}

else if (this->collectionState == CollectionStateParallelMark)

{

this->ProcessParallelMark(false, &this->markContext);

}

else if (this->IsConcurrentMarkState())

{

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(this, this->collectionState == CollectionStateConcurrentFinishMark?

Js::BackgroundFinishMarkPhase : Js::ConcurrentMarkPhase);

GCETW(GC\_START, (this, BackgroundMarkETWEventGCActivationKind(this->collectionState)));

#ifdef ENABLE\_BASIC\_TELEMETRY

if (GetCurrentThreadContextId() == mainThreadId && IsMemProtectMode() == false)

{

gcTel.LogGCPauseStartTime();

}

#endif

DebugOnly(this->markContext.GetPageAllocator()->SetConcurrentThreadId(::GetCurrentThreadId()));

Assert(this->enableConcurrentMark);

if (this->collectionState != CollectionStateConcurrentFinishMark)

{

this->StartQueueTrackedObject();

}

switch (this->collectionState)

{

case CollectionStateConcurrentResetMarks:

this->BackgroundResetMarks();

this->BackgroundResetWriteWatchAll();

this->collectionState = CollectionStateConcurrentFindRoots;

// fall-through

case CollectionStateConcurrentFindRoots:

this->BackgroundFindRoots();

this->BackgroundScanStack();

this->collectionState = CollectionStateConcurrentMark;

// fall-through

case CollectionStateConcurrentMark:

this->BackgroundMark();

Assert(this->collectionState == CollectionStateConcurrentMark);

RECORD\_TIMESTAMP(concurrentMarkFinishTime);

break;

case CollectionStateConcurrentFinishMark:

this->backgroundRescanRootBytes = this->BackgroundFinishMark();

Assert(!HasPendingMarkObjects());

break;

default:

Assert(false);

break;

};

GCETW(GC\_STOP, (this, BackgroundMarkETWEventGCActivationKind(this->collectionState)));

#ifdef ENABLE\_BASIC\_TELEMETRY

if (GetCurrentThreadContextId() == mainThreadId && IsMemProtectMode() == false)

{

gcTel.LogGCPauseEndTime();

}

#endif

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, this->collectionState == CollectionStateConcurrentFinishMark?

Js::BackgroundFinishMarkPhase : Js::ConcurrentMarkPhase);

this->collectionState = CollectionStateRescanWait;

DebugOnly(this->markContext.GetPageAllocator()->ClearConcurrentThreadId());

}

else

{

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(this, Js::ConcurrentSweepPhase);

GCETW(GC\_START, (this, ETWEvent\_ConcurrentSweep));

#ifdef ENABLE\_BASIC\_TELEMETRY

if (GetCurrentThreadContextId() == mainThreadId && IsMemProtectMode() == false)

{

gcTel.LogGCPauseStartTime();

}

#endif

GCETW(GC\_BACKGROUNDZEROPAGE\_START, (this));

Assert(this->enableConcurrentSweep);

Assert(this->collectionState == CollectionStateConcurrentSweep);

// Zero the queued pages first so they are available to be allocated

recyclerPageAllocator.BackgroundZeroQueuedPages();

recyclerLargeBlockPageAllocator.BackgroundZeroQueuedPages();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.BackgroundZeroQueuedPages();

#endif

GCETW(GC\_BACKGROUNDZEROPAGE\_STOP, (this));

GCETW(GC\_BACKGROUNDSWEEP\_START, (this));

Assert(this->recyclerSweep != nullptr);

this->recyclerSweep->BackgroundSweep();

uint sweptBytes = 0;

#ifdef RECYCLER\_STATS

sweptBytes = (uint)collectionStats.objectSweptBytes;

#endif

GCETW(GC\_BACKGROUNDSWEEP\_STOP, (this, sweptBytes));

// Drain the zero queue again as we might have free more during sweep

// in the background

GCETW(GC\_BACKGROUNDZEROPAGE\_START, (this));

recyclerPageAllocator.BackgroundZeroQueuedPages();

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

recyclerWithBarrierPageAllocator.BackgroundZeroQueuedPages();

#endif

recyclerLargeBlockPageAllocator.BackgroundZeroQueuedPages();

GCETW(GC\_BACKGROUNDZEROPAGE\_STOP, (this));

GCETW(GC\_STOP, (this, ETWEvent\_ConcurrentSweep));

#ifdef ENABLE\_BASIC\_TELEMETRY

if (GetCurrentThreadContextId() == mainThreadId && IsMemProtectMode() == false)

{

gcTel.LogGCPauseEndTime();

}

#endif

Assert(this->collectionState == CollectionStateConcurrentSweep);

this->collectionState = CollectionStateTransferSweptWait;

RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(this, Js::ConcurrentSweepPhase);

}

SetEvent(this->concurrentWorkDoneEvent);

collectionWrapper->WaitCollectionCallBack();

}

DWORD

Recycler::ThreadProc()

{

Assert(this->IsConcurrentEnabled());

#if !defined(\_UCRT)

// We do this before we set the concurrentWorkDoneEvent because GetModuleHandleEx requires

// getting the loader lock. We could have the following case:

// Thread A => Initialize Concurrent Thread (C)

// C signals Signal Done

// C yields since its lower priority

// Thread A starts running- and is told to shut down.

// Thread A grabs loader lock as part of the shutdown sequence

// Thread A waits for C to be done

// C wakes up now- and tries to grab loader lock.

// To prevent this deadlock, we call GetModuleHandleEx first and then set the concurrentWorkDoneEvent

HMODULE dllHandle = NULL;

if (!GetModuleHandleEx(GET\_MODULE\_HANDLE\_EX\_FLAG\_FROM\_ADDRESS, (LPCTSTR)&Recycler::StaticThreadProc, &dllHandle))

{

dllHandle = NULL;

}

#endif

#ifdef ENABLE\_JS\_ETW

// Create an ETW ActivityId for this thread, to help tools correlate ETW events we generate

GUID activityId = { 0 };

auto result = EventActivityIdControl(EVENT\_ACTIVITY\_CTRL\_CREATE\_SET\_ID, &activityId);

Assert(result == ERROR\_SUCCESS);

#endif

// Signal that the thread has started

SetEvent(this->concurrentWorkDoneEvent);

SetThreadPriority(::GetCurrentThread(), THREAD\_PRIORITY\_BELOW\_NORMAL);

#if defined(DBG) && defined(PROFILE\_EXEC)

this->backgroundProfilerPageAllocator.SetConcurrentThreadId(::GetCurrentThreadId());

#endif

#ifdef IDLE\_DECOMMIT\_ENABLED

DWORD handleCount = this->concurrentIdleDecommitEvent? 2 : 1;

HANDLE handles[2] = { this->concurrentWorkReadyEvent, this->concurrentIdleDecommitEvent };

#endif

do

{

#ifdef IDLE\_DECOMMIT\_ENABLED

needIdleDecommitSignal = IdleDecommitSignal\_None;

DWORD threadPageAllocatorWaitTime = threadPageAllocator->IdleDecommit();

DWORD recyclerPageAllocatorWaitTime = recyclerPageAllocator.IdleDecommit();

DWORD waitTime = min(threadPageAllocatorWaitTime, recyclerPageAllocatorWaitTime);

DWORD recyclerLargeBlockPageAllocatorWaitTime = recyclerLargeBlockPageAllocator.IdleDecommit();

waitTime = min(waitTime, recyclerLargeBlockPageAllocatorWaitTime);

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

DWORD recyclerWithBarrierPageAllocatorWaitTime = recyclerWithBarrierPageAllocator.IdleDecommit();

waitTime = min(waitTime, recyclerWithBarrierPageAllocatorWaitTime);

#endif

if (waitTime == INFINITE)

{

DWORD ret = ::InterlockedCompareExchange(&needIdleDecommitSignal, IdleDecommitSignal\_NeedSignal, IdleDecommitSignal\_None);

if (ret == IdleDecommitSignal\_NeedTimer)

{

#if DBG

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::IdleDecommitPhase))

{

Output::Print(L"Recycler Thread IdleDecommit Need Timer\n");

Output::Flush();

}

#endif

continue;

}

}

#if DBG

else

{

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::IdleDecommitPhase))

{

Output::Print(L"Recycler Thread IdleDecommit Wait %d\n", waitTime);

Output::Flush();

}

}

#endif

DWORD result = WaitForMultipleObjectsEx(handleCount, handles, FALSE, waitTime, FALSE);

if (result != WAIT\_OBJECT\_0)

{

Assert((handleCount == 2 && result == WAIT\_OBJECT\_0 + 1) || (waitTime != INFINITE && result == WAIT\_TIMEOUT));

#if DBG

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::IdleDecommitPhase))

{

if (result == WAIT\_TIMEOUT)

{

Output::Print(L"Recycler Thread IdleDecommit Timeout: %d\n", waitTime);

}

else

{

Output::Print(L"Recycler Thread IdleDecommit Signaled\n");

}

Output::Flush();

}

#endif

continue;

}

#else

DWORD result = WaitForSingleObject(this->concurrentWorkReadyEvent, INFINITE);

Assert(result == WAIT\_OBJECT\_0);

#endif

if (this->collectionState == CollectionStateExit)

{

#if DBG

this->concurrentThreadExited = true;

#endif

break;

}

DoBackgroundWork();

}

while (true);

SetEvent(this->concurrentWorkDoneEvent);

#if !defined(\_UCRT)

if (dllHandle)

{

FreeLibraryAndExitThread(dllHandle, 0);

}

else

#endif

{

return 0;

}

}

#endif //defined(CONCURRENT\_GC\_ENABLED)

void

Recycler::FinishCollection(bool needConcurrentSweep)

{

Assert(!!this->InConcurrentSweep() == needConcurrentSweep);

if (!needConcurrentSweep)

{

FinishCollection();

}

else

{

FinishDisposeObjects();

}

}

void

Recycler::FinishCollection()

{

Assert(!this->hasBackgroundFinishPartial);

Assert(!this->hasPendingDeleteGuestArena);

// Reset the time heuristics

ScheduleNextCollection();

{

AutoSwitchCollectionStates collectionState(this,

/\* entry state \*/ CollectionStatePostCollectionCallback,

/\* exit state \*/ CollectionStateNotCollecting);

collectionWrapper->PostCollectionCallBack();

}

this->backgroundFinishMarkCount = 0;

// Do a partial page decommit now

if (decommitOnFinish)

{

ForRecyclerPageAllocator(DecommitNow(false));

this->decommitOnFinish = false;

}

RECYCLER\_SLOW\_CHECK(autoHeap.Check());

#ifdef RECYCLER\_MEMORY\_VERIFY

this->Verify(Js::RecyclerPhase);

#endif

#ifdef RECYCLER\_FINALIZE\_CHECK

autoHeap.VerifyFinalize();

#endif

#ifdef ENABLE\_JS\_ETW

FlushFreeRecord();

#endif

FinishDisposeObjects();

#ifdef RECYCLER\_FINALIZE\_CHECK

if (!this->IsMarkState())

{

autoHeap.VerifyFinalize();

}

#endif

#ifdef RECYCLER\_STATS

if (CUSTOM\_PHASE\_STATS1(this->GetRecyclerFlagsTable(), Js::RecyclerPhase))

{

PrintCollectStats();

}

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

if (MemoryProfiler::IsTraceEnabled(true))

{

PrintAllocStats();

}

#endif

#ifdef DUMP\_FRAGMENTATION\_STATS

if (GetRecyclerFlagsTable().DumpFragmentationStats)

{

autoHeap.DumpFragmentationStats();

}

#endif

RECORD\_TIMESTAMP(currentCollectionEndTime);

}

char \*

Recycler::Realloc(void\* buffer, size\_t existingBytes, size\_t requestedBytes, bool truncate)

{

Assert(requestedBytes > 0);

if (existingBytes == 0)

{

Assert(buffer == nullptr);

return Alloc(requestedBytes);

}

Assert(buffer != nullptr);

size\_t nbytes = AllocSizeMath::Align(requestedBytes, HeapConstants::ObjectGranularity);

// Since we successfully allocated, we shouldn't have integer overflow here

size\_t nbytesExisting = AllocSizeMath::Align(existingBytes, HeapConstants::ObjectGranularity);

Assert(nbytesExisting >= existingBytes);

if (nbytes == nbytesExisting)

{

return (char \*)buffer;

}

char\* replacementBuf = this->Alloc(requestedBytes);

if (replacementBuf != nullptr)

{

// Truncate

if (existingBytes > requestedBytes && truncate)

{

js\_memcpy\_s(replacementBuf, requestedBytes, buffer, requestedBytes);

}

else

{

js\_memcpy\_s(replacementBuf, requestedBytes, buffer, existingBytes);

}

}

if (nbytesExisting > 0)

{

this->Free(buffer, nbytesExisting);

}

return replacementBuf;

}

bool

Recycler::ForceSweepObject()

{

#ifdef RECYCLER\_TEST\_SUPPORT

if (BinaryFeatureControl::RecyclerTest())

{

if (checkFn != nullptr)

{

return true;

}

}

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

if (trackerDictionary != nullptr)

{

// Need to sweep object if we are tracing recycler allocs

return true;

}

#endif

#ifdef RECYCLER\_STATS

if (CUSTOM\_PHASE\_STATS1(this->GetRecyclerFlagsTable(), Js::RecyclerPhase))

{

return true;

}

#endif

#if DBG

// Force sweeping the object so we can assert that we are not sweeping objects that are still implicit roots

if (this->enableScanImplicitRoots)

{

return true;

}

#endif

return false;

}

bool

Recycler::ShouldIdleCollectOnExit()

{

// Always reset partial heuristics even if we are not doing idle collecting

// So we don't carry the heuristics to the next script activation

this->ResetPartialHeuristicCounters();

if (this->CollectionInProgress())

{

#ifdef RECYCLER\_TRACE

CUSTOM\_PHASE\_PRINT\_VERBOSE\_TRACE1(GetRecyclerFlagsTable(), Js::IdleCollectPhase, L"%04X> Skipping scheduling Idle Collect. Reason: Collection in progress\n", ::GetCurrentThreadId());

#endif

// Don't schedule a idle collect if there is a collection going on already

// IDLE-GC-TODO: Fix ResetHeuristics in the GC so we can detect memory allocation during

// the concurrent collect and still schedule an idle collect

return false;

}

if (CUSTOM\_PHASE\_FORCE1(GetRecyclerFlagsTable(), Js::IdleCollectPhase))

{

return true;

}

ulong nextTime = tickCountNextCollection - tickDiffToNextCollect;

// We will try to start a concurrent collect if we are within .9 ms to next scheduled collection, AND,

// the size of allocation is larger than 32M. This is similar to CollectionAllocation logic, just

// earlier in both time heuristic and size heuristic, so we can do some concurrent GC while we are

// not in script.

if (autoHeap.uncollectedAllocBytes >= RecyclerHeuristic::Instance.MaxUncollectedAllocBytesOnExit

&& GetTickCount() > nextTime)

{

#ifdef RECYCLER\_TRACE

if (CUSTOM\_PHASE\_TRACE1(GetRecyclerFlagsTable(), Js::IdleCollectPhase))

{

if (autoHeap.uncollectedAllocBytes >= RecyclerHeuristic::Instance.MaxUncollectedAllocBytesOnExit)

{

Output::Print(L"%04X> Idle collect on exit: alloc %d\n", ::GetCurrentThreadId(), autoHeap.uncollectedAllocBytes);

}

else

{

Output::Print(L"%04X> Idle collect on exit: time %d\n", ::GetCurrentThreadId(), tickCountNextCollection - GetTickCount());

}

Output::Flush();

}

#endif

this->CollectNow<CollectNowConcurrent>();

return false;

}

Assert(!this->CollectionInProgress());

// Idle GC use the size heuristic. Only need to schedule on if we passed it.

return (autoHeap.uncollectedAllocBytes >= RecyclerHeuristic::IdleUncollectedAllocBytesCollection);

}

bool

RecyclerParallelThread::StartConcurrent()

{

if (this->recycler->threadService->HasCallback())

{

// This may be the first time. If so, initialize by creating the doneEvent.

if (this->concurrentWorkDoneEvent == NULL)

{

this->concurrentWorkDoneEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

if (this->concurrentWorkDoneEvent == nullptr)

{

return false;

}

}

Assert(concurrentThread == NULL);

Assert(concurrentWorkReadyEvent == NULL);

// Invoke thread service to process work

if (!this->recycler->threadService->Invoke(RecyclerParallelThread::StaticBackgroundWorkCallback, this))

{

return false;

}

}

else

{

// This may be the first time. If so, initialize and create thread.

if (this->concurrentWorkDoneEvent == NULL)

{

return this->EnableConcurrent(false);

}

else

{

Assert(this->concurrentThread != NULL);

Assert(this->concurrentWorkReadyEvent != NULL);

// signal that thread has been initialized

SetEvent(this->concurrentWorkReadyEvent);

}

}

return true;

}

bool

RecyclerParallelThread::EnableConcurrent(bool waitForThread)

{

this->synchronizeOnStartup = waitForThread;

Assert(this->concurrentWorkDoneEvent == NULL);

Assert(this->concurrentWorkReadyEvent == NULL);

Assert(this->concurrentThread == NULL);

this->concurrentWorkDoneEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

if (this->concurrentWorkDoneEvent == nullptr)

{

return false;

}

this->concurrentWorkReadyEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

if (this->concurrentWorkReadyEvent == nullptr)

{

CloseHandle(this->concurrentWorkDoneEvent);

this->concurrentWorkDoneEvent = NULL;

return false;

}

this->concurrentThread = (HANDLE)\_beginthreadex(NULL, Recycler::ConcurrentThreadStackSize, &RecyclerParallelThread::StaticThreadProc, this, STACK\_SIZE\_PARAM\_IS\_A\_RESERVATION, NULL);

if (this->concurrentThread != nullptr && waitForThread)

{

// Wait for thread to initialize

HANDLE handle[2] = { this->concurrentWorkDoneEvent, this->concurrentThread };

DWORD ret = WaitForMultipleObjectsEx(2, handle, FALSE, INFINITE, FALSE);

if (ret == WAIT\_OBJECT\_0)

{

return true;

}

CloseHandle(concurrentThread);

concurrentThread = nullptr;

}

if (this->concurrentThread == nullptr)

{

CloseHandle(this->concurrentWorkDoneEvent);

this->concurrentWorkDoneEvent = NULL;

CloseHandle(this->concurrentWorkReadyEvent);

this->concurrentWorkReadyEvent = NULL;

return false;

}

return true;

}

template <uint parallelId>

void

Recycler::ParallelWorkFunc()

{

Assert(parallelId == 0 || parallelId == 1);

MarkContext \* markContext = (parallelId == 0 ? &this->parallelMarkContext2 : &this->parallelMarkContext3);

switch (this->collectionState)

{

case CollectionStateParallelMark:

this->ProcessParallelMark(false, markContext);

break;

case CollectionStateBackgroundParallelMark:

this->ProcessParallelMark(true, markContext);

break;

default:

Assert(false);

}

}

void

RecyclerParallelThread::WaitForConcurrent()

{

Assert(this->concurrentThread != NULL || this->recycler->threadService->HasCallback());

Assert(this->concurrentWorkDoneEvent != NULL);

DWORD ret = WaitForSingleObject(concurrentWorkDoneEvent, INFINITE);

Assert(ret == WAIT\_OBJECT\_0);

}

void

RecyclerParallelThread::Shutdown()

{

Assert(this->recycler->collectionState == CollectionStateExit);

if (this->recycler->threadService->HasCallback())

{

if (this->concurrentWorkDoneEvent != NULL)

{

CloseHandle(this->concurrentWorkDoneEvent);

this->concurrentWorkDoneEvent = NULL;

}

}

else

{

if (this->concurrentThread != NULL)

{

HANDLE handles[2] = { concurrentWorkDoneEvent, concurrentThread };

SetEvent(concurrentWorkReadyEvent);

// During process shutdown, OS might kill this (recycler parallel i.e. concurrent) thread and it will not get chance to signal concurrentWorkDoneEvent.

// When we are performing shutdown of main (recycler) thread here, if we wait on concurrentWorkDoneEvent, WaitForObject() will never return.

// Hence wait for concurrentWorkDoneEvent + concurrentThread so if concurrentThread got killed, WaitForObject() will return and we will

// proceed further.

DWORD fRet = WaitForMultipleObjectsEx(2, handles, FALSE, INFINITE, FALSE);

AssertMsg(fRet != WAIT\_FAILED, "Check handles passed to WaitForMultipleObjectsEx.");

CloseHandle(this->concurrentWorkDoneEvent);

this->concurrentWorkDoneEvent = NULL;

CloseHandle(this->concurrentWorkReadyEvent);

this->concurrentWorkReadyEvent = NULL;

CloseHandle(this->concurrentThread);

this->concurrentThread = NULL;

}

}

Assert(this->concurrentThread == NULL);

Assert(this->concurrentWorkReadyEvent == NULL);

Assert(this->concurrentWorkDoneEvent == NULL);

}

// static

unsigned int

RecyclerParallelThread::StaticThreadProc(LPVOID lpParameter)

{

DWORD ret = (DWORD)-1;

\_\_try

{

RecyclerParallelThread \* parallelThread = (RecyclerParallelThread \*)lpParameter;

Recycler \* recycler = parallelThread->recycler;

RecyclerParallelThread::WorkFunc workFunc = parallelThread->workFunc;

Assert(recycler->IsConcurrentEnabled());

#if !defined(\_UCRT)

HMODULE dllHandle = NULL;

if (!GetModuleHandleEx(GET\_MODULE\_HANDLE\_EX\_FLAG\_FROM\_ADDRESS, (LPCTSTR)&RecyclerParallelThread::StaticThreadProc, &dllHandle))

{

dllHandle = NULL;

}

#endif

#ifdef ENABLE\_JS\_ETW

// Create an ETW ActivityId for this thread, to help tools correlate ETW events we generate

GUID activityId = { 0 };

auto result = EventActivityIdControl(EVENT\_ACTIVITY\_CTRL\_CREATE\_SET\_ID, &activityId);

Assert(result == ERROR\_SUCCESS);

#endif

// If this thread is created on demand we already have work to process and do not need to wait

bool mustWait = parallelThread->synchronizeOnStartup;

do

{

if (mustWait)

{

// Signal completion and wait for next work

SetEvent(parallelThread->concurrentWorkDoneEvent);

DWORD result = WaitForSingleObject(parallelThread->concurrentWorkReadyEvent, INFINITE);

Assert(result == WAIT\_OBJECT\_0);

}

if (recycler->collectionState == CollectionStateExit)

{

// Exit thread

break;

}

// Invoke the workFunc to do real work

(recycler->\*workFunc)();

// We always wait after the first time

mustWait = true;

}

while (true);

// Signal to main thread that we have stopped processing and will shut down.

// Note that after this point, we cannot access anything on the Recycler instance

// because the main thread may have torn it down already.

SetEvent(parallelThread->concurrentWorkDoneEvent);

#if !defined(\_UCRT)

if (dllHandle)

{

FreeLibraryAndExitThread(dllHandle, 0);

}

#endif

ret = 0;

}

\_\_except(Recycler::ExceptFilter(GetExceptionInformation()))

{

Assert(false);

}

return ret;

}

// static

void

RecyclerParallelThread::StaticBackgroundWorkCallback(void \* callbackData)

{

RecyclerParallelThread \* parallelThread = (RecyclerParallelThread \*)callbackData;

Recycler \* recycler = parallelThread->recycler;

RecyclerParallelThread::WorkFunc workFunc = parallelThread->workFunc;

(recycler->\*workFunc)();

SetEvent(parallelThread->concurrentWorkDoneEvent);

}

#ifdef RECYCLER\_TRACE

void

Recycler::CaptureCollectionParam(CollectionFlags flags, bool repeat)

{

collectionParam.priorityBoostConcurentSweepOverride = false;

collectionParam.repeat = repeat;

collectionParam.finishOnly = false;

collectionParam.flags = flags;

collectionParam.uncollectedAllocBytes = autoHeap.uncollectedAllocBytes;

#ifdef PARTIAL\_GC\_ENABLED

collectionParam.uncollectedNewPageCountPartialCollect = this->uncollectedNewPageCountPartialCollect;

collectionParam.inPartialCollectMode = inPartialCollectMode;

collectionParam.uncollectedNewPageCount = autoHeap.uncollectedNewPageCount;

collectionParam.unusedPartialCollectFreeBytes = autoHeap.unusedPartialCollectFreeBytes;

#endif

}

void

Recycler::PrintCollectTrace(Js::Phase phase, bool finish, bool noConcurrentWork)

{

if (GetRecyclerFlagsTable().Trace.IsEnabled(Js::RecyclerPhase) ||

GetRecyclerFlagsTable().Trace.IsEnabled(phase))

{

const BOOL allocSize = collectionParam.flags & CollectHeuristic\_AllocSize;

const BOOL timedIfScriptActive = collectionParam.flags & CollectHeuristic\_TimeIfScriptActive;

const BOOL timedIfInScript = collectionParam.flags & CollectHeuristic\_TimeIfInScript;

const BOOL timed = (timedIfScriptActive && isScriptActive) || (timedIfInScript && isInScript) || (collectionParam.flags & CollectHeuristic\_Time);

const BOOL concurrent = collectionParam.flags & CollectMode\_Concurrent;

const BOOL finishConcurrent = collectionParam.flags & CollectOverride\_FinishConcurrent;

const BOOL exhaustive = collectionParam.flags & CollectMode\_Exhaustive;

const BOOL forceInThread = collectionParam.flags & CollectOverride\_ForceInThread;

const BOOL forceFinish = collectionParam.flags & CollectOverride\_ForceFinish;

#ifdef PARTIAL\_GC\_ENABLED

BOOL partial = collectionParam.flags & CollectMode\_Partial ;

#endif

Output::Print(L"%04X> RC(%p): %s%s%s%s%s%s%s:", this->mainThreadId, this,

collectionParam.domCollect? L"[DOM] " : L"",

collectionParam.repeat? L"[Repeat] ": L"",

this->inDispose? L"[Nested]" : L"",

forceInThread? L"Force In thread " : L"",

finish? L"Finish " : L"",

exhaustive? L"Exhaustive " : L"",

Js::PhaseNames[phase]);

if (noConcurrentWork)

{

Assert(finish);

Output::Print(L" No concurrent work");

}

else if (collectionParam.finishOnly)

{

Assert(!collectionParam.repeat);

Assert(finish);

#ifdef CONCURRENT\_GC\_ENABLED

if (collectionState == CollectionStateRescanWait)

{

if (forceFinish)

{

Output::Print(L" Force finish mark and sweep");

}

else if (concurrent && this->enableConcurrentSweep)

{

if (!collectionParam.priorityBoostConcurentSweepOverride)

{

Output::Print(L" Finish mark and start concurrent sweep");

}

else

{

Output::Print(L" Finish mark and sweep (priority boost overrided concurrent sweep)");

}

}

else

{

Output::Print(L" Finish mark and sweep");

}

}

else

{

Assert(collectionState == CollectionStateTransferSweptWait);

if (forceFinish)

{

Output::Print(L" Force finish sweep");

}

else

{

Output::Print(L" Finish sweep");

}

}

#endif // CONCURRENT\_GC\_ENABLED

}

else

{

if (finish && !concurrent)

{

Output::Print(L" Not concurrent collect");

}

if ((finish && finishConcurrent))

{

Output::Print(L" No heuristic");

}

#ifdef CONCURRENT\_GC\_ENABLED

else if (finish && priorityBoost)

{

Output::Print(L" Priority boost no heuristic");

}

#endif

else

{

Output::SkipToColumn(50);

bool byteCountUsed = false;

bool timeUsed = false;

#ifdef PARTIAL\_GC\_ENABLED

bool newPageUsed = false;

if (phase == Js::PartialCollectPhase || phase == Js::ConcurrentPartialCollectPhase)

{

Assert(collectionParam.flags & CollectMode\_Partial);

newPageUsed = !!allocSize;

}

else if (partial && collectionParam.inPartialCollectMode && collectionParam.uncollectedNewPageCount > collectionParam.uncollectedNewPageCountPartialCollect)

{

newPageUsed = true;

}

else

#endif // PARTIAL\_GC\_ENABLED

{

byteCountUsed = !!allocSize;

timeUsed = !!timed;

}

Output::Print(byteCountUsed? L"\*" : (allocSize? L" " : L"~"));

Output::Print(L"B:%8d ", collectionParam.uncollectedAllocBytes);

Output::Print(timeUsed? L"\*" : (timed? L" " : L"~"));

Output::Print(L"T:%4d ", -collectionParam.timeDiff);

#ifdef PARTIAL\_GC\_ENABLED

if (collectionParam.inPartialCollectMode)

{

Output::Print(L"L:%5d ", collectionParam.uncollectedNewPageCountPartialCollect);

}

else

{

Output::Print(L"L:----- ");

}

Output::Print(newPageUsed? L"\*" : (partial? L" " : L"~"));

Output::Print(L"P:%5d(%9d) ", collectionParam.uncollectedNewPageCount, collectionParam.uncollectedNewPageCount \* AutoSystemInfo::PageSize);

Output::Print(L"U:%8d", collectionParam.unusedPartialCollectFreeBytes);

#endif // PARTIAL\_GC\_ENABLED

}

}

Output::Print(L"\n");

Output::Flush();

}

}

#endif

#ifdef RECYCLER\_STATS

void

Recycler::PrintHeapBlockStats(wchar\_t const \* name, HeapBlock::HeapBlockType type)

{

size\_t liveCount = collectionStats.heapBlockCount[type] - collectionStats.heapBlockFreeCount[type];

Output::Print(L" %6s : %5d %5d %5d %5.1f", name,

liveCount, collectionStats.heapBlockFreeCount[type], collectionStats.heapBlockCount[type],

(double)collectionStats.heapBlockFreeCount[type] / (double)collectionStats.heapBlockCount[type] \* 100);

if (type < HeapBlock::SmallBlockTypeCount)

{

Output::Print(L" : %5d %6.1f : %5d %6.1f",

collectionStats.heapBlockSweptCount[type],

(double)collectionStats.heapBlockSweptCount[type] / (double)liveCount \* 100,

collectionStats.heapBlockConcurrentSweptCount[type],

(double)collectionStats.heapBlockConcurrentSweptCount[type] / (double)collectionStats.heapBlockSweptCount[type] \* 100);

}

}

void

Recycler::PrintHeapBlockMemoryStats(wchar\_t const \* name, HeapBlock::HeapBlockType type)

{

size\_t allocableFreeByteCount = collectionStats.heapBlockFreeByteCount[type];

size\_t partialUnusedBytes = 0;

if (this->enablePartialCollect)

{

partialUnusedBytes = allocableFreeByteCount

- collectionStats.smallNonLeafHeapBlockPartialReuseBytes[type];

allocableFreeByteCount -= partialUnusedBytes;

}

size\_t totalByteCount = (collectionStats.heapBlockCount[type] - collectionStats.heapBlockFreeCount[type]) \* AutoSystemInfo::PageSize;

size\_t liveByteCount = totalByteCount - collectionStats.heapBlockFreeByteCount[type];

Output::Print(L" %6s: %10d %10d", name, liveByteCount, allocableFreeByteCount);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect &&

(type == HeapBlock::HeapBlockType::SmallNormalBlockType

|| type == HeapBlock::HeapBlockType::SmallFinalizableBlockType

#ifdef RECYCLER\_WRITE\_BARRIER

|| type == HeapBlock::HeapBlockType::SmallNormalBlockWithBarrierType

|| type == HeapBlock::HeapBlockType::SmallFinalizableBlockWithBarrierType

#endif

|| type == HeapBlock::HeapBlockType::MediumNormalBlockType

|| type == HeapBlock::HeapBlockType::MediumFinalizableBlockType

#ifdef RECYCLER\_WRITE\_BARRIER

|| type == HeapBlock::HeapBlockType::MediumNormalBlockWithBarrierType

|| type == HeapBlock::HeapBlockType::MediumFinalizableBlockWithBarrierType

#endif

))

{

Output::Print(L" %10d", partialUnusedBytes);

}

else

#endif

{

Output::Print(L" ");

}

Output::Print(L" %10d %6.1f", totalByteCount,

(double)allocableFreeByteCount / (double)totalByteCount \* 100);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect &&

(type == HeapBlock::HeapBlockType::SmallNormalBlockType

|| type == HeapBlock::HeapBlockType::SmallFinalizableBlockType

#ifdef RECYCLER\_WRITE\_BARRIER

|| type == HeapBlock::HeapBlockType::SmallNormalBlockWithBarrierType

|| type == HeapBlock::HeapBlockType::SmallFinalizableBlockWithBarrierType

#endif

|| type == HeapBlock::HeapBlockType::MediumNormalBlockType

|| type == HeapBlock::HeapBlockType::MediumFinalizableBlockType

#ifdef RECYCLER\_WRITE\_BARRIER

|| type == HeapBlock::HeapBlockType::MediumNormalBlockWithBarrierType

|| type == HeapBlock::HeapBlockType::MediumFinalizableBlockWithBarrierType

#endif

))

{

Output::Print(L" %6.1f", (double)partialUnusedBytes / (double)totalByteCount \* 100);

}

#endif

}

void

Recycler::PrintHeuristicCollectionStats()

{

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

Output::Print(L"GC Trigger : %10s %10s %10s", L"Start", L"Continue", L"Finish");

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Heuristics : %10s %10s %5s", L"", L"", L"%");

}

#endif

Output::Print(L"\n");

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

Output::Print(L" Alloc bytes : %10d %10d %10d", collectionStats.startCollectAllocBytes, collectionStats.continueCollectAllocBytes, this->autoHeap.uncollectedAllocBytes);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Cost : %10d %10d %5.1f", collectionStats.rescanRootBytes, collectionStats.estimatedPartialReuseBytes, collectionStats.collectCost \* 100);

}

#endif

Output::Print(L"\n");

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Efficacy : %10s %10s %5.1f\n", L"", L"", collectionStats.collectEfficacy \* 100);

}

#endif

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" New page : %10d %10s %10d", collectionStats.startCollectNewPageCount, L"", autoHeap.uncollectedNewPageCount);

Output::Print(L" | Partial Uncollect New Page : %10d %10d", collectionStats.uncollectedNewPageCountPartialCollect \* AutoSystemInfo::PageSize, this->uncollectedNewPageCountPartialCollect \* AutoSystemInfo::PageSize);

Output::Print(L"\n");

}

#endif

Output::Print(L" Finish try : %10d %10s %10s", collectionStats.finishCollectTryCount, L"", L"");

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Partial Reuse Min Free Bytes : %10d", collectionStats.partialCollectSmallHeapBlockReuseMinFreeBytes \* AutoSystemInfo::PageSize);

}

#endif

Output::Print(L"\n");

}

void

Recycler::PrintMarkCollectionStats()

{

size\_t nonMark = collectionStats.tryMarkCount + collectionStats.tryMarkInteriorCount - collectionStats.remarkCount - collectionStats.markData.markCount;

size\_t invalidCount = nonMark - collectionStats.tryMarkNullCount - collectionStats.tryMarkUnalignedCount

- collectionStats.tryMarkNonRecyclerMemoryCount

- collectionStats.tryMarkInteriorNonRecyclerMemoryCount

- collectionStats.tryMarkInteriorNullCount;

size\_t leafCount = collectionStats.markData.markCount - collectionStats.scanCount;

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

Output::Print(L"Try Mark :%9s %5s %10s | Non-Mark : %9s %5s | Mark :%9s %5s \n", L"Count", L"%", L"Bytes", L"Count", L"%", L"Count", L"%");

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

Output::Print(L" TryMark :%9d %10d | Null : %9d %5.1f | Scan :%9d %5.1f\n",

collectionStats.tryMarkCount, collectionStats.tryMarkCount \* sizeof(void \*),

collectionStats.tryMarkNullCount, (double)collectionStats.tryMarkNullCount / (double)nonMark \* 100,

collectionStats.scanCount, (double)collectionStats.scanCount / (double)collectionStats.markData.markCount \* 100);

Output::Print(L" Non-Mark :%9d %5.1f | Unaligned : %9d %5.1f | Leaf :%9d %5.1f\n",

nonMark, (double)nonMark / (double)collectionStats.tryMarkCount \* 100,

collectionStats.tryMarkUnalignedCount, (double)collectionStats.tryMarkUnalignedCount / (double)nonMark \* 100,

leafCount, (double)leafCount / (double)collectionStats.markData.markCount \* 100);

Output::Print(L" Mark :%9d %5.1f %10d | Non GC : %9d %5.1f | Track :%9d\n",

collectionStats.markData.markCount, (double)collectionStats.markData.markCount / (double)collectionStats.tryMarkCount \* 100, collectionStats.markData.markBytes,

collectionStats.tryMarkNonRecyclerMemoryCount, (double)collectionStats.tryMarkNonRecyclerMemoryCount / (double)nonMark \* 100,

collectionStats.trackCount);

Output::Print(L" Remark :%9d %5.1f | Invalid : %9d %5.1f \n",

collectionStats.remarkCount, (double)collectionStats.remarkCount / (double)collectionStats.tryMarkCount \* 100,

invalidCount, (double)invalidCount / (double)nonMark \* 100);

Output::Print(L" TryMark Int:%9d %10d | Null Int : %9d %5.1f | Root :%9d | New :%9d\n",

collectionStats.tryMarkInteriorCount, collectionStats.tryMarkInteriorCount \* sizeof(void \*),

collectionStats.tryMarkInteriorNullCount, (double)collectionStats.tryMarkInteriorNullCount / (double)nonMark \* 100,

collectionStats.rootCount, collectionStats.markThruNewObjCount);

Output::Print(L" | Non GC Int: %9d %5.1f | Stack :%9d | NewFalse:%9d\n",

collectionStats.tryMarkInteriorNonRecyclerMemoryCount, (double)collectionStats.tryMarkInteriorNonRecyclerMemoryCount / (double)nonMark \* 100,

collectionStats.stackCount, collectionStats.markThruFalseNewObjCount);

}

void

Recycler::PrintBackgroundCollectionStat(RecyclerCollectionStats::MarkData const& markData)

{

Output::Print(L"BgSmall : %5d %6d %10d | BgLarge : %5d %6d %10d | BgMark :%9d ",

markData.rescanPageCount,

markData.rescanObjectCount,

markData.rescanObjectByteCount,

markData.rescanLargePageCount,

markData.rescanLargeObjectCount,

markData.rescanLargeByteCount,

markData.markCount);

double markRatio = (double)markData.markCount / (double)collectionStats.markData.markCount \* 100;

if (markRatio == 100.0)

{

Output::Print(L" 100");

}

else

{

Output::Print(L"%4.1f", markRatio);

}

Output::Print(L"\n");

}

void

Recycler::PrintBackgroundCollectionStats()

{

#ifdef CONCURRENT\_GC\_ENABLED

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

Output::Print(L"BgSmall : %5s %6s %10s | BgLarge : %5s %6s %10s | BgMark :%9s %4s %s\n",

L"Pages", L"Count", L"Bytes", L"Pages", L"Count", L"Bytes", L"Count", L"%", L"NonLeafBytes %");

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

this->PrintBackgroundCollectionStat(collectionStats.backgroundMarkData[0]);

for (uint repeatCount = 1; repeatCount < RecyclerHeuristic::MaxBackgroundRepeatMarkCount; repeatCount++)

{

if (collectionStats.backgroundMarkData[repeatCount].markCount == 0)

{

break;

}

collectionStats.backgroundMarkData[repeatCount].rescanPageCount -= collectionStats.backgroundMarkData[repeatCount - 1].rescanPageCount;

collectionStats.backgroundMarkData[repeatCount].rescanObjectCount -= collectionStats.backgroundMarkData[repeatCount - 1].rescanObjectCount;

collectionStats.backgroundMarkData[repeatCount].rescanObjectByteCount -= collectionStats.backgroundMarkData[repeatCount - 1].rescanObjectByteCount;

collectionStats.backgroundMarkData[repeatCount].rescanLargePageCount -= collectionStats.backgroundMarkData[repeatCount - 1].rescanLargePageCount;

collectionStats.backgroundMarkData[repeatCount].rescanLargeObjectCount -= collectionStats.backgroundMarkData[repeatCount - 1].rescanLargeObjectCount;

collectionStats.backgroundMarkData[repeatCount].rescanLargeByteCount -= collectionStats.backgroundMarkData[repeatCount - 1].rescanLargeByteCount;

this->PrintBackgroundCollectionStat(collectionStats.backgroundMarkData[repeatCount]);

}

#endif

}

void

Recycler::PrintMemoryStats()

{

Output::Print(L"----------------------------------------------------------------------------------------------------------------\n");

Output::Print(L"Memory (Bytes) %4s %10s %10s %10s %6s %6s\n", L"Live", L"Free", L"Unused", L"Total", L"Free%", L"Unused%");

Output::Print(L"----------------------------------------------------------------------------------------------------------------\n");

PrintHeapBlockMemoryStats(L"Small", HeapBlock::SmallNormalBlockType);

Output::Print(L"\n");

PrintHeapBlockMemoryStats(L"SmFin", HeapBlock::SmallFinalizableBlockType);

Output::Print(L"\n");

#ifdef RECYCLER\_WRITE\_BARRIER

PrintHeapBlockMemoryStats(L"SmSWB", HeapBlock::SmallNormalBlockWithBarrierType);

Output::Print(L"\n");

PrintHeapBlockMemoryStats(L"SmFinSWB", HeapBlock::SmallFinalizableBlockWithBarrierType);

Output::Print(L"\n");

#endif

PrintHeapBlockMemoryStats(L"SmLeaf", HeapBlock::SmallLeafBlockType);

Output::Print(L"\n");

PrintHeapBlockMemoryStats(L"Medium", HeapBlock::MediumNormalBlockType);

Output::Print(L"\n");

PrintHeapBlockMemoryStats(L"MdFin", HeapBlock::MediumFinalizableBlockType);

Output::Print(L"\n");

#ifdef RECYCLER\_WRITE\_BARRIER

PrintHeapBlockMemoryStats(L"MdSWB", HeapBlock::MediumNormalBlockWithBarrierType);

Output::Print(L"\n");

PrintHeapBlockMemoryStats(L"MdFinSWB", HeapBlock::MediumFinalizableBlockWithBarrierType);

Output::Print(L"\n");

#endif

PrintHeapBlockMemoryStats(L"MdLeaf", HeapBlock::MediumLeafBlockType);

Output::Print(L"\n");

size\_t largeHeapBlockUnusedByteCount = collectionStats.largeHeapBlockTotalByteCount - collectionStats.largeHeapBlockUsedByteCount

- collectionStats.heapBlockFreeByteCount[HeapBlock::LargeBlockType];

Output::Print(L" Large: %10d %10d %10d %10d %6.1f %6.1f\n",

collectionStats.largeHeapBlockUsedByteCount,

collectionStats.heapBlockFreeByteCount[HeapBlock::LargeBlockType],

largeHeapBlockUnusedByteCount,

collectionStats.largeHeapBlockTotalByteCount,

(double)collectionStats.heapBlockFreeByteCount[HeapBlock::LargeBlockType] / (double)collectionStats.largeHeapBlockTotalByteCount \* 100,

(double)largeHeapBlockUnusedByteCount / (double)collectionStats.largeHeapBlockTotalByteCount \* 100);

Output::Print(L"\nSmall heap block zeroing stats since last GC\n");

Output::Print(L"Number of blocks with sweep state empty: normal=%d finalizable=%d leaf=%d\nNumber of blocks zeroed: %d\n",

collectionStats.numEmptySmallBlocks[HeapBlock::SmallNormalBlockType]

#ifdef RECYCLER\_WRITE\_BARRIER

+ collectionStats.numEmptySmallBlocks[HeapBlock::SmallNormalBlockWithBarrierType]

#endif

, collectionStats.numEmptySmallBlocks[HeapBlock::SmallFinalizableBlockType]

#ifdef RECYCLER\_WRITE\_BARRIER

+ collectionStats.numEmptySmallBlocks[HeapBlock::SmallFinalizableBlockWithBarrierType]

#endif

+ collectionStats.numEmptySmallBlocks[HeapBlock::MediumNormalBlockType]

#ifdef RECYCLER\_WRITE\_BARRIER

+ collectionStats.numEmptySmallBlocks[HeapBlock::MediumNormalBlockWithBarrierType]

#endif

, collectionStats.numEmptySmallBlocks[HeapBlock::MediumFinalizableBlockType]

#ifdef RECYCLER\_WRITE\_BARRIER

+ collectionStats.numEmptySmallBlocks[HeapBlock::MediumFinalizableBlockWithBarrierType]

#endif

, collectionStats.numEmptySmallBlocks[HeapBlock::SmallLeafBlockType]

+ collectionStats.numEmptySmallBlocks[HeapBlock::MediumLeafBlockType],

collectionStats.numZeroedOutSmallBlocks);

}

void

Recycler::PrintCollectStats()

{

Output::Print(L"Collection Stats:\n");

PrintHeuristicCollectionStats();

PrintMarkCollectionStats();

PrintBackgroundCollectionStats();

size\_t freeCount = collectionStats.objectSweptCount - collectionStats.objectSweptFreeListCount;

size\_t freeBytes = collectionStats.objectSweptBytes - collectionStats.objectSweptFreeListBytes;

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

Output::Print(L"Rescan : %5s %6s %10s | Track : %5s | ", L"Pages", L"Count", L"Bytes", L"Count");

#endif

Output::Print(L"Sweep : %7s | SweptObj : %5s %5s %10s\n", L"Count", L"Count", L"%%", L"Bytes");

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

Output::Print(L" Small : ");

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

Output::Print(L"%5d %6d %10d | ", collectionStats.markData.rescanPageCount, collectionStats.markData.rescanObjectCount, collectionStats.markData.rescanObjectByteCount);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

Output::Print(L"Process : %5d | ", collectionStats.trackedObjectCount);

#else

Output::Print(L" | ");

#endif

Output::Print(L" Scan : %7d | Free : %6d %5.1f %10d\n",

collectionStats.objectSweepScanCount,

freeCount, (double)freeCount / (double) collectionStats.objectSweptCount \* 100, freeBytes);

Output::Print(L" Large : ");

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

Output::Print(L"%5d %6d %10d | ",

collectionStats.markData.rescanLargePageCount, collectionStats.markData.rescanLargeObjectCount, collectionStats.markData.rescanLargeByteCount);

#endif

#ifdef PARTIAL\_GC\_ENABLED

Output::Print(L"Client : %5d | ", collectionStats.clientTrackedObjectCount);

#else

Output::Print(L" | ");

#endif

Output::Print(L" Finalize : %7d | Free List: %6d %5.1f %10d\n",

collectionStats.finalizeSweepCount,

collectionStats.objectSweptFreeListCount, (double)collectionStats.objectSweptFreeListCount / (double) collectionStats.objectSweptCount \* 100, collectionStats.objectSweptFreeListBytes);

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

Output::Print(L"SweptBlk: Live Free Total Free%% : Swept Swept%% : CSwpt CSwpt%%");

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Partial : Count Bytes Existing");

}

#endif

Output::Print(L"\n");

Output::Print(L"---------------------------------------------------------------------------------------------------------------\n");

PrintHeapBlockStats(L"Small", HeapBlock::SmallNormalBlockType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Reuse : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialReuseCount[HeapBlock::SmallNormalBlockType],

collectionStats.smallNonLeafHeapBlockPartialReuseBytes[HeapBlock::MediumNormalBlockType],

collectionStats.smallNonLeafHeapBlockPartialReuseCount[HeapBlock::SmallNormalBlockType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialReuseBytes[HeapBlock::SmallNormalBlockType]);

}

#endif

Output::Print(L"\n");

PrintHeapBlockStats(L"SmFin", HeapBlock::SmallFinalizableBlockType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Unused : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::SmallFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::SmallFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::SmallFinalizableBlockType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::SmallFinalizableBlockType]);

}

#endif

Output::Print(L"\n");

#ifdef RECYCLER\_WRITE\_BARRIER

PrintHeapBlockStats(L"SmSWB", HeapBlock::SmallNormalBlockWithBarrierType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Unused : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::SmallNormalBlockWithBarrierType],

collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::SmallNormalBlockWithBarrierType],

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::SmallNormalBlockWithBarrierType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::SmallNormalBlockWithBarrierType]);

}

#endif

Output::Print(L"\n");

PrintHeapBlockStats(L"SmFin", HeapBlock::SmallFinalizableBlockWithBarrierType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Unused : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::SmallFinalizableBlockWithBarrierType],

collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::SmallFinalizableBlockWithBarrierType],

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::SmallFinalizableBlockWithBarrierType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::SmallFinalizableBlockWithBarrierType]);

}

#endif

Output::Print(L"\n");

#endif

// TODO: This seems suspicious- why are we looking at smallNonLeaf while print out leaf...

PrintHeapBlockStats(L"SmLeaf", HeapBlock::SmallLeafBlockType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | ReuseFin : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialReuseCount[HeapBlock::SmallFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialReuseBytes[HeapBlock::SmallFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialReuseCount[HeapBlock::SmallFinalizableBlockType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialReuseBytes[HeapBlock::SmallFinalizableBlockType]);

}

#endif

Output::Print(L"\n");

PrintHeapBlockStats(L"Medium", HeapBlock::MediumNormalBlockType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Reuse : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialReuseCount[HeapBlock::MediumNormalBlockType],

collectionStats.smallNonLeafHeapBlockPartialReuseBytes[HeapBlock::MediumNormalBlockType],

collectionStats.smallNonLeafHeapBlockPartialReuseCount[HeapBlock::MediumNormalBlockType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialReuseBytes[HeapBlock::MediumNormalBlockType]);

}

#endif

Output::Print(L"\n");

PrintHeapBlockStats(L"MdFin", HeapBlock::MediumFinalizableBlockType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Unused : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::MediumFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::MediumFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::MediumFinalizableBlockType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::MediumFinalizableBlockType]);

}

#endif

Output::Print(L"\n");

#ifdef RECYCLER\_WRITE\_BARRIER

PrintHeapBlockStats(L"MdSWB", HeapBlock::MediumNormalBlockWithBarrierType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Unused : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::MediumNormalBlockWithBarrierType],

collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::MediumNormalBlockWithBarrierType],

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::MediumNormalBlockWithBarrierType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::MediumNormalBlockWithBarrierType]);

}

#endif

Output::Print(L"\n");

PrintHeapBlockStats(L"MdFin", HeapBlock::MediumFinalizableBlockWithBarrierType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | Unused : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::MediumFinalizableBlockWithBarrierType],

collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::MediumFinalizableBlockWithBarrierType],

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::MediumFinalizableBlockWithBarrierType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::MediumFinalizableBlockWithBarrierType]);

}

#endif

Output::Print(L"\n");

#endif

// TODO: This seems suspicious- why are we looking at smallNonLeaf while print out leaf...

PrintHeapBlockStats(L"MdLeaf", HeapBlock::MediumNormalBlockType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | ReuseFin : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialReuseCount[HeapBlock::MediumFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialReuseBytes[HeapBlock::MediumFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialReuseCount[HeapBlock::MediumFinalizableBlockType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialReuseBytes[HeapBlock::MediumFinalizableBlockType]);

}

#endif

Output::Print(L"\n");

// TODO: This can't possibly be correct...check on this later

PrintHeapBlockStats(L"Large", HeapBlock::LargeBlockType);

#ifdef PARTIAL\_GC\_ENABLED

if (this->enablePartialCollect)

{

Output::Print(L" | UnusedFin : %5d %10d %10d",

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::SmallFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::SmallFinalizableBlockType],

collectionStats.smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::SmallFinalizableBlockType] \* AutoSystemInfo::PageSize

- collectionStats.smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::SmallFinalizableBlockType]);

}

#endif

Output::Print(L"\n");

PrintMemoryStats();

Output::Flush();

}

#endif

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

void

Recycler::VerifyZeroFill(void \* address, size\_t size)

{

byte expectedFill = 0;

#ifdef RECYCLER\_MEMORY\_VERIFY

if (this->VerifyEnabled())

{

expectedFill = Recycler::VerifyMemFill;

}

#endif

for (uint i = 0; i < size; i++)

{

Assert(((byte \*)address)[i] == expectedFill);

}

}

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void

Recycler::FillCheckPad(void \* address, size\_t size, size\_t alignedAllocSize, bool objectAlreadyInitialized)

{

if (this->VerifyEnabled())

{

void\* addressToVerify = address;

size\_t sizeToVerify = alignedAllocSize;

if (objectAlreadyInitialized)

{

addressToVerify = ((char\*) address + size);

sizeToVerify = (alignedAllocSize - size);

}

// Actually this is filling the non-pad to zero

VerifyCheckFill(addressToVerify, sizeToVerify - sizeof(size\_t));

// Ignore the first word

if (!objectAlreadyInitialized && size > sizeof(FreeObject))

{

memset((char \*)address + sizeof(FreeObject), 0, size - sizeof(FreeObject));

}

// write the pad size at the end;

\*(size\_t \*)((char \*)address + alignedAllocSize - sizeof(size\_t)) = alignedAllocSize - size;

}

}

void Recycler::Verify(Js::Phase phase)

{

if (verifyEnabled && (!this->CollectionInProgress()))

{

if (GetRecyclerFlagsTable().RecyclerVerify.IsEnabled(phase))

{

autoHeap.Verify();

}

}

}

void Recycler::VerifyCheck(BOOL cond, wchar\_t const \* msg, void \* address, void \* corruptedAddress)

{

if (!(cond))

{

fwprintf(stderr, L"RECYCLER CORRUPTION: StartAddress=%p CorruptedAddress=%p: %s", address, corruptedAddress, msg);

Js::Throw::FatalInternalError();

}

}

void Recycler::VerifyCheckFill(void \* address, size\_t size)

{

for (byte \* i = (byte \*)address; i < (byte \*)address + size; i++)

{

Recycler::VerifyCheck(\*i == Recycler::VerifyMemFill, L"memory written after freed", address, i);

}

}

void Recycler::VerifyCheckPadExplicitFreeList(void \* address, size\_t size)

{

size\_t \* paddingAddress = (size\_t \*)((byte \*)address + size - sizeof(size\_t));

size\_t padding = \*paddingAddress;

#pragma warning(suppress:4310)

Assert(padding != (size\_t)0xCACACACACACACACA); // Explicit free objects have to have been initialized at some point before they were freed

Recycler::VerifyCheck(padding >= verifyPad + sizeof(size\_t) && padding < size, L"Invalid padding size", address, paddingAddress);

for (byte \* i = (byte \*)address + size - padding; i < (byte \*)paddingAddress; i++)

{

Recycler::VerifyCheck(\*i == Recycler::VerifyMemFill, L"buffer overflow", address, i);

}

}

void Recycler::VerifyCheckPad(void \* address, size\_t size)

{

size\_t \* paddingAddress = (size\_t \*)((byte \*)address + size - sizeof(size\_t));

size\_t padding = \*paddingAddress;

#pragma warning(suppress:4310)

if (padding == (size\_t)0xCACACACACACACACA)

{

// Nascent block have objects that are not initialized with pad size

Recycler::VerifyCheckFill(address, size);

return;

}

Recycler::VerifyCheck(padding >= verifyPad + sizeof(size\_t) && padding < size, L"Invalid padding size", address, paddingAddress);

for (byte \* i = (byte \*)address + size - padding; i < (byte \*)paddingAddress; i++)

{

Recycler::VerifyCheck(\*i == Recycler::VerifyMemFill, L"buffer overflow", address, i);

}

}

#endif

Recycler::AutoSetupRecyclerForNonCollectingMark::AutoSetupRecyclerForNonCollectingMark(Recycler& recycler, bool setupForHeapEnumeration)

: m\_recycler(recycler), m\_setupDone(false)

{

if (! setupForHeapEnumeration)

{

DoCommonSetup();

}

}

void Recycler::AutoSetupRecyclerForNonCollectingMark::DoCommonSetup()

{

Assert(m\_recycler.collectionState == CollectionStateNotCollecting || m\_recycler.collectionState == CollectionStateExit);

Assert(!m\_recycler.DoQueueTrackedObject());

#ifdef PARTIAL\_GC\_ENABLED

// We need to get out of partial collect before we do the mark because we

// will mess with the free bit vector state

// GC-CONSIDER: don't mess with the free bit vector?

if (m\_recycler.inPartialCollectMode)

{

m\_recycler.FinishPartialCollect();

}

#endif

m\_previousCollectionState = m\_recycler.collectionState;

#ifdef RECYCLER\_STATS

m\_previousCollectionStats = m\_recycler.collectionStats;

memset(&m\_recycler.collectionStats, 0, sizeof(RecyclerCollectionStats));

#endif

m\_setupDone = true;

}

void Recycler::AutoSetupRecyclerForNonCollectingMark::SetupForHeapEnumeration()

{

Assert(!m\_recycler.isHeapEnumInProgress);

Assert(!m\_recycler.allowAllocationDuringHeapEnum);

m\_recycler.EnsureNotCollecting();

DoCommonSetup();

m\_recycler.ResetMarks(ResetMarkFlags\_HeapEnumeration);

m\_recycler.collectionState = CollectionStateNotCollecting;

m\_recycler.isHeapEnumInProgress = true;

m\_recycler.isCollectionDisabled = true;

}

Recycler::AutoSetupRecyclerForNonCollectingMark::~AutoSetupRecyclerForNonCollectingMark()

{

Assert(m\_setupDone);

Assert(!m\_recycler.allowAllocationDuringHeapEnum);

#ifdef RECYCLER\_STATS

m\_recycler.collectionStats = m\_previousCollectionStats;

#endif

m\_recycler.collectionState = m\_previousCollectionState;

m\_recycler.isHeapEnumInProgress = false;

m\_recycler.isCollectionDisabled = false;

}

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

bool Recycler::DumpObjectGraph(RecyclerObjectGraphDumper::Param \* param)

{

bool succeeded = false;

bool isExited = (this->collectionState == CollectionStateExit);

if (isExited)

{

this->collectionState = CollectionStateNotCollecting;

}

if (this->collectionState != CollectionStateNotCollecting)

{

Output::Print(L"Can't dump object graph when collecting\n");

Output::Flush();

return succeeded;

}

BEGIN\_NO\_EXCEPTION

{

RecyclerObjectGraphDumper objectGraphDumper(this, param);

Recycler::AutoSetupRecyclerForNonCollectingMark AutoSetupRecyclerForNonCollectingMark(\*this);

AutoRestoreValue<bool> skipStackToggle(&this->skipStack, this->skipStack || (param && param->skipStack));

this->Mark();

this->objectGraphDumper = nullptr;

#ifdef RECYCLER\_STATS

if (param)

{

param->stats = this->collectionStats;

}

#endif

succeeded = !objectGraphDumper.isOutOfMemory;

}

END\_NO\_EXCEPTION

if (isExited)

{

this->collectionState = CollectionStateExit;

}

if (!succeeded)

{

Output::Print(L"Out of memory dumping object graph\n");

}

Output::Flush();

return succeeded;

}

void

Recycler::DumpObjectDescription(void \*objectAddress)

{

#ifdef PROFILE\_RECYCLER\_ALLOC

type\_info const \* typeinfo = nullptr;

bool isArray = false;

if (this->trackerDictionary)

{

TrackerData \* trackerData = GetTrackerData(objectAddress);

if (trackerData != nullptr)

{

typeinfo = trackerData->typeinfo;

isArray = trackerData->isArray;

}

else

{

Assert(false);

}

}

RecyclerObjectDumper::DumpObject(typeinfo, isArray, objectAddress);

#else

Output::Print(L"Address %p", objectAddress);

#endif

}

#endif

#ifdef RECYCLER\_STRESS

// All stress mode collect art implicitly instantiate here

bool

Recycler::StressCollectNow()

{

if (this->recyclerStress)

{

this->CollectNow<CollectStress>();

return true;

}

#ifdef CONCURRENT\_GC\_ENABLED

else if (this->recyclerBackgroundStress)

{

this->CollectNow<CollectBackgroundStress>();

return true;

}

else if ((this->enableConcurrentMark || this->enableConcurrentSweep)

&& (this->recyclerConcurrentStress

|| this->recyclerConcurrentRepeatStress))

{

#ifdef PARTIAL\_GC\_ENABLED

if (this->recyclerPartialStress)

{

this->CollectNow<CollectConcurrentPartialStress>();

return true;

}

else

#endif // PARTIAL\_GC\_ENABLED

{

this->CollectNow<CollectConcurrentStress>();

return true;

}

}

#endif // CONCURRENT\_GC\_ENABLED

#ifdef PARTIAL\_GC\_ENABLED

else if (this->recyclerPartialStress)

{

this->CollectNow<CollectPartialStress>();

return true;

}

#endif // PARTIAL\_GC\_ENABLED

return false;

}

#endif // RECYCLER\_STRESS

#ifdef TRACK\_ALLOC

Recycler \*

Recycler::TrackAllocInfo(TrackAllocData const& data)

{

#ifdef PROFILE\_RECYCLER\_ALLOC

if (this->trackerDictionary != nullptr)

{

Assert(nextAllocData.IsEmpty());

nextAllocData = data;

}

#endif

return this;

}

void

Recycler::ClearTrackAllocInfo(TrackAllocData\* data/\* = NULL\*/)

{

#ifdef PROFILE\_RECYCLER\_ALLOC

if (this->trackerDictionary != nullptr)

{

AssertMsg(!nextAllocData.IsEmpty(), "Missing tracking information for this allocation, are you not using the macros?");

if (data)

{

\*data = nextAllocData;

}

nextAllocData.Clear();

}

#endif

}

#ifdef PROFILE\_RECYCLER\_ALLOC

bool

Recycler::DoProfileAllocTracker()

{

bool doTracker = false;

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

doTracker = Js::Configuration::Global.flags.DumpObjectGraphOnExit

|| Js::Configuration::Global.flags.DumpObjectGraphOnCollect

|| Js::Configuration::Global.flags.DumpObjectGraphOnEnum;

#endif

#ifdef LEAK\_REPORT

if (Js::Configuration::Global.flags.IsEnabled(Js::LeakReportFlag))

{

doTracker = true;

}

#endif

#ifdef CHECK\_MEMORY\_LEAK

if (Js::Configuration::Global.flags.CheckMemoryLeak)

{

doTracker = true;

}

#endif

return doTracker || MemoryProfiler::DoTrackRecyclerAllocation();

}

void

Recycler::InitializeProfileAllocTracker()

{

if (DoProfileAllocTracker())

{

trackerDictionary = NoCheckHeapNew(TypeInfotoTrackerItemMap, &NoCheckHeapAllocator::Instance, 163);

#pragma prefast(suppress:6031, "InitializeCriticalSectionAndSpinCount always succeed since Vista. No need to check return value");

InitializeCriticalSectionAndSpinCount(&trackerCriticalSection, 1000);

}

nextAllocData.Clear();

}

void

Recycler::TrackAllocCore(void \* object, size\_t size, const TrackAllocData& trackAllocData, bool traceLifetime)

{

Assert(GetTrackerData(object) == nullptr || GetTrackerData(object) == &TrackerData::ExplicitFreeListObjectData);

Assert(trackAllocData.GetTypeInfo() != nullptr);

TrackerItem \* item;

size\_t allocCount = trackAllocData.GetCount();

size\_t itemSize = (size - trackAllocData.GetPlusSize());

bool isArray;

if (allocCount != (size\_t)-1)

{

isArray = true;

itemSize = itemSize / allocCount;

}

else

{

isArray = false;

allocCount = 1;

}

if (!trackerDictionary->TryGetValue(trackAllocData.GetTypeInfo(), &item))

{

item = NoCheckHeapNew(TrackerItem, trackAllocData.GetTypeInfo());

item->instanceData.ItemSize = itemSize;

item->arrayData.ItemSize = itemSize;

trackerDictionary->Item(trackAllocData.GetTypeInfo(), item);

}

else

{

Assert(item->instanceData.typeinfo == trackAllocData.GetTypeInfo());

Assert(item->instanceData.ItemSize == itemSize);

Assert(item->arrayData.ItemSize == itemSize);

}

TrackerData& data = (isArray)? item->arrayData : item->instanceData;

data.ItemCount += allocCount;

data.AllocCount++;

data.ReqSize += size;

data.AllocSize += HeapInfo::GetAlignedSizeNoCheck(size);

#ifdef TRACE\_OBJECT\_LIFETIME

data.TraceLifetime = traceLifetime;

if (traceLifetime)

{

Output::Print(data.isArray ? L"Allocated %S[] %p\n" : L"Allocated %S %p\n", data.typeinfo->name(), object);

}

#endif

#ifdef PERF\_COUNTERS

++data.counter;

data.sizeCounter += HeapInfo::GetAlignedSizeNoCheck(size);

#endif

SetTrackerData(object, &data);

}

void\* Recycler::TrackAlloc(void\* object, size\_t size, const TrackAllocData& trackAllocData, bool traceLifetime)

{

if (this->trackerDictionary != nullptr)

{

Assert(nextAllocData.IsEmpty()); // should have been cleared

EnterCriticalSection(&trackerCriticalSection);

TrackAllocCore(object, size, trackAllocData);

LeaveCriticalSection(&trackerCriticalSection);

}

return object;

}

void

Recycler::TrackIntegrate(\_\_in\_ecount(blockSize) char \* blockAddress, size\_t blockSize, size\_t allocSize, size\_t objectSize, const TrackAllocData& trackAllocData)

{

if (this->trackerDictionary != nullptr)

{

Assert(nextAllocData.IsEmpty()); // should have been cleared

EnterCriticalSection(&trackerCriticalSection);

char \* address = blockAddress;

char \* blockEnd = blockAddress + blockSize;

while (address + allocSize <= blockEnd)

{

TrackAllocCore(address, objectSize, trackAllocData);

address += allocSize;

}

LeaveCriticalSection(&trackerCriticalSection);

}

}

BOOL Recycler::TrackFree(const char\* address, size\_t size)

{

if (this->trackerDictionary != nullptr)

{

EnterCriticalSection(&trackerCriticalSection);

TrackerData \* data = GetTrackerData((char \*)address);

if (data != nullptr)

{

if (data != &TrackerData::EmptyData)

{

#ifdef PERF\_COUNTERS

--data->counter;

data->sizeCounter -= size;

#endif

if (data->typeinfo == &typeid(RecyclerWeakReferenceBase))

{

TrackFreeWeakRef((RecyclerWeakReferenceBase \*)address);

}

data->FreeSize += size;

data->FreeCount++;

#ifdef TRACE\_OBJECT\_LIFETIME

if (data->TraceLifetime)

{

Output::Print(data->isArray ? L"Freed %S[] %p\n" : L"Freed %S %p\n", data->typeinfo->name(), address);

}

#endif

}

SetTrackerData((char \*)address, nullptr);

}

else

{

Assert(false);

}

LeaveCriticalSection(&trackerCriticalSection);

}

return true;

}

Recycler::TrackerData \*

Recycler::GetTrackerData(void \* address)

{

HeapBlock \* heapBlock = this->FindHeapBlock(address);

Assert(heapBlock != nullptr);

return (Recycler::TrackerData \*)heapBlock->GetTrackerData(address);

}

void

Recycler::SetTrackerData(void \* address, TrackerData \* data)

{

HeapBlock \* heapBlock = this->FindHeapBlock(address);

Assert(heapBlock != nullptr);

heapBlock->SetTrackerData(address, data);

}

void

Recycler::TrackUnallocated(\_\_in char\* address, \_\_in char \*endAddress, size\_t sizeCat)

{

if (this->trackerDictionary != nullptr)

{

EnterCriticalSection(&trackerCriticalSection);

while (address + sizeCat <= endAddress)

{

Assert(GetTrackerData(address) == nullptr);

SetTrackerData(address, &TrackerData::EmptyData);

address += sizeCat;

}

LeaveCriticalSection(&trackerCriticalSection);

}

}

void

Recycler::TrackAllocWeakRef(RecyclerWeakReferenceBase \* weakRef)

{

Assert(weakRef->typeInfo != nullptr);

#if DBG && defined(PERF\_COUNTERS)

if (this->trackerDictionary != nullptr)

{

TrackerItem \* item;

if (trackerDictionary->TryGetValue(weakRef->typeInfo, &item))

{

weakRef->counter = &item->weakRefCounter;

}

else

{

weakRef->counter = &PerfCounter::RecyclerTrackerCounterSet::GetWeakRefPerfCounter(weakRef->typeInfo);

}

++(\*weakRef->counter);

}

#endif

}

void

Recycler::TrackFreeWeakRef(RecyclerWeakReferenceBase \* weakRef)

{

#if DBG && defined(PERF\_COUNTERS)

if (weakRef->counter != nullptr)

{

--(\*weakRef->counter);

}

#endif

}

void

Recycler::PrintAllocStats()

{

if (this->trackerDictionary == nullptr)

{

return;

}

size\_t itemCount = 0;

int allocCount = 0;

int64 reqSize = 0;

int64 allocSize = 0;

int freeCount = 0;

int64 freeSize = 0;

Output::Print(L"=================================================================================================================\n");

Output::Print(L"Recycler Allocations\n");

Output::Print(L"=================================================================================================================\n");

Output::Print(L"ItemSize ItemCount AllocCount RequestSize AllocSize FreeCount FreeSize DiffCount DiffSize \n");

Output::Print(L"-------- ---------- ---------- --------------- --------------- ---------- --------------- ---------- ---------------\n");

for (int i = 0; i < trackerDictionary->Count(); i++)

{

TrackerItem \* item = trackerDictionary->GetValueAt(i);

type\_info const \* typeinfo = trackerDictionary->GetKeyAt(i);

if (item->instanceData.AllocCount != 0)

{

Output::Print(L"%8d %10d %10d %15I64d %15I64d %10d %15I64d %10d %15I64d %S\n",

item->instanceData.ItemSize, item->instanceData.ItemCount, item->instanceData.AllocCount, item->instanceData.ReqSize,

item->instanceData.AllocSize, item->instanceData.FreeCount, item->instanceData.FreeSize,

item->instanceData.AllocCount - item->instanceData.FreeCount, item->instanceData.AllocSize - item->instanceData.FreeSize, typeinfo->name());

itemCount += item->instanceData.ItemCount;

allocCount += item->instanceData.AllocCount;

reqSize += item->instanceData.ReqSize;

allocSize += item->instanceData.AllocSize;

freeCount += item->instanceData.FreeCount;

freeSize += item->instanceData.FreeSize;

}

if (item->arrayData.AllocCount != 0)

{

Output::Print(L"%8d %10d %10d %15I64d %15I64d %10d %15I64d %10d %15I64d %S[]\n",

item->arrayData.ItemSize, item->arrayData.ItemCount, item->arrayData.AllocCount, item->arrayData.ReqSize,

item->arrayData.AllocSize, item->arrayData.FreeCount, item->arrayData.FreeSize,

item->instanceData.AllocCount - item->instanceData.FreeCount, item->arrayData.AllocSize - item->arrayData.FreeSize, typeinfo->name());

itemCount += item->arrayData.ItemCount;

allocCount += item->arrayData.AllocCount;

reqSize += item->arrayData.ReqSize;

allocSize += item->arrayData.AllocSize;

freeCount += item->arrayData.FreeCount;

freeSize += item->arrayData.FreeSize;

}

}

Output::Print(L"-------- ---------- ---------- --------------- --------------- ---------- --------------- ---------- ---------------\n");

Output::Print(L" %8d %10d %15I64d %15I64d %10d %15I64d %10d %15I64d \*\*Total\*\*\n",

itemCount, allocCount, reqSize, allocSize, freeCount, freeSize, allocCount - freeCount, allocSize - freeSize);

#ifdef EXCEL\_FRIENDLY\_DUMP

Output::Print(L"\nExcel friendly version\nItemSize\tItemCount\tAllocCount\tRequestSize\tAllocSize\tFreeCount\tFreeSize\tDiffCount\tDiffSize\tType\n");

for (int i = 0; i < trackerDictionary->Count(); i++)

{

TrackerItem \* item = trackerDictionary->GetValueAt(i);

type\_info const \* typeinfo = trackerDictionary->GetKeyAt(i);

if (item->instanceData.AllocCount != 0)

{

Output::Print(L"%d\t%d\t%d\t%I64d\t%I64d\t%d\t%I64d\t%d\t%I64d\t%S\n",

item->instanceData.ItemSize, item->instanceData.ItemCount, item->instanceData.AllocCount, item->instanceData.ReqSize,

item->instanceData.AllocSize, item->instanceData.FreeCount, item->instanceData.FreeSize,

item->instanceData.AllocCount - item->instanceData.FreeCount, item->instanceData.AllocSize - item->instanceData.FreeSize, typeinfo->name());

}

if (item->arrayData.AllocCount != 0)

{

Output::Print(L"%d\t%d\t%d\t%I64d\t%I64d\t%d\t%I64d\t%d\t%I64d\t%S[]\n",

item->arrayData.ItemSize, item->arrayData.ItemCount, item->arrayData.AllocCount, item->arrayData.ReqSize,

item->arrayData.AllocSize, item->arrayData.FreeCount, item->arrayData.FreeSize,

item->instanceData.AllocCount - item->instanceData.FreeCount, item->arrayData.AllocSize - item->arrayData.FreeSize, typeinfo->name());

}

}

#endif // EXCEL\_FRIENDLY\_DUMP

Output::Flush();

}

#endif // PROFILE\_RECYCLER\_ALLOC

#endif // TRACK\_ALLOC

#ifdef RECYCLER\_VERIFY\_MARK

void

Recycler::VerifyMark()

{

VerifyMarkRoots();

// Can't really verify stack since the recycler code between ScanStack to now may have introduce false references.

// VerifyMarkStack();

autoHeap.VerifyMark();

}

void

Recycler::VerifyMarkRoots()

{

{

this->VerifyMark(transientPinnedObject);

pinnedObjectMap.Map([this](void \* obj, PinRecord const &refCount)

{

if (refCount == 0)

{

Assert(this->hasPendingUnpinnedObject);

}

else

{

// Use the pinrecord as the source reference

this->VerifyMark(obj);

}

});

}

DList<GuestArenaAllocator, HeapAllocator>::Iterator guestArenaIter(&guestArenaList);

while (guestArenaIter.Next())

{

if (guestArenaIter.Data().pendingDelete)

{

Assert(this->hasPendingDeleteGuestArena);

}

else

{

VerifyMarkArena(&guestArenaIter.Data());

}

}

DList<ArenaData \*, HeapAllocator>::Iterator externalGuestArenaIter(&externalGuestArenaList);

while (externalGuestArenaIter.Next())

{

VerifyMarkArena(externalGuestArenaIter.Data());

}

// We can't check external roots here

}

void

Recycler::VerifyMarkArena(ArenaData \* alloc)

{

VerifyMarkBigBlockList(alloc->GetBigBlocks(false));

VerifyMarkBigBlockList(alloc->GetFullBlocks());

VerifyMarkArenaMemoryBlockList(alloc->GetMemoryBlocks());

}

void

Recycler::VerifyMarkBigBlockList(BigBlock \* memoryBlocks)

{

size\_t scanRootBytes = 0;

BigBlock \*blockp = memoryBlocks;

while (blockp != NULL)

{

void\*\* base=(void\*\*)blockp->GetBytes();

size\_t slotCount = blockp->currentByte / sizeof(void\*);

scanRootBytes += blockp->currentByte;

for (size\_t i=0; i < slotCount; i++)

{

VerifyMark(base[i]);

}

blockp = blockp->nextBigBlock;

}

}

void

Recycler::VerifyMarkArenaMemoryBlockList(ArenaMemoryBlock \* memoryBlocks)

{

size\_t scanRootBytes = 0;

ArenaMemoryBlock \*blockp = memoryBlocks;

while (blockp != NULL)

{

void\*\* base=(void\*\*)blockp->GetBytes();

size\_t slotCount = blockp->nbytes / sizeof(void\*);

scanRootBytes += blockp->nbytes;

for (size\_t i=0; i< slotCount; i++)

{

VerifyMark(base[i]);

}

blockp = blockp->next;

}

}

void

Recycler::VerifyMarkStack()

{

SAVE\_THREAD\_CONTEXT();

void \*\* stackTop = (void\*\*) this->savedThreadContext.GetStackTop();

void \* stackStart = GetStackBase();

Assert(stackStart > stackTop);

for (;stackTop < stackStart; stackTop++)

{

void\* candidate = \*stackTop;

VerifyMark(candidate);

}

void\*\* registers = this->savedThreadContext.GetRegisters();

for (int i = 0; i < SavedRegisterState::NumRegistersToSave; i++)

{

VerifyMark(registers[i]);

}

}

void

Recycler::VerifyMark(void \* candidate)

{

void \* realAddress;

HeapBlock \* heapBlock;

if (this->enableScanInteriorPointers)

{

heapBlock = heapBlockMap.GetHeapBlock(candidate);

if (heapBlock == nullptr)

{

return;

}

realAddress = heapBlock->GetRealAddressFromInterior(candidate);

if (realAddress == nullptr)

{

return;

}

}

else

{

heapBlock = this->FindHeapBlock(candidate);

if (heapBlock == nullptr)

{

return;

}

realAddress = candidate;

}

heapBlock->VerifyMark(realAddress);

}

#endif

ArenaAllocator \*

Recycler::CreateGuestArena(wchar\_t const \* name, void (\*outOfMemoryFunc)())

{

// Note, guest arenas use the large block allocator.

return guestArenaList.PrependNode(&HeapAllocator::Instance, name, &recyclerLargeBlockPageAllocator, outOfMemoryFunc);

}

void

Recycler::DeleteGuestArena(ArenaAllocator \* arenaAllocator)

{

GuestArenaAllocator \* guestArenaAllocator = static\_cast<GuestArenaAllocator \*>(arenaAllocator);

if (this->hasPendingConcurrentFindRoot)

{

// We are doing concurrent find root, don't modify the list and mark the arena to be delete

// later when we do find root in thread.

Assert(guestArenaList.HasElement(guestArenaAllocator));

this->hasPendingDeleteGuestArena = true;

guestArenaAllocator->pendingDelete = true;

}

else

{

guestArenaList.RemoveElement(&HeapAllocator::Instance, guestArenaAllocator);

}

}

#ifdef LEAK\_REPORT

void

Recycler::ReportLeaks()

{

if (GetRecyclerFlagsTable().IsEnabled(Js::LeakReportFlag))

{

if (GetRecyclerFlagsTable().ForceMemoryLeak)

{

AUTO\_HANDLED\_EXCEPTION\_TYPE(ExceptionType\_DisableCheck);

struct FakeMemory { int f; };

FakeMemory \* f = RecyclerNewStruct(this, FakeMemory);

this->RootAddRef(f);

}

LeakReport::StartSection(L"Object Graph");

LeakReport::StartRedirectOutput();

RecyclerObjectGraphDumper::Param param = { 0 };

param.skipStack = true;

if (!this->DumpObjectGraph(&param))

{

LeakReport::Print(L"--------------------------------------------------------------------------------\n");

LeakReport::Print(L"ERROR: Out of memory generating leak report\n");

param.stats.markData.markCount = 0;

}

LeakReport::EndRedirectOutput();

if (param.stats.markData.markCount != 0)

{

LeakReport::Print(L"--------------------------------------------------------------------------------\n");

LeakReport::Print(L"Recycler Leaked Object: %d bytes (%d objects)\n",

param.stats.markData.markBytes, param.stats.markData.markCount);

if (GetRecyclerFlagsTable().LeakStackTrace)

{

LeakReport::StartSection(L"Pinned object stack traces");

LeakReport::StartRedirectOutput();

this->PrintPinnedObjectStackTraces();

LeakReport::EndRedirectOutput();

LeakReport::EndSection();

}

}

LeakReport::EndSection();

}

}

void

Recycler::ReportLeaksOnProcessDetach()

{

if (GetRecyclerFlagsTable().IsEnabled(Js::LeakReportFlag))

{

AUTO\_LEAK\_REPORT\_SECTION(this->GetRecyclerFlagsTable(), L"Recycler (%p): Process Termination", this);

LeakReport::StartRedirectOutput();

ReportOnProcessDetach([=]() { this->ReportLeaks(); });

LeakReport::EndRedirectOutput();

}

}

#endif

#ifdef CHECK\_MEMORY\_LEAK

void

Recycler::CheckLeaks(wchar\_t const \* header)

{

if (GetRecyclerFlagsTable().CheckMemoryLeak && this->isPrimaryMarkContextInitialized)

{

if (GetRecyclerFlagsTable().ForceMemoryLeak)

{

AUTO\_HANDLED\_EXCEPTION\_TYPE(ExceptionType\_DisableCheck);

struct FakeMemory { int f; };

FakeMemory \* f = RecyclerNewStruct(this, FakeMemory);

this->RootAddRef(f);

}

Output::CaptureStart();

Output::Print(L"-------------------------------------------------------------------------------------\n");

Output::Print(L"Recycler (%p): %s Leaked Roots\n", this, header);

Output::Print(L"-------------------------------------------------------------------------------------\n");

RecyclerObjectGraphDumper::Param param = { 0 };

param.dumpRootOnly = true;

param.skipStack = true;

if (!this->DumpObjectGraph(&param))

{

free(Output::CaptureEnd());

Output::Print(L"ERROR: Out of memory generating leak report\n");

return;

}

if (param.stats.markData.markCount != 0)

{

if (GetRecyclerFlagsTable().LeakStackTrace)

{

Output::Print(L"-------------------------------------------------------------------------------------\n");

Output::Print(L"Pinned object stack traces");

Output::Print(L"-------------------------------------------------------------------------------------\n");

this->PrintPinnedObjectStackTraces();

}

Output::Print(L"-------------------------------------------------------------------------------------\n");

Output::Print(L"Recycler Leaked Object: %d bytes (%d objects)\n",

param.stats.markData.markBytes, param.stats.markData.markCount);

wchar\_t \* buffer = Output::CaptureEnd();

MemoryLeakCheck::AddLeakDump(buffer, param.stats.markData.markBytes, param.stats.markData.markCount);

#ifdef GENERATE\_DUMP

if (GetRecyclerFlagsTable().IsEnabled(Js::DumpOnLeakFlag))

{

Js::Throw::GenerateDump(GetRecyclerFlagsTable().DumpOnLeak);

}

#endif

}

else

{

free(Output::CaptureEnd());

}

}

}

void

Recycler::CheckLeaksOnProcessDetach(wchar\_t const \* header)

{

if (GetRecyclerFlagsTable().CheckMemoryLeak)

{

ReportOnProcessDetach([=]() { this->CheckLeaks(header); });

}

}

#endif

#if defined(LEAK\_REPORT) || defined(CHECK\_MEMORY\_LEAK)

template <class Fn>

void

Recycler::ReportOnProcessDetach(Fn fn)

{

#if DBG

// Process detach can be done on any thread, just disable the thread check

this->markContext.GetPageAllocator()->SetDisableThreadAccessCheck();

#endif

if (this->IsConcurrentState())

{

this->AbortConcurrent(true);

}

if (this->CollectionInProgress())

{

Output::Print(L"WARNING: Thread terminated during GC. Can't dump object graph\n");

return;

}

// Don't mark external roots on another thread

this->SetExternalRootMarker(NULL, NULL);

#if DBG

this->ResetThreadId();

#endif

fn();

}

void

Recycler::PrintPinnedObjectStackTraces()

{

pinnedObjectMap.Map([this](void \* object, PinRecord const& pinRecord)

{

this->DumpObjectDescription(object);

Output::Print(L"\n");

StackBackTraceNode::PrintAll(pinRecord.stackBackTraces);

}

);

}

#endif

#if defined(RECYCLER\_DUMP\_OBJECT\_GRAPH) || defined(LEAK\_REPORT) || defined(CHECK\_MEMORY\_LEAK)

void

Recycler::SetInDllCanUnloadNow()

{

inDllCanUnloadNow = true;

// Just clear out the root marker for the dump graph and report leaks

SetExternalRootMarker(NULL, NULL);

}

void

Recycler::SetInDetachProcess()

{

inDetachProcess = true;

// Just clear out the root marker for the dump graph and report leaks

SetExternalRootMarker(NULL, NULL);

}

#endif

#ifdef ENABLE\_JS\_ETW

ULONG Recycler::EventWriteFreeMemoryBlock(HeapBlock\* heapBlock)

{

if (EventEnabledJSCRIPT\_RECYCLER\_FREE\_MEMORY\_BLOCK())

{

char\* memoryAddress = NULL;

ULONG objectSize = 0;

ULONG blockSize = 0;

switch (heapBlock->GetHeapBlockType())

{

case HeapBlock::HeapBlockType::SmallFinalizableBlockType:

case HeapBlock::HeapBlockType::SmallNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::SmallFinalizableBlockWithBarrierType:

case HeapBlock::HeapBlockType::SmallNormalBlockWithBarrierType:

#endif

case HeapBlock::HeapBlockType::SmallLeafBlockType:

{

SmallHeapBlock\* smallHeapBlock = static\_cast<SmallHeapBlock\*>(heapBlock);

memoryAddress = smallHeapBlock->GetAddress();

blockSize = (ULONG)(smallHeapBlock->GetEndAddress() - memoryAddress);

objectSize = smallHeapBlock->GetObjectSize();

}

break;

case HeapBlock::HeapBlockType::MediumFinalizableBlockType:

case HeapBlock::HeapBlockType::MediumNormalBlockType:

#ifdef RECYCLER\_WRITE\_BARRIER

case HeapBlock::HeapBlockType::MediumFinalizableBlockWithBarrierType:

case HeapBlock::HeapBlockType::MediumNormalBlockWithBarrierType:

#endif

case HeapBlock::HeapBlockType::MediumLeafBlockType:

{

MediumHeapBlock\* mediumHeapBlock = static\_cast<MediumHeapBlock\*>(heapBlock);

memoryAddress = mediumHeapBlock->GetAddress();

blockSize = (ULONG)(mediumHeapBlock->GetEndAddress() - memoryAddress);

objectSize = mediumHeapBlock->GetObjectSize();

}

case HeapBlock::HeapBlockType::LargeBlockType:

{

LargeHeapBlock\* largeHeapBlock = static\_cast<LargeHeapBlock\*>(heapBlock);

memoryAddress = largeHeapBlock->GetBeginAddress();

blockSize = (ULONG)(largeHeapBlock->GetEndAddress() - memoryAddress);

objectSize = blockSize;

}

break;

default:

AssertMsg(FALSE, "invalid heapblock type");

}

EventWriteJSCRIPT\_RECYCLER\_FREE\_MEMORY\_BLOCK(memoryAddress, blockSize, objectSize);

}

return S\_OK;

}

void Recycler::FlushFreeRecord()

{

Assert(bulkFreeMemoryWrittenCount <= Recycler::BulkFreeMemoryCount);

JS\_ETW(EventWriteJSCRIPT\_RECYCLER\_FREE\_MEMORY(bulkFreeMemoryWrittenCount, sizeof(Recycler::ETWFreeRecord), etwFreeRecords));

bulkFreeMemoryWrittenCount = 0;

}

void Recycler::AppendFreeMemoryETWRecord(\_\_in char \*address, size\_t size)

{

Assert(bulkFreeMemoryWrittenCount < Recycler::BulkFreeMemoryCount);

\_\_analysis\_assume(bulkFreeMemoryWrittenCount < Recycler::BulkFreeMemoryCount);

etwFreeRecords[bulkFreeMemoryWrittenCount].memoryAddress = address;

// TODO: change to size\_t or uint64?

etwFreeRecords[bulkFreeMemoryWrittenCount].objectSize = (uint)size;

bulkFreeMemoryWrittenCount++;

if (bulkFreeMemoryWrittenCount == Recycler::BulkFreeMemoryCount)

{

FlushFreeRecord();

Assert(bulkFreeMemoryWrittenCount == 0);

}

}

#endif

#ifdef PROFILE\_EXEC

ArenaAllocator \*

Recycler::AddBackgroundProfilerArena()

{

return this->backgroundProfilerArena.PrependNode(&HeapAllocator::Instance,

L"BgGCProfiler", &this->backgroundProfilerPageAllocator, Js::Throw::OutOfMemory);

}

void

Recycler::ReleaseBackgroundProfilerArena(ArenaAllocator \* arena)

{

this->backgroundProfilerArena.RemoveElement(&HeapAllocator::Instance, arena);

}

void

Recycler::SetProfiler(Js::Profiler \* profiler, Js::Profiler \* backgroundProfiler)

{

this->profiler = profiler;

this->backgroundProfiler = backgroundProfiler;

}

#endif

void Recycler::SetObjectBeforeCollectCallback(void\* object, ObjectBeforeCollectCallback callback, void\* callbackState)

{

if (objectBeforeCollectCallbackState == ObjectBeforeCollectCallback\_Shutdown)

{

return; // NOP at shutdown

}

if (objectBeforeCollectCallbackMap == nullptr)

{

if (callback == nullptr) return;

objectBeforeCollectCallbackMap = HeapNew(ObjectBeforeCollectCallbackMap, &HeapAllocator::Instance);

}

// only allow 1 callback per object

objectBeforeCollectCallbackMap->Item(object, ObjectBeforeCollectCallbackData(callback, callbackState));

if (callback != nullptr && this->IsInObjectBeforeCollectCallback()) // revive

{

this->ScanMemory(&object, sizeof(object));

this->ProcessMark(/\*background\*/false);

}

}

bool Recycler::ProcessObjectBeforeCollectCallbacks(bool atShutdown/\*= false\*/)

{

if (this->objectBeforeCollectCallbackMap == nullptr)

{

return false; // no callbacks

}

Assert(atShutdown || this->IsMarkState());

Assert(!this->IsInObjectBeforeCollectCallback());

AutoRestoreValue<ObjectBeforeCollectCallbackState> autoInObjectBeforeCollectCallback(&objectBeforeCollectCallbackState,

atShutdown ? ObjectBeforeCollectCallback\_Shutdown: ObjectBeforeCollectCallback\_Normal);

// The callbacks may register/unregister callbacks while we are enumerating the current map. To avoid

// conflicting usage of the callback map, we swap it out. New registration will go to a new map.

AutoAllocatorObjectPtr<ObjectBeforeCollectCallbackMap, HeapAllocator> oldCallbackMap(

this->objectBeforeCollectCallbackMap, &HeapAllocator::Instance);

this->objectBeforeCollectCallbackMap = nullptr;

bool hasRemainingCallbacks = false;

oldCallbackMap->MapAndRemoveIf([&](const ObjectBeforeCollectCallbackMap::EntryType& entry)

{

const ObjectBeforeCollectCallbackData& data = entry.Value();

if (data.callback != nullptr)

{

void\* object = entry.Key();

if (atShutdown || !this->IsObjectMarked(object))

{

data.callback(object, data.callbackState);

}

else

{

hasRemainingCallbacks = true;

return false; // Do not remove this entry, remaining callback for future

}

}

return true; // Remove this entry

});

// Merge back remaining callbacks if any

if (hasRemainingCallbacks)

{

if (this->objectBeforeCollectCallbackMap == nullptr)

{

this->objectBeforeCollectCallbackMap = oldCallbackMap.Detach();

}

else

{

if (oldCallbackMap->Count() > this->objectBeforeCollectCallbackMap->Count())

{

// Swap so that oldCallbackMap is the smaller one

ObjectBeforeCollectCallbackMap\* tmp = oldCallbackMap.Detach();

\*&oldCallbackMap = this->objectBeforeCollectCallbackMap;

this->objectBeforeCollectCallbackMap = tmp;

}

oldCallbackMap->Map([&](void\* object, const ObjectBeforeCollectCallbackData& data)

{

this->objectBeforeCollectCallbackMap->Item(object, data);

});

}

}

return true; // maybe called callbacks

}

void Recycler::ClearObjectBeforeCollectCallbacks()

{

// This is called at shutting down. All objects will be gone. Invoke each registered callback if any.

ProcessObjectBeforeCollectCallbacks(/\*atShutdown\*/true);

Assert(objectBeforeCollectCallbackMap == nullptr);

}

#ifdef RECYCLER\_TEST\_SUPPORT

void Recycler::SetCheckFn(BOOL(\*checkFn)(char\* addr, size\_t size))

{

Assert(BinaryFeatureControl::RecyclerTest());

this->EnsureNotCollecting();

this->checkFn = checkFn;

}

#endif

void

Recycler::NotifyFree(\_\_in char \*address, size\_t size)

{

RecyclerVerboseTrace(GetRecyclerFlagsTable(), L"Sweeping object %p\n", address);

#ifdef RECYCLER\_TEST\_SUPPORT

if (BinaryFeatureControl::RecyclerTest())

{

if (checkFn != NULL)

checkFn(address, size);

}

#endif

#ifdef ENABLE\_JS\_ETW

if (EventEnabledJSCRIPT\_RECYCLER\_FREE\_MEMORY())

{

AppendFreeMemoryETWRecord(address, (UINT)size);

}

#endif

RecyclerMemoryTracking::ReportFree(this, address, size);

RECYCLER\_PERF\_COUNTER\_DEC(LiveObject);

RECYCLER\_PERF\_COUNTER\_SUB(LiveObjectSize, size);

RECYCLER\_PERF\_COUNTER\_ADD(FreeObjectSize, size);

if (HeapInfo::IsSmallBlockAllocation(HeapInfo::GetAlignedSizeNoCheck(size)))

{

RECYCLER\_PERF\_COUNTER\_DEC(SmallHeapBlockLiveObject);

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockLiveObjectSize, size);

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockFreeObjectSize, size);

}

else

{

RECYCLER\_PERF\_COUNTER\_DEC(LargeHeapBlockLiveObject);

RECYCLER\_PERF\_COUNTER\_SUB(LargeHeapBlockLiveObjectSize, size);

RECYCLER\_PERF\_COUNTER\_ADD(LargeHeapBlockFreeObjectSize, size);

}

#ifdef RECYCLER\_MEMORY\_VERIFY

if (this->VerifyEnabled())

{

VerifyCheckPad(address, size);

}

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

TrackFree(address, size);

#endif

#ifdef RECYCLER\_STATS

collectionStats.objectSweptCount++;

collectionStats.objectSweptBytes += size;

if (!isForceSweeping)

{

collectionStats.objectSweptFreeListCount++;

collectionStats.objectSweptFreeListBytes += size;

}

#endif

}

size\_t

RecyclerHeapObjectInfo::GetSize() const

{

Assert(m\_heapBlock);

size\_t size;

#if LARGEHEAPBLOCK\_ENCODING

if (isUsingLargeHeapBlock)

{

size = m\_largeHeapBlockHeader->objectSize;

}

#else

if (m\_heapBlock->IsLargeHeapBlock())

{

size = ((LargeHeapBlock\*)m\_heapBlock)->GetObjectSize(m\_address);

}

#endif

else

{

// All small heap block types have the same layout for the object size field.

size = ((SmallHeapBlock\*)m\_heapBlock)->GetObjectSize();

}

#ifdef RECYCLER\_MEMORY\_VERIFY

if (m\_recycler->VerifyEnabled())

{

size -= \*(size\_t \*)(((char \*)m\_address) + size - sizeof(size\_t));

}

#endif

return size;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#include "CollectionState.h"

namespace Js

{

class Profiler;

enum Phase;

};

namespace JsUtil

{

class ThreadService;

};

class StackBackTraceNode;

class ScriptEngineBase;

class JavascriptThreadService;

#ifdef PROFILE\_MEM

struct RecyclerMemoryData;

#endif

namespace Memory

{

template <typename T> class RecyclerRootPtr;

class AutoBooleanToggle

{

public:

AutoBooleanToggle(bool \* b, bool value = true, bool valueMayChange = false)

: b(b)

{

Assert(!(\*b));

\*b = value;

#if DBG

this->value = value;

this->valueMayChange = valueMayChange;

#endif

}

~AutoBooleanToggle()

{

if (b)

{

Assert(valueMayChange || \*b == value);

\*b = false;

}

}

void Leave()

{

Assert(valueMayChange || \*b == value);

\*b = false;

b = nullptr;

}

private:

bool \* b;

#if DBG

bool value;

bool valueMayChange;

#endif

};

template <class T>

class AutoRestoreValue

{

public:

AutoRestoreValue(T\* var, const T& val):

variable(var)

{

Assert(var);

oldValue = (\*variable);

(\*variable) = val;

#ifdef DEBUG

debugSetValue = val;

#endif

}

~AutoRestoreValue()

{

Assert((\*variable) == debugSetValue);

(\*variable) = oldValue;

}

private:

#ifdef DEBUG

T debugSetValue;

#endif

T\* variable;

T oldValue;

};

class Recycler;

class RecyclerScanMemoryCallback

{

public:

RecyclerScanMemoryCallback(Recycler\* recycler) : recycler(recycler) {}

void operator()(void\*\* obj, size\_t byteCount);

private:

Recycler\* recycler;

};

template<ObjectInfoBits infoBits>

struct InfoBitsWrapper{};

// Allocation macro

#define RecyclerNew(recycler,T,...) AllocatorNewBase(Recycler, recycler, AllocInlined, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewPlus(recycler,size,T,...) AllocatorNewPlus(Recycler, recycler, size, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewPlusLeaf(recycler,size,T,...) AllocatorNewPlusLeaf(Recycler, recycler, size, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewPlusZ(recycler,size,T,...) AllocatorNewPlusZ(Recycler, recycler, size, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewPlusLeafZ(recycler,size,T,...) AllocatorNewPlusLeafZ(Recycler, recycler, size, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewZ(recycler,T,...) AllocatorNewBase(Recycler, recycler, AllocZeroInlined, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewStruct(recycler,T) AllocatorNewStructBase(Recycler, recycler, AllocInlined, T)

#define RecyclerNewStructZ(recycler,T) AllocatorNewStructBase(Recycler, recycler, AllocZeroInlined, T)

#define RecyclerNewStructPlus(recycler,size,T) AllocatorNewStructPlus(Recycler, recycler, size, T)

#define RecyclerNewStructLeaf(recycler,T) AllocatorNewStructBase(Recycler, recycler, AllocLeafInlined, T)

#define RecyclerNewStructLeafZ(recycler,T) AllocatorNewStructBase(Recycler, recycler, AllocLeafZeroInlined, T)

#define RecyclerNewLeaf(recycler,T,...) AllocatorNewBase(Recycler, recycler, AllocLeafInlined, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewLeafZ(recycler,T,...) AllocatorNewBase(Recycler, recycler, AllocLeafZeroInlined, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewArrayLeafZ(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, AllocLeafZero, T, count)

#define RecyclerNewArray(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, Alloc, T, count)

#define RecyclerNewArrayZ(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, AllocZero, T, count)

#define RecyclerNewArrayLeaf(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, AllocLeaf, T, count)

// Use static\_cast to make sure the finalized and tracked object have the right base class

#define RecyclerNewFinalized(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocFinalizedInlined, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewFinalizedLeaf(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocFinalizedLeafInlined, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewFinalizedPlus(recycler, size, T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewPlusBase(Recycler, recycler, AllocFinalized, size, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewFinalizedLeafPlus(recycler, size, T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewPlusBase(Recycler, recycler, AllocFinalizedLeaf, size, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewTracked(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocTrackedInlined, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewTrackedLeaf(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocTrackedLeafInlined, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewTrackedLeafPlusZ(recycler,size,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewPlusBase(Recycler, recycler, AllocZeroTrackedLeafInlined, size, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewEnumClass(recycler, enumClass, T, ...) new (TRACK\_ALLOC\_INFO(static\_cast<Recycler \*>(recycler), T, Recycler, 0, (size\_t)-1), enumClass) T(\_\_VA\_ARGS\_\_)

#define RecyclerNewWithInfoBits(recycler, infoBits, T, ...) new (TRACK\_ALLOC\_INFO(static\_cast<Recycler \*>(recycler), T, Recycler, 0, (size\_t)-1), InfoBitsWrapper<infoBits>()) T(\_\_VA\_ARGS\_\_)

#define RecyclerNewFinalizedClientTracked(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocFinalizedClientTrackedInlined, T, \_\_VA\_ARGS\_\_)))

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC

#define RecyclerNewWithBarrier(recycler,T,...) AllocatorNewBase(Recycler, recycler, AllocWithBarrier, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewWithBarrierPlus(recycler,size,T,...) AllocatorNewPlusBase(Recycler, recycler, AllocWithBarrier, size, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewWithBarrierPlusZ(recycler,size,T,...) AllocatorNewPlusBase(Recycler, recycler, AllocZeroWithBarrier, size, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewWithBarrierArray(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, AllocWithBarrier, T, count)

#define RecyclerNewWithBarrierArrayZ(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, AllocZeroWithBarrier, T, count)

#define RecyclerNewWithBarrierStruct(recycler,T) AllocatorNewStructBase(Recycler, recycler, AllocWithBarrier, T)

#define RecyclerNewWithBarrierStructZ(recycler,T) AllocatorNewStructBase(Recycler, recycler, AllocZeroWithBarrier, T)

#define RecyclerNewWithBarrierFinalized(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocFinalizedWithBarrierInlined, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewWithBarrierFinalizedPlus(recycler, size, T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewPlusBase(Recycler, recycler, AllocFinalizedWithBarrier, size, T, \_\_VA\_ARGS\_\_)))

#else

#define RecyclerNewWithBarrier RecyclerNew

#define RecyclerNewWithBarrierPlus RecyclerNewPlus

#define RecyclerNewWithBarrierPlusZ RecyclerNewPlusZ

#define RecyclerNewWithBarrierArray RecyclerNewArray

#define RecyclerNewWithBarrierArrayZ RecyclerNewArrayZ

#define RecyclerNewWithBarrierStruct RecyclerNewStruct

#define RecyclerNewWithBarrierStructZ RecyclerNewStructZ

#define RecyclerNewWithBarrierFinalized RecyclerNewFinalized

#define RecyclerNewWithBarrierFinalizedPlus RecyclerNewFinalizedPlus

#endif

#ifdef TRACE\_OBJECT\_LIFETIME

#define RecyclerNewLeafTrace(recycler,T,...) AllocatorNewBase(Recycler, recycler, AllocLeafTrace, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewLeafZTrace(recycler,T,...) AllocatorNewBase(Recycler, recycler, AllocLeafZeroTrace, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewPlusLeafTrace(recycler,size,T,...) AllocatorNewPlusBase(Recycler, recycler, AllocLeafTrace, size, T, \_\_VA\_ARGS\_\_)

#define RecyclerNewArrayLeafZTrace(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, AllocLeafZeroTrace, T, count)

#define RecyclerNewArrayTrace(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, AllocTrace, T, count)

#define RecyclerNewArrayZTrace(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, AllocZeroTrace, T, count)

#define RecyclerNewArrayLeafTrace(recycler,T,count) AllocatorNewArrayBase(Recycler, recycler, AllocLeafTrace, T, count)

// Use static\_cast to make sure the finalized and tracked object have the right base class

#define RecyclerNewFinalizedTrace(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocFinalizedTrace, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewFinalizedLeafTrace(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocFinalizedLeafTrace, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewFinalizedPlusTrace(recycler, size, T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewPlusBase(Recycler, recycler, AllocFinalizedTrace, size, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewTrackedTrace(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocTrackedTrace, T, \_\_VA\_ARGS\_\_)))

#define RecyclerNewTrackedLeafTrace(recycler,T,...) static\_cast<T \*>(static\_cast<FinalizableObject \*>(AllocatorNewBase(Recycler, recycler, AllocTrackedLeafTrace, T, \_\_VA\_ARGS\_\_)))

#else

#define RecyclerNewLeafTrace RecyclerNewLeaf

#define RecyclerNewLeafZTrace RecyclerNewLeafZ

#define RecyclerNewPlusLeafTrace RecyclerNewPlusLeaf

#define RecyclerNewArrayLeafZTrace RecyclerNewArrayLeafZ

#define RecyclerNewArrayTrace RecyclerNewArray

#define RecyclerNewArrayZTrace RecyclerNewArrayZ

#define RecyclerNewArrayLeafTrace RecyclerNewArrayLeaf

#define RecyclerNewFinalizedTrace RecyclerNewFinalized

#define RecyclerNewFinalizedLeafTrace RecyclerNewFinalizedLeaf

#define RecyclerNewFinalizedPlusTrace RecyclerNewFinalizedPlus

#define RecyclerNewTrackedTrace RecyclerNewTracked

#define RecyclerNewTrackedLeafTrace RecyclerNewTrackedLeaf

#endif

#ifdef RECYCLER\_TRACE

#define RecyclerVerboseTrace(flags, ...) \

if (flags.Verbose && flags.Trace.IsEnabled(Js::RecyclerPhase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#define AllocationVerboseTrace(flags, ...) \

if (flags.Verbose && flags.Trace.IsEnabled(Js::MemoryAllocationPhase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#define LargeAllocationVerboseTrace(flags, ...) \

if (flags.Verbose && \

(flags.Trace.IsEnabled(Js::MemoryAllocationPhase) || \

flags.Trace.IsEnabled(Js::LargeMemoryAllocationPhase))) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#define PageAllocatorAllocationVerboseTrace(flags, ...) \

if (flags.Verbose && flags.Trace.IsEnabled(Js::PageAllocatorAllocPhase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#else

#define RecyclerVerboseTrace(...)

#define AllocationVerboseTrace(...)

#define LargeAllocationVerboseTrace(...)

#endif

#define RecyclerHeapNew(recycler,heapInfo,T,...) new (recycler, heapInfo) T(\_\_VA\_ARGS\_\_)

#define RecyclerHeapDelete(recycler,heapInfo,addr) (static\_cast<Recycler \*>(recycler)->HeapFree(heapInfo,addr))

typedef void (\_\_cdecl\* ExternalRootMarker)(void \*);

enum CollectionFlags

{

CollectHeuristic\_AllocSize = 0x00000001,

CollectHeuristic\_Time = 0x00000002,

CollectHeuristic\_TimeIfScriptActive = 0x00000004,

CollectHeuristic\_TimeIfInScript = 0x00000008,

CollectHeuristic\_Never = 0x00000080,

CollectHeuristic\_Mask = 0x000000FF,

CollectOverride\_FinishConcurrent = 0x00001000,

CollectOverride\_ExhaustiveCandidate = 0x00002000,

CollectOverride\_ForceInThread = 0x00004000,

CollectOverride\_AllowDispose = 0x00008000,

CollectOverride\_AllowReentrant = 0x00010000,

CollectOverride\_ForceFinish = 0x00020000,

CollectOverride\_Explicit = 0x00040000,

CollectOverride\_DisableIdleFinish = 0x00080000,

CollectOverride\_BackgroundFinishMark= 0x00100000,

CollectOverride\_FinishConcurrentTimeout = 0x00200000,

CollectOverride\_NoExhaustiveCollect = 0x00400000,

CollectOverride\_SkipStack = 0x01000000,

CollectMode\_Partial = 0x08000000,

CollectMode\_Concurrent = 0x10000000,

CollectMode\_Exhaustive = 0x20000000,

CollectMode\_DecommitNow = 0x40000000,

CollectMode\_CacheCleanup = 0x80000000,

CollectNowForceInThread = CollectOverride\_ForceInThread,

CollectNowForceInThreadExternal = CollectOverride\_ForceInThread | CollectOverride\_AllowDispose,

CollectNowForceInThreadExternalNoStack = CollectOverride\_ForceInThread | CollectOverride\_AllowDispose | CollectOverride\_SkipStack,

CollectNowDefault = CollectOverride\_FinishConcurrent,

CollectNowDefaultLSCleanup = CollectOverride\_FinishConcurrent | CollectOverride\_AllowDispose,

CollectNowDecommitNowExplicit = CollectNowDefault | CollectMode\_DecommitNow | CollectMode\_CacheCleanup | CollectOverride\_Explicit | CollectOverride\_AllowDispose,

CollectNowConcurrent = CollectOverride\_FinishConcurrent | CollectMode\_Concurrent,

CollectNowExhaustive = CollectOverride\_FinishConcurrent | CollectMode\_Exhaustive | CollectOverride\_AllowDispose,

CollectNowPartial = CollectOverride\_FinishConcurrent | CollectMode\_Partial,

CollectNowConcurrentPartial = CollectMode\_Concurrent | CollectNowPartial,

CollectOnAllocation = CollectHeuristic\_AllocSize | CollectHeuristic\_Time | CollectMode\_Concurrent | CollectMode\_Partial | CollectOverride\_FinishConcurrent | CollectOverride\_AllowReentrant | CollectOverride\_FinishConcurrentTimeout,

CollectOnTypedArrayAllocation = CollectHeuristic\_AllocSize | CollectHeuristic\_Time | CollectMode\_Concurrent | CollectMode\_Partial | CollectOverride\_FinishConcurrent | CollectOverride\_AllowReentrant | CollectOverride\_FinishConcurrentTimeout | CollectOverride\_AllowDispose,

CollectOnScriptIdle = CollectOverride\_FinishConcurrent | CollectMode\_Concurrent | CollectMode\_CacheCleanup | CollectOverride\_SkipStack,

CollectOnScriptExit = CollectHeuristic\_AllocSize | CollectOverride\_FinishConcurrent | CollectMode\_Concurrent | CollectMode\_CacheCleanup,

CollectExhaustiveCandidate = CollectHeuristic\_Never | CollectOverride\_ExhaustiveCandidate,

CollectOnScriptCloseNonPrimary = CollectNowConcurrent | CollectOverride\_ExhaustiveCandidate | CollectOverride\_AllowDispose,

CollectOnRecoverFromOutOfMemory = CollectOverride\_ForceInThread | CollectMode\_DecommitNow,

CollectOnSuspendCleanup = CollectNowConcurrent | CollectMode\_Exhaustive | CollectMode\_DecommitNow | CollectOverride\_DisableIdleFinish,

FinishConcurrentOnIdle = CollectMode\_Concurrent | CollectOverride\_DisableIdleFinish,

FinishConcurrentOnIdleAtRoot = CollectMode\_Concurrent | CollectOverride\_DisableIdleFinish | CollectOverride\_SkipStack,

FinishConcurrentOnExitScript = CollectMode\_Concurrent | CollectOverride\_DisableIdleFinish | CollectOverride\_BackgroundFinishMark,

FinishConcurrentOnEnterScript = CollectMode\_Concurrent | CollectOverride\_DisableIdleFinish | CollectOverride\_BackgroundFinishMark,

FinishConcurrentOnAllocation = CollectMode\_Concurrent | CollectOverride\_DisableIdleFinish | CollectOverride\_BackgroundFinishMark,

FinishDispose = CollectOverride\_AllowDispose,

FinishDisposeTimed = CollectOverride\_AllowDispose | CollectHeuristic\_TimeIfScriptActive,

ForceFinishCollection = CollectOverride\_ForceFinish | CollectOverride\_ForceInThread,

#ifdef RECYCLER\_STRESS

CollectStress = CollectNowForceInThread,

#ifdef PARTIAL\_GC\_ENABLED

CollectPartialStress = CollectMode\_Partial,

#endif

#ifdef CONCURRENT\_GC\_ENABLED

CollectBackgroundStress = CollectNowDefault,

CollectConcurrentStress = CollectNowConcurrent,

#ifdef PARTIAL\_GC\_ENABLED

CollectConcurrentPartialStress = CollectConcurrentStress | CollectPartialStress,

#endif

#endif

#endif

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

CollectNowFinalGC = CollectNowExhaustive | CollectOverride\_ForceInThread | CollectOverride\_SkipStack | CollectOverride\_Explicit | CollectOverride\_AllowDispose,

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

CollectNowExhaustiveSkipStack = CollectNowExhaustive | CollectOverride\_SkipStack, // Used by test

#endif

};

class RecyclerCollectionWrapper

{

public:

typedef BOOL (Recycler::\*CollectionFunction)(CollectionFlags flags);

virtual void PreCollectionCallBack(CollectionFlags flags) = 0;

virtual void PreSweepCallback() = 0;

virtual void PreRescanMarkCallback() = 0;

virtual size\_t RootMarkCallback(RecyclerScanMemoryCallback& scanMemoryCallback, BOOL \* stacksScannedByRuntime) = 0;

virtual void RescanMarkTimeoutCallback() = 0;

virtual void EndMarkCallback() = 0;

virtual void ConcurrentCallback() = 0;

virtual void WaitCollectionCallBack() = 0;

virtual void PostCollectionCallBack() = 0;

virtual BOOL ExecuteRecyclerCollectionFunction(Recycler \* recycler, CollectionFunction function, CollectionFlags flags) = 0;

virtual uint GetRandomNumber() = 0;

#ifdef FAULT\_INJECTION

virtual void DisposeScriptContextByFaultInjectionCallBack() = 0;

#endif

virtual void DisposeObjects(Recycler \* recycler) = 0;

#ifdef ENABLE\_PROJECTION

virtual void MarkExternalWeakReferencedObjects(bool inPartialCollect) = 0;

virtual void ResolveExternalWeakReferencedObjects() = 0;

#endif

#if DBG || defined(PROFILE\_EXEC)

virtual bool AsyncHostOperationStart(void \*) = 0;

virtual void AsyncHostOperationEnd(bool wasInAsync, void \*) = 0;

#endif

};

class DefaultRecyclerCollectionWrapper : public RecyclerCollectionWrapper

{

public:

virtual void PreCollectionCallBack(CollectionFlags flags) override {}

virtual void PreSweepCallback() override {}

virtual void PreRescanMarkCallback() override {}

virtual void RescanMarkTimeoutCallback() override {}

virtual void EndMarkCallback() override {}

virtual size\_t RootMarkCallback(RecyclerScanMemoryCallback& scanMemoryCallback, BOOL \* stacksScannedByRuntime) override { \*stacksScannedByRuntime = FALSE; return 0; }

virtual void ConcurrentCallback() override {}

virtual void WaitCollectionCallBack() override {}

virtual void PostCollectionCallBack() override {}

virtual BOOL ExecuteRecyclerCollectionFunction(Recycler \* recycler, CollectionFunction function, CollectionFlags flags) override;

virtual uint GetRandomNumber() override { return 0; }

#ifdef FAULT\_INJECTION

virtual void DisposeScriptContextByFaultInjectionCallBack() override {};

#endif

virtual void DisposeObjects(Recycler \* recycler) override;

#ifdef ENABLE\_PROJECTION

virtual void MarkExternalWeakReferencedObjects(bool inPartialCollect) override {};

virtual void ResolveExternalWeakReferencedObjects() override {};

#endif

#if DBG || defined(PROFILE\_EXEC)

virtual bool AsyncHostOperationStart(void \*) override { return false; };

virtual void AsyncHostOperationEnd(bool wasInAsync, void \*) override {};

#endif

static DefaultRecyclerCollectionWrapper Instance;

private:

static bool IsCollectionDisabled(Recycler \* recycler);

};

#ifdef RECYCLER\_STATS

struct RecyclerCollectionStats

{

size\_t startCollectAllocBytes;

#ifdef PARTIAL\_GC\_ENABLED

size\_t startCollectNewPageCount;

#endif

size\_t continueCollectAllocBytes;

size\_t finishCollectTryCount;

// Heuristic Stats

#ifdef PARTIAL\_GC\_ENABLED

size\_t rescanRootBytes;

size\_t estimatedPartialReuseBytes;

size\_t uncollectedNewPageCountPartialCollect;

size\_t partialCollectSmallHeapBlockReuseMinFreeBytes;

double collectEfficacy;

double collectCost;

#endif

// Mark stats

size\_t tryMarkCount; // # of pointer try mark (\* pointer size to get total number byte looked at)

size\_t tryMarkNullCount;

size\_t tryMarkUnalignedCount;

size\_t tryMarkNonRecyclerMemoryCount;

size\_t tryMarkInteriorCount;

size\_t tryMarkInteriorNullCount;

size\_t tryMarkInteriorNonRecyclerMemoryCount;

size\_t rootCount;

size\_t stackCount;

size\_t remarkCount;

size\_t scanCount; // non-leaf objects marked.

size\_t trackCount;

size\_t finalizeCount;

size\_t markThruNewObjCount;

size\_t markThruFalseNewObjCount;

struct MarkData

{

// Rescan stats

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t rescanPageCount;

size\_t rescanObjectCount;

size\_t rescanObjectByteCount;

size\_t rescanLargePageCount;

size\_t rescanLargeObjectCount;

size\_t rescanLargeByteCount;

#endif

size\_t markCount; // total number of object marked

size\_t markBytes; // size of all objects marked.

} markData;

#ifdef CONCURRENT\_GC\_ENABLED

MarkData backgroundMarkData[RecyclerHeuristic::MaxBackgroundRepeatMarkCount];

size\_t trackedObjectCount;

#endif

#ifdef PARTIAL\_GC\_ENABLED

size\_t clientTrackedObjectCount;

#endif

// Sweep stats

size\_t heapBlockCount[HeapBlock::BlockTypeCount]; // number of heap blocks (processed during swept)

size\_t heapBlockFreeCount[HeapBlock::BlockTypeCount]; // number of heap blocks deleted

size\_t heapBlockConcurrentSweptCount[HeapBlock::SmallBlockTypeCount];

size\_t heapBlockSweptCount[HeapBlock::SmallBlockTypeCount]; // number of heap blocks swept

size\_t objectSweptCount; // objects freed (free list + whole page freed)

size\_t objectSweptBytes;

size\_t objectSweptFreeListCount; // objects freed (free list)

size\_t objectSweptFreeListBytes;

size\_t objectSweepScanCount; // number of objects walked for sweeping (exclude whole page freed)

size\_t finalizeSweepCount; // number of objects finalizer/dispose called

#ifdef PARTIAL\_GC\_ENABLED

size\_t smallNonLeafHeapBlockPartialReuseCount[HeapBlock::SmallBlockTypeCount];

size\_t smallNonLeafHeapBlockPartialReuseBytes[HeapBlock::SmallBlockTypeCount];

size\_t smallNonLeafHeapBlockPartialUnusedCount[HeapBlock::SmallBlockTypeCount];

size\_t smallNonLeafHeapBlockPartialUnusedBytes[HeapBlock::SmallBlockTypeCount];

#endif

// Memory Stats

size\_t heapBlockFreeByteCount[HeapBlock::BlockTypeCount]; // The remaining usable free byte count

size\_t largeHeapBlockUsedByteCount; // Used byte count

size\_t largeHeapBlockTotalByteCount; // Total byte count

// Empty/zero heap block stats

uint numEmptySmallBlocks[HeapBlock::SmallBlockTypeCount];

uint numZeroedOutSmallBlocks;

};

#define RECYCLER\_STATS\_INC\_IF(cond, r, f) if (cond) { RECYCLER\_STATS\_INC(r, f); }

#define RECYCLER\_STATS\_INC(r, f) ++r->collectionStats.f

#define RECYCLER\_STATS\_INTERLOCKED\_INC(r, f) { InterlockedIncrement((LONG \*)&r->collectionStats.f); }

#define RECYCLER\_STATS\_DEC(r, f) --r->collectionStats.f

#define RECYCLER\_STATS\_ADD(r, f, v) r->collectionStats.f += (v)

#define RECYCLER\_STATS\_INTERLOCKED\_ADD(r, f, v) { InterlockedAdd((LONG \*)&r->collectionStats.f, (LONG)(v)); }

#define RECYCLER\_STATS\_SUB(r, f, v) r->collectionStats.f -= (v)

#define RECYCLER\_STATS\_SET(r, f, v) r->collectionStats.f = v

#else

#define RECYCLER\_STATS\_INC\_IF(cond, r, f)

#define RECYCLER\_STATS\_INC(r, f)

#define RECYCLER\_STATS\_INTERLOCKED\_INC(r, f)

#define RECYCLER\_STATS\_DEC(r, f)

#define RECYCLER\_STATS\_ADD(r, f, v)

#define RECYCLER\_STATS\_INTERLOCKED\_ADD(r, f, v)

#define RECYCLER\_STATS\_SUB(r, f, v)

#define RECYCLER\_STATS\_SET(r, f, v)

#endif

#ifdef RECYCLER\_TRACE

struct CollectionParam

{

CollectionFlags flags;

bool finishOnly;

bool repeat;

bool priorityBoostConcurentSweepOverride;

bool domCollect;

int timeDiff;

size\_t uncollectedAllocBytes;

size\_t uncollectedPinnedObjects;

#ifdef PARTIAL\_GC\_ENABLED

size\_t uncollectedNewPageCountPartialCollect;

size\_t uncollectedNewPageCount;

size\_t unusedPartialCollectFreeBytes;

bool inPartialCollectMode;

#endif

};

#endif

#include "RecyclerObjectGraphDumper.h"

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

// Macro to be used within the recycler

#define ForRecyclerPageAllocator(action) { \

this->recyclerPageAllocator.##action; \

this->recyclerLargeBlockPageAllocator.##action; \

this->recyclerWithBarrierPageAllocator.##action; \

this->threadPageAllocator->##action; \

}

// Macro that external objects referencing the recycler can use

#define ForEachRecyclerPageAllocatorIn(recycler, action) { \

recycler->GetRecyclerPageAllocator()->##action; \

recycler->GetRecyclerLargeBlockPageAllocator()->##action; \

recycler->GetRecyclerWithBarrierPageAllocator()->##action; \

recycler->GetRecyclerLeafPageAllocator()->##action; \

}

#else

// Macro to be used within the recycler

#define ForRecyclerPageAllocator(action) { \

this->recyclerPageAllocator.##action; \

this->recyclerLargeBlockPageAllocator.##action; \

this->threadPageAllocator->##action; \

}

// Macro that external objects referencing the recycler can use

#define ForEachRecyclerPageAllocatorIn(recycler, action) { \

recycler->GetRecyclerPageAllocator()->##action; \

recycler->GetRecyclerLargeBlockPageAllocator()->##action; \

recycler->GetRecyclerLeafPageAllocator()->##action; \

}

#endif

class RecyclerParallelThread

{

public:

typedef void (Recycler::\* WorkFunc)();

RecyclerParallelThread(Recycler \* recycler, WorkFunc workFunc) :

recycler(recycler),

workFunc(workFunc),

concurrentWorkReadyEvent(NULL),

concurrentWorkDoneEvent(NULL),

concurrentThread(NULL)

{

}

~RecyclerParallelThread()

{

Assert(concurrentThread == NULL);

Assert(concurrentWorkReadyEvent == NULL);

Assert(concurrentWorkDoneEvent == NULL);

}

bool StartConcurrent();

void WaitForConcurrent();

void Shutdown();

bool EnableConcurrent(bool synchronizeOnStartup);

private:

// Static entry point for thread creation

static unsigned int StaticThreadProc(LPVOID lpParameter);

// Static entry point for thread service usage

static void StaticBackgroundWorkCallback(void \* callbackData);

private:

WorkFunc workFunc;

Recycler \* recycler;

HANDLE concurrentWorkReadyEvent;// main thread uses this event to tell concurrent threads that the work is ready

HANDLE concurrentWorkDoneEvent;// concurrent threads use this event to tell main thread that the work allocated is done

HANDLE concurrentThread;

bool synchronizeOnStartup;

};

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

class AutoProtectPages

{

public:

AutoProtectPages(Recycler\* recycler, bool protectEnabled);

~AutoProtectPages();

void Unprotect();

private:

Recycler\* recycler;

bool isReadOnly;

};

#endif

class Recycler

{

friend class RecyclerScanMemoryCallback;

friend class RecyclerSweep;

friend class MarkContext;

friend class HeapBlock;

friend class HeapBlockMap32;

friend class RecyclerParallelThread;

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

friend class AutoProtectPages;

#endif

template <typename T> friend class RecyclerWeakReference;

template <typename T> friend class WeakReferenceHashTable;

template <typename TBlockType>

friend class SmallHeapBlockAllocator; // Needed for FindHeapBlock

#if defined(RECYCLER\_TRACE)

friend class JavascriptThreadService;

#endif

#ifdef HEAP\_ENUMERATION\_VALIDATION

friend class ActiveScriptProfilerHeapEnum;

#endif

friend class ScriptEngineBase; // This is for disabling GC for certain Host operations.

friend class CodeGenNumberThreadAllocator;

public:

static const uint ConcurrentThreadStackSize = 300000;

static const bool FakeZeroLengthArray = true;

#ifdef RECYCLER\_PAGE\_HEAP

// Keeping as constant in case we want to tweak the value here

// Set to 0 so that the tool can do the filtering instead of the runtime

static const int s\_numFramesToSkipForPageHeapAlloc = 0;

static const int s\_numFramesToSkipForPageHeapFree = 0;

static const int s\_numFramesToCaptureForPageHeap = 20;

#endif

uint Cookie;

class AutoEnterExternalStackSkippingGCMode

{

public:

AutoEnterExternalStackSkippingGCMode(Recycler\* recycler):

\_recycler(recycler)

{

// Setting this in a re-entrant mode is not allowed

Assert(!recycler->isExternalStackSkippingGC);

#if DBG

\_recycler->isExternalStackSkippingGC = true;

#endif

}

~AutoEnterExternalStackSkippingGCMode()

{

#if DBG

\_recycler->isExternalStackSkippingGC = false;

#endif

}

private:

Recycler\* \_recycler;

};

private:

class AutoSwitchCollectionStates

{

public:

AutoSwitchCollectionStates(Recycler\* recycler, CollectionState entryState, CollectionState exitState):

\_recycler(recycler),

\_exitState(exitState)

{

\_recycler->collectionState = entryState;

}

~AutoSwitchCollectionStates()

{

\_recycler->collectionState = \_exitState;

}

private:

Recycler\* \_recycler;

CollectionState \_exitState;

};

CollectionState collectionState;

IdleDecommitPageAllocator \* threadPageAllocator;

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

RecyclerPageAllocator recyclerWithBarrierPageAllocator;

#endif

RecyclerPageAllocator recyclerPageAllocator;

RecyclerPageAllocator recyclerLargeBlockPageAllocator;

JsUtil::ThreadService \*threadService;

HeapBlockMap heapBlockMap;

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

struct PinRecord

{

PinRecord() : refCount(0), stackBackTraces(nullptr) {}

PinRecord& operator=(uint newRefCount)

{

Assert(stackBackTraces == nullptr); Assert(newRefCount == 0); refCount = 0; return \*this;

}

PinRecord& operator++() { ++refCount; return \*this; }

PinRecord& operator--() { --refCount; return \*this; }

operator uint() const { return refCount; }

StackBackTraceNode \* stackBackTraces;

private:

uint refCount;

};

#else

typedef uint PinRecord;

#endif

typedef SimpleHashTable<void \*, PinRecord, HeapAllocator, DefaultComparer, true, PrimePolicy> PinnedObjectHashTable;

PinnedObjectHashTable pinnedObjectMap;

WeakReferenceHashTable<PrimePolicy> weakReferenceMap;

uint weakReferenceCleanupId;

void \* transientPinnedObject;

#if defined(CHECK\_MEMORY\_LEAK) || defined(LEAK\_REPORT)

StackBackTrace \* transientPinnedObjectStackBackTrace;

#endif

struct GuestArenaAllocator : public ArenaAllocator

{

GuestArenaAllocator(\_\_in LPCWSTR name, PageAllocator \* pageAllocator, void (\*outOfMemoryFunc)())

: ArenaAllocator(name, pageAllocator, outOfMemoryFunc), pendingDelete(false)

{

}

bool pendingDelete;

};

DListBase<GuestArenaAllocator> guestArenaList;

DListBase<ArenaData\*> externalGuestArenaList; // guest arenas are scanned for roots

HeapInfo autoHeap;

#ifdef RECYCLER\_PAGE\_HEAP

\_\_inline bool IsPageHeapEnabled() const { return isPageHeapEnabled; }

\_\_inline bool ShouldCapturePageHeapAllocStack() const { return capturePageHeapAllocStack; }

bool isPageHeapEnabled;

bool capturePageHeapAllocStack;

bool capturePageHeapFreeStack;

#else

\_\_inline const bool IsPageHeapEnabled() const { return false; }

\_\_inline bool ShouldCapturePageHeapAllocStack() const { return false; }

#endif

#ifdef RECYCLER\_MARK\_TRACK

MarkMap\* markMap;

CriticalSection markMapCriticalSection;

void PrintMarkMap();

void ClearMarkMap();

#endif

// Number of pages to reserve for the primary mark stack

// This is the minimum number of pages to guarantee that a single heap block

// can be rescanned in the worst possible case where every object in a heap block

// in the smallest bucket needs to be rescanned

// These many pages being reserved guarantees that in OOM Rescan, we can make progress

// on every rescan iteration

// We add one because there is a small amount of the page reserved for page pool metadata

// so we need to allocate an additional page to be sure

// Currently, this works out to 2 pages on 32-bit and 5 pages on 64-bit

static const int PrimaryMarkStackReservedPageCount =

((SmallAllocationBlockAttributes::PageCount \* MarkContext::MarkCandidateSize) / SmallAllocationBlockAttributes::MinObjectSize) + 1;

MarkContext markContext;

// Contexts for parallel marking.

// We support up to 4 way parallelism, main context + 3 additional parallel contexts.

MarkContext parallelMarkContext1;

MarkContext parallelMarkContext2;

MarkContext parallelMarkContext3;

// Page pools for above markContexts

PagePool markPagePool;

PagePool parallelMarkPagePool1;

PagePool parallelMarkPagePool2;

PagePool parallelMarkPagePool3;

bool IsMarkStackEmpty();

bool HasPendingMarkObjects() const { return markContext.HasPendingMarkObjects() || parallelMarkContext1.HasPendingMarkObjects() || parallelMarkContext2.HasPendingMarkObjects() || parallelMarkContext3.HasPendingMarkObjects(); }

bool HasPendingTrackObjects() const { return markContext.HasPendingTrackObjects() || parallelMarkContext1.HasPendingTrackObjects() || parallelMarkContext2.HasPendingTrackObjects() || parallelMarkContext3.HasPendingTrackObjects(); }

RecyclerCollectionWrapper \* collectionWrapper;

bool inDispose;

#if DBG

uint collectionCount;

#endif

#if DBG || defined RECYCLER\_TRACE

bool inResolveExternalWeakReferences;

#endif

bool allowDispose;

bool inDisposeWrapper;

bool needOOMRescan;

bool hasDisposableObject;

DWORD tickCountNextDispose;

bool hasPendingTransferDisposedObjects;

bool inExhaustiveCollection;

bool hasExhaustiveCandidate;

bool inCacheCleanupCollection;

bool inDecommitNowCollection;

bool isScriptActive;

bool isInScript;

bool isShuttingDown;

bool scanPinnedObjectMap;

bool hasScannedInitialImplicitRoots;

bool hasPendingUnpinnedObject;

bool hasPendingDeleteGuestArena;

bool inEndMarkOnLowMemory;

bool decommitOnFinish;

bool enableScanInteriorPointers;

bool enableScanImplicitRoots;

bool disableCollectOnAllocationHeuristics;

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

bool disableCollection;

#endif

#ifdef PARTIAL\_GC\_ENABLED

bool enablePartialCollect;

bool inPartialCollectMode;

bool hasBackgroundFinishPartial;

bool partialConcurrentNextCollection;

#ifdef RECYCLER\_STRESS

bool forcePartialScanStack;

bool recyclerStress;

#ifdef CONCURRENT\_GC\_ENABLED

bool recyclerBackgroundStress;

bool recyclerConcurrentStress;

bool recyclerConcurrentRepeatStress;

#endif

#ifdef PARTIAL\_GC\_ENABLED

bool recyclerPartialStress;

#endif

#endif

#endif

#ifdef CONCURRENT\_GC\_ENABLED

bool skipStack;

#if DBG

bool isConcurrentGCOnIdle;

bool isFinishGCOnIdle;

bool isExternalStackSkippingGC;

#endif

bool queueTrackedObject;

bool hasPendingConcurrentFindRoot;

bool priorityBoost;

bool disableConcurrent;

bool enableConcurrentMark;

bool enableParallelMark;

bool enableConcurrentSweep;

uint maxParallelism; // Max # of total threads to run in parallel

byte backgroundRescanCount; // for ETW events and stats

byte backgroundFinishMarkCount;

size\_t backgroundRescanRootBytes;

HANDLE concurrentWorkReadyEvent; // main thread uses this event to tell concurrent threads that the work is ready

HANDLE concurrentWorkDoneEvent; // concurrent threads use this event to tell main thread that the work allocated is done

HANDLE concurrentThread;

HANDLE mainThreadHandle;

class SavedRegisterState

{

public:

#if \_M\_IX86

static const int NumRegistersToSave = 8;

#elif \_M\_ARM

static const int NumRegistersToSave = 13;

#elif \_M\_ARM64

static const int NumRegistersToSave = 13;

#elif \_M\_AMD64

static const int NumRegistersToSave = 16;

#endif

SavedRegisterState()

{

memset(registers, 0, sizeof(void\*) \* NumRegistersToSave);

}

void\*\* GetRegisters()

{

return registers;

}

void\* GetStackTop()

{

// By convention, our register-saving routine will always

// save the stack pointer as the first item in the array

return registers[0];

}

private:

void\* registers[NumRegistersToSave];

};

void \* stackBase;

SavedRegisterState savedThreadContext;

template <uint parallelId>

void ParallelWorkFunc();

RecyclerParallelThread parallelThread1;

RecyclerParallelThread parallelThread2;

Js::ConfigFlagsTable& recyclerFlagsTable;

#if DBG

// Variable indicating if the concurrent thread has exited or not

// If the concurrent thread hasn't started yet, this is set to true

// Once the concurrent thread starts, it sets this to false,

// and when the concurrent thread exits, it sets this to true.

bool concurrentThreadExited;

bool disableConcurentThreadExitedCheck;

bool isProcessingTrackedObjects;

bool hasIncompletedDoCollect;

// This is set to true when we begin a Rescan, and set to false when either:

// (1) We finish the final in-thread Rescan and are about to Mark

// (2) We do a conditional ResetWriteWatch and are about to Mark

// When this flag is true, we should not be modifying existing mark-related state,

// including markBits and rescanState.

bool isProcessingRescan;

#endif

uint tickCountStartConcurrent;

bool isAborting;

#endif

RecyclerSweep recyclerSweepInstance;

RecyclerSweep \* recyclerSweep;

static const uint tickDiffToNextCollect = 300;

#ifdef IDLE\_DECOMMIT\_ENABLED

HANDLE concurrentIdleDecommitEvent;

DWORD needIdleDecommitSignal;

#endif

#ifdef PARTIAL\_GC\_ENABLED

SListBase<void \*> clientTrackedObjectList;

ArenaAllocator clientTrackedObjectAllocator;

size\_t partialUncollectedAllocBytes;

// Dynamic Heuristics for partial GC

size\_t uncollectedNewPageCountPartialCollect;

#endif

uint tickCountNextCollection;

uint tickCountNextFinishCollection;

void (\*outOfMemoryFunc)();

#ifdef RECYCLER\_TEST\_SUPPORT

BOOL (\*checkFn)(char\* addr, size\_t size);

#endif

ExternalRootMarker externalRootMarker;

void \* externalRootMarkerContext;

#ifdef PROFILE\_EXEC

Js::Profiler \* profiler;

Js::Profiler \* backgroundProfiler;

PageAllocator backgroundProfilerPageAllocator;

DListBase<ArenaAllocator> backgroundProfilerArena;

#endif

#ifdef PROFILE\_MEM

RecyclerMemoryData \* memoryData;

#endif

ThreadContextId mainThreadId;

#ifdef ENABLE\_BASIC\_TELEMETRY

Js::GCTelemetry gcTel;

#endif

#if DBG

uint heapBlockCount;

bool disableThreadAccessCheck;

#endif

#if DBG || defined(RECYCLER\_STATS)

bool isForceSweeping;

#endif

RecyclerWatsonTelemetryBlock localTelemetryBlock;

RecyclerWatsonTelemetryBlock \* telemetryBlock;

#ifdef RECYCLER\_STATS

RecyclerCollectionStats collectionStats;

void PrintHeapBlockStats(wchar\_t const \* name, HeapBlock::HeapBlockType type);

void PrintHeapBlockMemoryStats(wchar\_t const \* name, HeapBlock::HeapBlockType type);

void PrintCollectStats();

void PrintHeuristicCollectionStats();

void PrintMarkCollectionStats();

void PrintBackgroundCollectionStats();

void PrintMemoryStats();

void PrintBackgroundCollectionStat(RecyclerCollectionStats::MarkData const& markData);

#endif

#ifdef RECYCLER\_TRACE

CollectionParam collectionParam;

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

uint verifyPad;

bool verifyEnabled;

#endif

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

friend class RecyclerObjectGraphDumper;

RecyclerObjectGraphDumper \* objectGraphDumper;

public:

bool dumpObjectOnceOnCollect;

#endif

public:

Recycler(AllocationPolicyManager \* policyManager, IdleDecommitPageAllocator \* pageAllocator, void(\*outOfMemoryFunc)(), Js::ConfigFlagsTable& flags);

~Recycler();

void Initialize(const bool forceInThread, JsUtil::ThreadService \*threadService, const bool deferThreadStartup = false

#ifdef RECYCLER\_PAGE\_HEAP

, PageHeapMode pageheapmode = PageHeapMode::PageHeapModeOff

, bool captureAllocCallStack = false

, bool captureFreeCallStack = false

#endif

);

Js::ConfigFlagsTable& GetRecyclerFlagsTable() const { return this->recyclerFlagsTable; }

void SetMemProtectMode();

bool IsMemProtectMode()

{

return this->enableScanImplicitRoots;

}

size\_t GetUsedBytes()

{

size\_t usedBytes = threadPageAllocator->usedBytes;

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

usedBytes += recyclerWithBarrierPageAllocator.usedBytes;

#endif

usedBytes += recyclerPageAllocator.usedBytes;

usedBytes += recyclerLargeBlockPageAllocator.usedBytes;

return usedBytes;

}

void LogMemProtectHeapSize(bool fromGC);

char\* Realloc(void\* buffer, size\_t existingBytes, size\_t requestedBytes, bool truncate = true);

void SetTelemetryBlock(RecyclerWatsonTelemetryBlock \* telemetryBlock) { this->telemetryBlock = telemetryBlock; }

void Prime();

void\* GetOwnerContext() { return (void\*) this->collectionWrapper; }

PageAllocator \* GetPageAllocator() { return threadPageAllocator; }

bool NeedOOMRescan() const

{

return this->needOOMRescan;

}

void SetNeedOOMRescan()

{

this->needOOMRescan = true;

}

void ClearNeedOOMRescan()

{

this->needOOMRescan = false;

markContext.GetPageAllocator()->ResetDisableAllocationOutOfMemory();

parallelMarkContext1.GetPageAllocator()->ResetDisableAllocationOutOfMemory();

parallelMarkContext2.GetPageAllocator()->ResetDisableAllocationOutOfMemory();

parallelMarkContext3.GetPageAllocator()->ResetDisableAllocationOutOfMemory();

}

BOOL RequestConcurrentWrapperCallback();

BOOL CollectionInProgress() const

{

return collectionState != CollectionStateNotCollecting;

}

BOOL IsExiting() const

{

return (collectionState == Collection\_Exit);

}

BOOL IsSweeping() const

{

return ((collectionState & Collection\_Sweep) == Collection\_Sweep);

}

#ifdef RECYCLER\_PAGE\_HEAP

\_\_inline bool ShouldCapturePageHeapFreeStack() const { return capturePageHeapFreeStack; }

#else

\_\_inline bool ShouldCapturePageHeapFreeStack() const { return false; }

#endif

void SetIsThreadBound();

void SetIsScriptActive(bool isScriptActive)

{

Assert(this->isInScript);

Assert(this->isScriptActive != isScriptActive);

this->isScriptActive = isScriptActive;

if (isScriptActive)

{

this->tickCountNextDispose = ::GetTickCount() + RecyclerHeuristic::TickCountFinishCollection;

}

}

void SetIsInScript(bool isInScript)

{

Assert(this->isInScript != isInScript);

this->isInScript = isInScript;

}

bool ShouldIdleCollectOnExit();

void ScheduleNextCollection();

IdleDecommitPageAllocator \* GetRecyclerLeafPageAllocator()

{

return this->threadPageAllocator;

}

IdleDecommitPageAllocator \* GetRecyclerPageAllocator()

{

return &this->recyclerPageAllocator;

}

IdleDecommitPageAllocator \* GetRecyclerLargeBlockPageAllocator()

{

return &this->recyclerLargeBlockPageAllocator;

}

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

IdleDecommitPageAllocator \* GetRecyclerWithBarrierPageAllocator()

{

return &this->recyclerWithBarrierPageAllocator;

}

#endif

BOOL IsShuttingDown() const { return this->isShuttingDown; }

#ifdef CONCURRENT\_GC\_ENABLED

#if DBG

BOOL IsConcurrentMarkEnabled() const { return enableConcurrentMark; }

BOOL IsConcurrentSweepEnabled() const { return enableConcurrentSweep; }

#endif

template <CollectionFlags flags>

BOOL FinishConcurrent();

void ShutdownThread();

bool EnableConcurrent(JsUtil::ThreadService \*threadService, bool startAllThreads);

void DisableConcurrent();

void StartQueueTrackedObject();

bool DoQueueTrackedObject() const;

void PrepareSweep();

#endif

template <CollectionFlags flags>

void SetupPostCollectionFlags();

void EnsureNotCollecting();

bool QueueTrackedObject(FinalizableObject \* trackableObject);

// FindRoots

void TryMarkNonInterior(void\* candidate, void\* parentReference = nullptr);

void TryMarkInterior(void \*candidate, void\* parentReference = nullptr);

bool InCacheCleanupCollection() { return inCacheCleanupCollection; }

void ClearCacheCleanupCollection() { Assert(inCacheCleanupCollection); inCacheCleanupCollection = false; }

// Finalizer support

void SetExternalRootMarker(ExternalRootMarker fn, void \* context)

{

externalRootMarker = fn;

externalRootMarkerContext = context;

}

HeapInfo\* CreateHeap();

void DestroyHeap(HeapInfo\* heapInfo);

ArenaAllocator \* CreateGuestArena(wchar\_t const \* name, void (\*outOfMemoryFunc)());

void DeleteGuestArena(ArenaAllocator \* arenaAllocator);

ArenaData \*\* RegisterExternalGuestArena(ArenaData\* guestArena)

{

return externalGuestArenaList.PrependNode(&NoThrowHeapAllocator::Instance, guestArena);

}

void UnregisterExternalGuestArena(ArenaData\* guestArena)

{

externalGuestArenaList.Remove(&NoThrowHeapAllocator::Instance, guestArena);

}

void UnregisterExternalGuestArena(ArenaData\*\* guestArena)

{

externalGuestArenaList.RemoveElement(&NoThrowHeapAllocator::Instance, guestArena);

}

#ifdef RECYCLER\_TEST\_SUPPORT

void SetCheckFn(BOOL(\*checkFn)(char\* addr, size\_t size));

#endif

void SetCollectionWrapper(RecyclerCollectionWrapper \* wrapper)

{

this->collectionWrapper = wrapper;

#if LARGEHEAPBLOCK\_ENCODING

this->Cookie = wrapper->GetRandomNumber();

#else

this->Cookie = 0;

#endif

}

static size\_t GetAlignedSize(size\_t size) { return HeapInfo::GetAlignedSize(size); }

HeapInfo\* GetAutoHeap() { return &autoHeap; }

template <CollectionFlags flags>

BOOL CollectNow();

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

void DisplayMemStats();

#endif

void AddExternalMemoryUsage(size\_t size);

bool NeedDispose()

{

return this->hasDisposableObject;

}

template <CollectionFlags flags>

bool FinishDisposeObjectsNow();

BOOL ReportExternalMemoryAllocation(size\_t size);

void ReportExternalMemoryFailure(size\_t size);

void ReportExternalMemoryFree(size\_t size);

#ifdef TRACE\_OBJECT\_LIFETIME

#define DEFINE\_RECYCLER\_ALLOC\_TRACE(AllocFunc, AllocWithAttributesFunc, attributes) \

\_\_inline char\* AllocFunc##Trace(size\_t size) \

{ \

return AllocWithAttributesFunc<(ObjectInfoBits)(attributes | TraceBit), /\* nothrow = \*/ false>(size); \

}

#else

#define DEFINE\_RECYCLER\_ALLOC\_TRACE(AllocFunc, AllocWithAttributeFunc, attributes)

#endif

#define DEFINE\_RECYCLER\_ALLOC\_BASE(AllocFunc, AllocWithAttributesFunc, attributes) \

\_\_inline char \* AllocFunc(size\_t size) \

{ \

return AllocWithAttributesFunc<attributes, /\* nothrow = \*/ false>(size); \

} \

\_\_forceinline char \* AllocFunc##Inlined(size\_t size) \

{ \

return AllocWithAttributesFunc##Inlined<attributes, /\* nothrow = \*/ false>(size); \

} \

DEFINE\_RECYCLER\_ALLOC\_TRACE(AllocFunc, AllocWithAttributesFunc, attributes);

#define DEFINE\_RECYCLER\_NOTHROW\_ALLOC\_BASE(AllocFunc, AllocWithAttributesFunc, attributes) \

\_\_inline char \* NoThrow##AllocFunc(size\_t size) \

{ \

return AllocWithAttributesFunc<attributes, /\* nothrow = \*/ true>(size); \

} \

\_\_inline char \* NoThrow##AllocFunc##Inlined(size\_t size) \

{ \

return AllocWithAttributesFunc##Inlined<attributes, /\* nothrow = \*/ true>(size); \

} \

DEFINE\_RECYCLER\_ALLOC\_TRACE(AllocFunc, AllocWithAttributesFunc, attributes);

#define DEFINE\_RECYCLER\_ALLOC(AllocFunc, attributes) DEFINE\_RECYCLER\_ALLOC\_BASE(AllocFunc, AllocWithAttributes, attributes)

#define DEFINE\_RECYCLER\_ALLOC\_ZERO(AllocFunc, attributes) DEFINE\_RECYCLER\_ALLOC\_BASE(AllocFunc, AllocZeroWithAttributes, attributes)

#define DEFINE\_RECYCLER\_NOTHROW\_ALLOC(AllocFunc, attributes) DEFINE\_RECYCLER\_NOTHROW\_ALLOC\_BASE(AllocFunc, AllocWithAttributes, attributes)

#define DEFINE\_RECYCLER\_NOTHROW\_ALLOC\_ZERO(AllocFunc, attributes) DEFINE\_RECYCLER\_NOTHROW\_ALLOC\_BASE(AllocFunc, AllocZeroWithAttributes, attributes)

DEFINE\_RECYCLER\_ALLOC(Alloc, NoBit);

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC

DEFINE\_RECYCLER\_ALLOC(AllocWithBarrier, WithBarrierBit);

DEFINE\_RECYCLER\_ALLOC(AllocFinalizedWithBarrier, FinalizableWithBarrierObjectBits);

#endif

DEFINE\_RECYCLER\_ALLOC(AllocFinalized, FinalizableObjectBits);

DEFINE\_RECYCLER\_ALLOC(AllocFinalizedClientTracked, ClientFinalizableObjectBits);

// All trackable object are client trackable

DEFINE\_RECYCLER\_ALLOC(AllocTracked, ClientTrackableObjectBits);

DEFINE\_RECYCLER\_ALLOC(AllocLeaf, LeafBit);

DEFINE\_RECYCLER\_ALLOC(AllocFinalizedLeaf, FinalizableLeafBits);

DEFINE\_RECYCLER\_ALLOC(AllocTrackedLeaf, ClientTrackableLeafBits);

DEFINE\_RECYCLER\_ALLOC\_ZERO(AllocZero, NoBit);

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC

DEFINE\_RECYCLER\_ALLOC\_ZERO(AllocZeroWithBarrier, WithBarrierBit);

#endif

DEFINE\_RECYCLER\_ALLOC\_ZERO(AllocLeafZero, LeafBit);

DEFINE\_RECYCLER\_ALLOC\_ZERO(AllocZeroTrackedLeaf, ClientTrackableLeafBits);

DEFINE\_RECYCLER\_NOTHROW\_ALLOC\_ZERO(AllocImplicitRootLeaf, ImplicitRootLeafBits);

DEFINE\_RECYCLER\_NOTHROW\_ALLOC\_ZERO(AllocImplicitRoot, ImplicitRootBit);

template <ObjectInfoBits enumClass>

char \* AllocEnumClass(size\_t size)

{

Assert((enumClass & EnumClassMask) != 0);

Assert((enumClass & ~EnumClassMask) == 0);

return AllocWithAttributes<(ObjectInfoBits)(enumClass), /\* nothrow = \*/ false>(size);

}

template <ObjectInfoBits infoBits>

char \* AllocWithInfoBits(size\_t size)

{

return AllocWithAttributes<infoBits, /\* nothrow = \*/ false>(size);

}

template<typename T>

RecyclerWeakReference<T>\* CreateWeakReferenceHandle(T\* pStrongReference);

uint GetWeakReferenceCleanupId() const { return weakReferenceCleanupId; }

template<typename T>

bool FindOrCreateWeakReferenceHandle(T\* pStrongReference, RecyclerWeakReference<T> \*\*ppWeakRef);

template<typename T>

bool TryGetWeakReferenceHandle(T\* pStrongReference, RecyclerWeakReference<T> \*\*weakReference);

template <ObjectInfoBits attributes>

char\* GetAddressOfAllocator(size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedSmallObjectSize(sizeCat));

return (char\*)this->autoHeap.GetBucket<attributes>(sizeCat).GetAllocator();

}

template <ObjectInfoBits attributes>

uint32 GetEndAddressOffset(size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedSmallObjectSize(sizeCat));

return this->autoHeap.GetBucket<attributes>(sizeCat).GetAllocator()->GetEndAddressOffset();

}

template <ObjectInfoBits attributes>

uint32 GetFreeObjectListOffset(size\_t sizeCat)

{

Assert(HeapInfo::IsAlignedSmallObjectSize(sizeCat));

return this->autoHeap.GetBucket<attributes>(sizeCat).GetAllocator()->GetFreeObjectListOffset();

}

void GetNormalHeapBlockAllocatorInfoForNativeAllocation(size\_t sizeCat, void\*& allocatorAddress, uint32& endAddressOffset, uint32& freeListOffset);

bool AllowNativeCodeBumpAllocation();

static void TrackNativeAllocatedMemoryBlock(Recycler \* recycler, void \* memBlock, size\_t sizeCat);

void Free(void\* buffer, size\_t size)

{

Assert(false);

}

bool ExplicitFreeLeaf(void\* buffer, size\_t size);

bool ExplicitFreeNonLeaf(void\* buffer, size\_t size);

template <ObjectInfoBits attributes>

bool ExplicitFreeInternalWrapper(void\* buffer, size\_t allocSize);

template <ObjectInfoBits attributes, typename TBlockAttributes>

bool ExplicitFreeInternal(void\* buffer, size\_t size, size\_t sizeCat);

size\_t GetAllocSize(size\_t size);

template <typename TBlockAttributes>

void SetExplicitFreeBitOnSmallBlock(HeapBlock\* heapBlock, size\_t sizeCat, void\* buffer, ObjectInfoBits attributes);

char\* HeapAllocR(HeapInfo\* eHeap, size\_t size)

{

return RealAlloc<LeafBit, /\* nothrow = \*/ false>(eHeap, size);

}

void HeapFree(HeapInfo\* eHeap,void\* candidate);

void EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size));

void RootAddRef(void\* obj, uint \*count = nullptr);

void RootRelease(void\* obj, uint \*count = nullptr);

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char\* RealAlloc(HeapInfo\* heap, size\_t size);

template <ObjectInfoBits attributes, bool isSmallAlloc, bool nothrow>

\_\_inline char\* RealAllocFromBucket(HeapInfo\* heap, size\_t size);

void EnterIdleDecommit();

void LeaveIdleDecommit();

void DisposeObjects();

BOOL IsValidObject(void\* candidate, size\_t minimumSize = 0);

#if DBG

void SetDisableThreadAccessCheck();

void SetDisableConcurentThreadExitedCheck();

void CheckAllocExternalMark() const;

BOOL IsFreeObject(void \* candidate);

BOOL IsReentrantState() const;

#endif

#if DBG\_DUMP

void PrintMarkStack();

#endif

#ifdef PROFILE\_EXEC

Js::Profiler \* GetProfiler() const { return this->profiler; }

ArenaAllocator \* AddBackgroundProfilerArena();

void ReleaseBackgroundProfilerArena(ArenaAllocator \* arena);

void SetProfiler(Js::Profiler \* profiler, Js::Profiler \* backgroundProfiler);

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

BOOL VerifyEnabled() const { return verifyEnabled; }

void Verify(Js::Phase phase);

static void VerifyCheck(BOOL cond, wchar\_t const \* msg, void \* address, void \* corruptedAddress);

static void VerifyCheckFill(void \* address, size\_t size);

void FillCheckPad(void \* address, size\_t size, size\_t alignedAllocSize, bool objectAlreadyInitialized);

void FillCheckPad(void \* address, size\_t size, size\_t alignedAllocSize)

{

FillCheckPad(address, size, alignedAllocSize, false);

}

void VerifyCheckPad(void \* address, size\_t size);

void VerifyCheckPadExplicitFreeList(void \* address, size\_t size);

static const byte VerifyMemFill = 0xCA;

#endif

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

void VerifyZeroFill(void \* address, size\_t size);

#endif

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

bool DumpObjectGraph(RecyclerObjectGraphDumper::Param \* param = nullptr);

void DumpObjectDescription(void \*object);

#endif

#ifdef LEAK\_REPORT

void ReportLeaks();

void ReportLeaksOnProcessDetach();

#endif

#ifdef CHECK\_MEMORY\_LEAK

void CheckLeaks(wchar\_t const \* header);

void CheckLeaksOnProcessDetach(wchar\_t const \* header);

#endif

#ifdef RECYCLER\_TRACE

void SetDomCollect(bool isDomCollect)

{

collectionParam.domCollect = isDomCollect;

}

void CaptureCollectionParam(CollectionFlags flags, bool repeat = false);

#endif

#ifdef ENABLE\_BASIC\_TELEMETRY

Js::GCPauseStats GetGCPauseStats()

{

return gcTel.GetGCPauseStats(); // returns the maxGCpause time in ms

}

void ResetGCPauseStats()

{

gcTel.Reset();

}

void SetIsScriptSiteCloseGC(bool val)

{

gcTel.SetIsScriptSiteCloseGC(val);

}

#endif

private:

// RecyclerRootPtr has implicit conversion to pointers, prevent it to be

// passed to RootAddRef/RootRelease directly

template <typename T>

void RootAddRef(RecyclerRootPtr<T>& ptr, uint \*count = nullptr);

template <typename T>

void RootRelease(RecyclerRootPtr<T>& ptr, uint \*count = nullptr);

template <CollectionFlags flags>

BOOL CollectInternal();

template <CollectionFlags flags>

BOOL Collect();

template <CollectionFlags flags>

BOOL CollectWithHeuristic();

template <CollectionFlags flags>

BOOL CollectWithExhaustiveCandidate();

template <CollectionFlags flags>

BOOL GetPartialFlag();

bool NeedExhaustiveRepeatCollect() const;

#if DBG

bool ExpectStackSkip() const;

#endif

static size\_t const InvalidScanRootBytes = (size\_t)-1;

// Small Allocator

template <typename SmallHeapBlockAllocatorType>

void AddSmallAllocator(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat);

template <typename SmallHeapBlockAllocatorType>

void RemoveSmallAllocator(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat);

template <ObjectInfoBits attributes, typename SmallHeapBlockAllocatorType>

char \* SmallAllocatorAlloc(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat);

// Allocation

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char \* AllocWithAttributesInlined(size\_t size);

template <ObjectInfoBits attributes, bool nothrow>

char \* AllocWithAttributes(size\_t size)

{

return AllocWithAttributesInlined<attributes, nothrow>(size);

}

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char\* AllocZeroWithAttributesInlined(size\_t size);

template <ObjectInfoBits attributes, bool nothrow>

char\* AllocZeroWithAttributes(size\_t size)

{

return AllocZeroWithAttributesInlined<attributes, nothrow>(size);

}

char\* AllocWeakReferenceEntry(size\_t size)

{

return AllocWithAttributes<WeakReferenceEntryBits, /\* nothrow = \*/ false>(size);

}

bool NeedDisposeTimed()

{

DWORD ticks = ::GetTickCount();

return (ticks > tickCountNextDispose && this->hasDisposableObject);

}

char\* TryLargeAlloc(HeapInfo\* heap, size\_t size, ObjectInfoBits attributes, bool nothrow);

template <bool nothrow>

char\* LargeAlloc(HeapInfo\* heap, size\_t size, ObjectInfoBits attributes);

void OutOfMemory();

// Collection

BOOL DoCollect(CollectionFlags flags);

BOOL DoCollectWrapped(CollectionFlags flags);

BOOL CollectOnAllocatorThread();

#if DBG

void ResetThreadId();

#endif

template <bool background>

size\_t ScanPinnedObjects();

size\_t ScanStack();

size\_t ScanArena(ArenaData \* alloc, bool background);

void ScanImplicitRoots();

void ScanInitialImplicitRoots();

void ScanNewImplicitRoots();

size\_t FindRoots();

size\_t TryMarkArenaMemoryBlockList(ArenaMemoryBlock \* memoryBlocks);

size\_t TryMarkBigBlockList(BigBlock \* memoryBlocks);

size\_t TryMarkBigBlockListWithWriteWatch(BigBlock \* memoryBlocks);

// Mark

void ResetMarks(ResetMarkFlags flags);

void Mark();

bool EndMark();

bool EndMarkCheckOOMRescan();

void EndMarkOnLowMemory();

void DoParallelMark();

void DoBackgroundParallelMark();

size\_t RootMark(CollectionState markState);

void ProcessMark(bool background);

void ProcessParallelMark(bool background, MarkContext \* markContext);

template <bool parallel, bool interior>

void ProcessMarkContext(MarkContext \* markContext);

public:

bool IsObjectMarked(void\* candidate) { return this->heapBlockMap.IsMarked(candidate); }

#ifdef RECYCLER\_STRESS

bool StressCollectNow();

#endif

private:

HeapBlock\* FindHeapBlock(void \* candidate);

struct FindBlockCache

{

FindBlockCache():

heapBlock(nullptr),

candidate(nullptr)

{

}

HeapBlock\* heapBlock;

void\* candidate;

} blockCache;

\_\_inline void ScanObjectInline(void \*\* obj, size\_t byteCount);

\_\_inline void ScanObjectInlineInterior(void \*\* obj, size\_t byteCount);

\_\_inline void ScanMemoryInline(void \*\* obj, size\_t byteCount);

void ScanMemory(void \*\* obj, size\_t byteCount) { if (byteCount != 0) { ScanMemoryInline(obj, byteCount); } }

bool AddMark(void \* candidate, size\_t byteCount);

// Sweep

bool Sweep(size\_t rescanRootBytes = (size\_t)-1, bool concurrent = false, bool adjustPartialHeuristics = false);

void SweepWeakReference();

void SweepHeap(bool concurrent, RecyclerSweep& recyclerSweep);

void FinishSweep(RecyclerSweep& recyclerSweep);

bool FinishDisposeObjects();

template <CollectionFlags flags>

bool FinishDisposeObjectsWrapped();

// end collection

void FinishCollection();

void FinishCollection(bool needConcurrentSweep);

void EndCollection();

void ResetCollectionState();

void ResetMarkCollectionState();

void ResetHeuristicCounters();

void ResetPartialHeuristicCounters();

BOOL IsMarkState() const;

BOOL IsFindRootsState() const;

BOOL IsInThreadFindRootsState() const;

template <Js::Phase phase>

void CollectionBegin();

template <Js::Phase phase>

void CollectionEnd();

#ifdef PARTIAL\_GC\_ENABLED

void ProcessClientTrackedObjects();

bool PartialCollect(bool concurrent);

void FinishPartialCollect(RecyclerSweep \* recyclerSweep = nullptr);

void ClearPartialCollect();

void BackgroundFinishPartialCollect(RecyclerSweep \* recyclerSweep);

#endif

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

size\_t RescanMark(DWORD waitTime);

size\_t FinishMark(DWORD waitTime);

size\_t FinishMarkRescan(bool background);

void ProcessTrackedObjects();

#endif

#ifdef CONCURRENT\_GC\_ENABLED

// Concurrent GC

BOOL IsConcurrentEnabled() const { return this->enableConcurrentMark || this->enableParallelMark || this->enableConcurrentSweep; }

BOOL IsConcurrentMarkState() const;

BOOL IsConcurrentMarkExecutingState() const;

BOOL IsConcurrentResetMarksState() const;

BOOL IsConcurrentFindRootState() const;

BOOL IsConcurrentExecutingState() const;

BOOL IsConcurrentSweepExecutingState() const;

BOOL IsConcurrentState() const;

BOOL InConcurrentSweep()

{

return ((collectionState & Collection\_ConcurrentSweep) == Collection\_ConcurrentSweep);

}

#if DBG

BOOL IsConcurrentFinishedState() const;

#endif // DBG

bool InitializeConcurrent(JsUtil::ThreadService\* threadService);

bool AbortConcurrent(bool restoreState);

void FinalizeConcurrent(bool restoreState);

static unsigned int StaticThreadProc(LPVOID lpParameter);

static int ExceptFilter(LPEXCEPTION\_POINTERS pEP);

DWORD ThreadProc();

void DoBackgroundWork(bool forceForeground = false);

static void StaticBackgroundWorkCallback(void \* callbackData);

BOOL CollectOnConcurrentThread();

bool StartConcurrent(CollectionState const state);

BOOL StartBackgroundMarkCollect();

BOOL StartSynchronousBackgroundMark();

BOOL StartAsynchronousBackgroundMark();

BOOL StartBackgroundMark(bool foregroundResetMark, bool foregroundFindRoots);

BOOL StartConcurrentSweepCollect();

template <CollectionFlags flags>

BOOL TryFinishConcurrentCollect();

BOOL WaitForConcurrentThread(DWORD waitTime);

void FlushBackgroundPages();

BOOL FinishConcurrentCollect(CollectionFlags flags);

BOOL FinishConcurrentCollectWrapped(CollectionFlags flags);

void BackgroundMark();

void BackgroundResetMarks();

void PrepareBackgroundFindRoots();

void RevertPrepareBackgroundFindRoots();

size\_t BackgroundFindRoots();

size\_t BackgroundScanStack();

size\_t BackgroundRepeatMark();

size\_t BackgroundRescan(RescanFlags rescanFlags);

void BackgroundResetWriteWatchAll();

size\_t BackgroundFinishMark();

char\* GetScriptThreadStackTop();

void SweepPendingObjects(RecyclerSweep& recyclerSweep);

void ConcurrentTransferSweptObjects(RecyclerSweep& recyclerSweep);

#ifdef PARTIAL\_GC\_ENABLED

void ConcurrentPartialTransferSweptObjects(RecyclerSweep& recyclerSweep);

#endif // PARTIAL\_GC\_ENABLED

#endif // CONCURRENT\_GC\_ENABLED

bool ForceSweepObject();

void NotifyFree(\_\_in char \* address, size\_t size);

template <bool pageheap, typename T>

void NotifyFree(T \* heapBlock);

void CleanupPendingUnroot();

#ifdef ENABLE\_JS\_ETW

ULONG EventWriteFreeMemoryBlock(HeapBlock\* heapBlock);

void FlushFreeRecord();

void AppendFreeMemoryETWRecord(\_\_in char \*address, size\_t size);

static const uint BulkFreeMemoryCount = 400;

uint bulkFreeMemoryWrittenCount;

struct ETWFreeRecord {

char\* memoryAddress;

uint32 objectSize;

};

ETWFreeRecord etwFreeRecords[BulkFreeMemoryCount];

#endif

template <ObjectInfoBits attributes>

bool IntegrateBlock(char \* blockAddress, PageSegment \* segment, size\_t allocSize, size\_t objectSize);

template <class TBlockAttributes> friend class SmallHeapBlockT;

template <class TBlockAttributes> friend class SmallNormalHeapBlockT;

template <class TBlockAttributes> friend class SmallLeafHeapBlockT;

template <class TBlockAttributes> friend class SmallFinalizableHeapBlockT;

friend class LargeHeapBlock;

friend class HeapInfo;

friend class LargeHeapBucket;

template <typename TBlockType>

friend class HeapBucketT;

template <typename TBlockType>

friend class SmallNormalHeapBucketBase;

template <typename T, ObjectInfoBits attributes = LeafBit>

friend class RecyclerFastAllocator;

#ifdef RECYCLER\_TRACE

void PrintCollectTrace(Js::Phase phase, bool finish = false, bool noConcurrentWork = false);

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void VerifyMark();

void VerifyMarkRoots();

void VerifyMarkStack();

void VerifyMarkArena(ArenaData \* arena);

void VerifyMarkBigBlockList(BigBlock \* memoryBlocks);

void VerifyMarkArenaMemoryBlockList(ArenaMemoryBlock \* memoryBlocks);

void VerifyMark(void \* address);

#endif

#if DBG\_DUMP

bool forceTraceMark;

#endif

bool isHeapEnumInProgress;

#if DBG

bool allowAllocationDuringHeapEnum;

bool allowAllocationDuringRenentrance;

#ifdef ENABLE\_PROJECTION

bool isInRefCountTrackingForProjection;

#endif

#endif

// There are two scenarios we allow limited allocation but disallow GC during those allocations:

// in heapenum when we allocate PropertyRecord, and

// in projection ExternalMark allowing allocating VarToDispEx. This is the common flag

// while we have debug only flag for each of the two scenarios.

bool isCollectionDisabled;

#ifdef TRACK\_ALLOC

public:

Recycler \* TrackAllocInfo(TrackAllocData const& data);

void ClearTrackAllocInfo(TrackAllocData\* data = NULL);

#ifdef PROFILE\_RECYCLER\_ALLOC

void PrintAllocStats();

private:

static bool DoProfileAllocTracker();

void InitializeProfileAllocTracker();

void TrackUnallocated(\_\_in char\* address, \_\_in char \*endAddress, size\_t sizeCat);

void TrackAllocCore(void \* object, size\_t size, const TrackAllocData& trackAllocData, bool traceLifetime = false);

void\* TrackAlloc(void \* object, size\_t size, const TrackAllocData& trackAllocData, bool traceLifetime = false);

void TrackIntegrate(\_\_in\_ecount(blockSize) char \* blockAddress, size\_t blockSize, size\_t allocSize, size\_t objectSize, const TrackAllocData& trackAllocData);

BOOL TrackFree(const char\* address, size\_t size);

void TrackAllocWeakRef(RecyclerWeakReferenceBase \* weakRef);

void TrackFreeWeakRef(RecyclerWeakReferenceBase \* weakRef);

struct TrackerData

{

TrackerData(type\_info const \* typeinfo, bool isArray) : typeinfo(typeinfo), isArray(isArray),

ItemSize(0), ItemCount(0), AllocCount(0), ReqSize(0), AllocSize(0), FreeCount(0), FreeSize(0), TraceLifetime(false)

#ifdef PERF\_COUNTERS

, counter(PerfCounter::RecyclerTrackerCounterSet::GetPerfCounter(typeinfo, isArray))

, sizeCounter(PerfCounter::RecyclerTrackerCounterSet::GetPerfSizeCounter(typeinfo, isArray))

#endif

{

}

type\_info const \* typeinfo;

bool isArray;

#ifdef TRACE\_OBJECT\_LIFETIME

bool TraceLifetime;

#endif

size\_t ItemSize;

size\_t ItemCount;

int AllocCount;

int64 ReqSize;

int64 AllocSize;

int FreeCount;

int64 FreeSize;

#ifdef PERF\_COUNTERS

PerfCounter::Counter& counter;

PerfCounter::Counter& sizeCounter;

#endif

static TrackerData EmptyData;

static TrackerData ExplicitFreeListObjectData;

};

TrackerData \* GetTrackerData(void \* address);

void SetTrackerData(void \* address, TrackerData \* data);

struct TrackerItem

{

TrackerItem(type\_info const \* typeinfo) : instanceData(typeinfo, false), arrayData(typeinfo, true)

#ifdef PERF\_COUNTERS

, weakRefCounter(PerfCounter::RecyclerTrackerCounterSet::GetWeakRefPerfCounter(typeinfo))

#endif

{}

TrackerData instanceData;

TrackerData arrayData;

#ifdef PERF\_COUNTERS

PerfCounter::Counter& weakRefCounter;

#endif

};

typedef JsUtil::BaseDictionary<type\_info const \*, TrackerItem \*, NoCheckHeapAllocator, PrimeSizePolicy, DefaultComparer, JsUtil::SimpleDictionaryEntry, JsUtil::NoResizeLock> TypeInfotoTrackerItemMap;

typedef JsUtil::BaseDictionary<void \*, TrackerData \*, NoCheckHeapAllocator, PrimeSizePolicy, RecyclerPointerComparer, JsUtil::SimpleDictionaryEntry, JsUtil::NoResizeLock> PointerToTrackerDataMap;

TypeInfotoTrackerItemMap \* trackerDictionary;

CRITICAL\_SECTION trackerCriticalSection;

#endif

TrackAllocData nextAllocData;

#endif

public:

// Enumeration

class AutoSetupRecyclerForNonCollectingMark

{

private:

Recycler& m\_recycler;

bool m\_setupDone;

CollectionState m\_previousCollectionState;

#ifdef RECYCLER\_STATS

RecyclerCollectionStats m\_previousCollectionStats;

#endif

public:

AutoSetupRecyclerForNonCollectingMark(Recycler& recycler, bool setupForHeapEnumeration = false);

~AutoSetupRecyclerForNonCollectingMark();

void DoCommonSetup();

void SetupForHeapEnumeration();

};

friend class RecyclerHeapObjectInfo;

bool FindImplicitRootObject(void\* candidate, RecyclerHeapObjectInfo& heapObject);

bool FindHeapObject(void\* candidate, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject);

bool FindHeapObjectWithClearedAllocators(void\* candidate, RecyclerHeapObjectInfo& heapObject);

bool IsCollectionDisabled() const { return isCollectionDisabled; }

bool IsHeapEnumInProgress() const { Assert(isHeapEnumInProgress ? isCollectionDisabled : true); return isHeapEnumInProgress; }

#if DBG

// There are limited cases that we have to allow allocation during heap enumeration. GC is explicitly

// disabled during heap enumeration for these limited cases. (See DefaultRecyclerCollectionWrapper)

// The only case of allocation right now is allocating property record for string based type handler

// so we can use the propertyId as the relation Id.

// Allocation during enumeration is still frown upon and should still be avoid if possible.

bool AllowAllocationDuringHeapEnum() const { return allowAllocationDuringHeapEnum; }

class AutoAllowAllocationDuringHeapEnum : public AutoBooleanToggle

{

public:

AutoAllowAllocationDuringHeapEnum(Recycler \* recycler) : AutoBooleanToggle(&recycler->allowAllocationDuringHeapEnum) {};

};

#ifdef ENABLE\_PROJECTION

bool IsInRefCountTrackingForProjection() const { return isInRefCountTrackingForProjection;}

class AutoIsInRefCountTrackingForProjection : public AutoBooleanToggle

{

public:

AutoIsInRefCountTrackingForProjection(Recycler \* recycler) : AutoBooleanToggle(&recycler->isInRefCountTrackingForProjection) {};

};

#endif

#endif

class AutoAllowAllocationDuringReentrance : public AutoBooleanToggle

{

public:

AutoAllowAllocationDuringReentrance(Recycler \* recycler) :

AutoBooleanToggle(&recycler->isCollectionDisabled)

#if DBG

, allowAllocationDuringRenentrance(&recycler->allowAllocationDuringRenentrance)

#endif

{};

#if DBG

private:

AutoBooleanToggle allowAllocationDuringRenentrance;

#endif

};

#ifdef HEAP\_ENUMERATION\_VALIDATION

typedef void(\*PostHeapEnumScanCallback)(const HeapObject& heapObject, void \*data);

PostHeapEnumScanCallback pfPostHeapEnumScanCallback;

void \*postHeapEnunScanData;

void PostHeapEnumScan(PostHeapEnumScanCallback callback, void\*data);

bool IsPostEnumHeapValidationInProgress() const { return pfPostHeapEnumScanCallback != NULL; }

#endif

private:

void\* GetRealAddressFromInterior(void\* candidate);

void BeginNonCollectingMark();

void EndNonCollectingMark();

#if defined(RECYCLER\_DUMP\_OBJECT\_GRAPH) || defined(LEAK\_REPORT) || defined(CHECK\_MEMORY\_LEAK)

public:

bool IsInDllCanUnloadNow() const { return inDllCanUnloadNow; }

bool IsInDetachProcess() const { return inDetachProcess; }

void SetInDllCanUnloadNow();

void SetInDetachProcess();

private:

bool inDllCanUnloadNow;

bool inDetachProcess;

bool isPrimaryMarkContextInitialized;

#endif

#if defined(LEAK\_REPORT) || defined(CHECK\_MEMORY\_LEAK)

template <class Fn>

void ReportOnProcessDetach(Fn fn);

void PrintPinnedObjectStackTraces();

#endif

public:

typedef void (CALLBACK \*ObjectBeforeCollectCallback)(void\* object, void\* callbackState); // same as jsrt JsObjectBeforeCollectCallback

void SetObjectBeforeCollectCallback(void\* object, ObjectBeforeCollectCallback callback, void\* callbackState);

void ClearObjectBeforeCollectCallbacks();

bool IsInObjectBeforeCollectCallback() const { return objectBeforeCollectCallbackState != ObjectBeforeCollectCallback\_None; }

private:

struct ObjectBeforeCollectCallbackData

{

ObjectBeforeCollectCallback callback;

void\* callbackState;

ObjectBeforeCollectCallbackData() {}

ObjectBeforeCollectCallbackData(ObjectBeforeCollectCallback callback, void\* callbackState) : callback(callback), callbackState(callbackState) {}

};

typedef JsUtil::BaseDictionary<void\*, ObjectBeforeCollectCallbackData, HeapAllocator,

PrimeSizePolicy, RecyclerPointerComparer, JsUtil::SimpleDictionaryEntry, JsUtil::NoResizeLock> ObjectBeforeCollectCallbackMap;

ObjectBeforeCollectCallbackMap\* objectBeforeCollectCallbackMap;

enum ObjectBeforeCollectCallbackState

{

ObjectBeforeCollectCallback\_None,

ObjectBeforeCollectCallback\_Normal, // Normal GC BeforeCollect callback

ObjectBeforeCollectCallback\_Shutdown, // At shutdown invoke all BeforeCollect callback

} objectBeforeCollectCallbackState;

bool ProcessObjectBeforeCollectCallbacks(bool atShutdown = false);

};

class RecyclerHeapObjectInfo

{

void\* m\_address;

Recycler \* m\_recycler;

HeapBlock\* m\_heapBlock;

#if LARGEHEAPBLOCK\_ENCODING

union

{

byte \* m\_attributes;

LargeObjectHeader \* m\_largeHeapBlockHeader;

};

bool isUsingLargeHeapBlock = false;

#else

byte \* m\_attributes;

#endif

public:

RecyclerHeapObjectInfo() : m\_address(NULL), m\_recycler(NULL), m\_heapBlock(NULL), m\_attributes(NULL) {}

RecyclerHeapObjectInfo(void\* address, Recycler \* recycler, HeapBlock\* heapBlock, byte \* attributes) :

m\_address(address), m\_recycler(recycler), m\_heapBlock(heapBlock), m\_attributes(attributes) { }

void\* GetObjectAddress() const { return m\_address; }

bool IsLeaf() const

{

#if LARGEHEAPBLOCK\_ENCODING

if (isUsingLargeHeapBlock)

{

return (m\_largeHeapBlockHeader->GetAttributes(m\_recycler->Cookie) & LeafBit) != 0;

}

#endif

return ((\*m\_attributes & LeafBit) != 0 || this->m\_heapBlock->IsLeafBlock());

}

bool IsImplicitRoot() const

{

#if LARGEHEAPBLOCK\_ENCODING

if (isUsingLargeHeapBlock)

{

return (m\_largeHeapBlockHeader->GetAttributes(m\_recycler->Cookie) & ImplicitRootBit) != 0;

}

#endif

return (\*m\_attributes & ImplicitRootBit) != 0;

}

bool IsObjectMarked() const { Assert(m\_recycler); return m\_recycler->heapBlockMap.IsMarked(m\_address); }

void SetObjectMarked() { Assert(m\_recycler); m\_recycler->heapBlockMap.SetMark(m\_address); }

ObjectInfoBits GetAttributes() const

{

#if LARGEHEAPBLOCK\_ENCODING

if (isUsingLargeHeapBlock)

{

return (ObjectInfoBits)m\_largeHeapBlockHeader->GetAttributes(m\_recycler->Cookie);

}

#endif

return (ObjectInfoBits)\*m\_attributes;

}

size\_t GetSize() const;

#if LARGEHEAPBLOCK\_ENCODING

void SetLargeHeapBlockHeader(LargeObjectHeader \* largeHeapBlockHeader)

{

m\_largeHeapBlockHeader = largeHeapBlockHeader;

isUsingLargeHeapBlock = true;

}

#endif

bool SetMemoryProfilerHasEnumerated()

{

Assert(m\_heapBlock);

#if LARGEHEAPBLOCK\_ENCODING

if (isUsingLargeHeapBlock)

{

return SetMemoryProfilerHasEnumeratedForLargeHeapBlock();

}

#endif

bool wasMemoryProfilerOldObject = (\*m\_attributes & MemoryProfilerOldObjectBit) != 0;

\*m\_attributes |= MemoryProfilerOldObjectBit;

return wasMemoryProfilerOldObject;

}

bool ClearImplicitRootBit()

{

// This can only be called on the main thread for non-finalizable block

// As finalizable block requires that the bit not be change during concurrent mark

// since the background thread change the NewTrackBit

Assert(!m\_heapBlock->IsAnyFinalizableBlock());

#ifdef RECYCLER\_PAGE\_HEAP

Recycler\* recycler = this->m\_recycler;

if (recycler->ShouldCapturePageHeapFreeStack())

{

Assert(recycler->IsPageHeapEnabled());

this->m\_heapBlock->CapturePageHeapFreeStack();

}

#endif

#if LARGEHEAPBLOCK\_ENCODING

if (isUsingLargeHeapBlock)

{

return ClearImplicitRootBitsForLargeHeapBlock();

}

#endif

Assert(m\_attributes);

bool wasImplicitRoot = (\*m\_attributes & ImplicitRootBit) != 0;

\*m\_attributes &= ~ImplicitRootBit;

return wasImplicitRoot;

}

void ExplicitFree()

{

if (\*m\_attributes == ObjectInfoBits::LeafBit)

{

m\_recycler->ExplicitFreeLeaf(m\_address, GetSize());

}

else

{

Assert(\*m\_attributes == ObjectInfoBits::NoBit);

m\_recycler->ExplicitFreeNonLeaf(m\_address, GetSize());

}

}

#if LARGEHEAPBLOCK\_ENCODING

bool ClearImplicitRootBitsForLargeHeapBlock()

{

Assert(m\_largeHeapBlockHeader);

byte attributes = m\_largeHeapBlockHeader->GetAttributes(m\_recycler->Cookie);

bool wasImplicitRoot = (attributes & ImplicitRootBit) != 0;

m\_largeHeapBlockHeader->SetAttributes(m\_recycler->Cookie, attributes & ~ImplicitRootBit);

return wasImplicitRoot;

}

bool SetMemoryProfilerHasEnumeratedForLargeHeapBlock()

{

Assert(m\_largeHeapBlockHeader);

byte attributes = m\_largeHeapBlockHeader->GetAttributes(m\_recycler->Cookie);

bool wasMemoryProfilerOldObject = (attributes & MemoryProfilerOldObjectBit) != 0;

m\_largeHeapBlockHeader->SetAttributes(m\_recycler->Cookie, attributes | MemoryProfilerOldObjectBit);

return wasMemoryProfilerOldObject;

}

#endif

};

// A fake heap block to replace the original heap block where the strong ref is when it has been collected

// as the original heap block may have been freed

class CollectedRecyclerWeakRefHeapBlock : public HeapBlock

{

public:

#if DBG

virtual BOOL IsFreeObject(void\* objectAddress) override { Assert(false); return false; }

#endif

virtual BOOL IsValidObject(void\* objectAddress) override { Assert(false); return false; }

virtual byte\* GetRealAddressFromInterior(void\* interiorAddress) override { Assert(false); return nullptr; }

virtual size\_t GetObjectSize(void\* object) override { Assert(false); return 0; }

virtual bool FindHeapObject(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject) override { Assert(false); return false; }

virtual bool TestObjectMarkedBit(void\* objectAddress) override { Assert(false); return false; }

virtual void SetObjectMarkedBit(void\* objectAddress) override { Assert(false); }

#ifdef RECYCLER\_VERIFY\_MARK

virtual void VerifyMark(void \* objectAddress) override { Assert(false); }

#endif

#ifdef RECYCLER\_PERF\_COUNTERS

virtual void UpdatePerfCountersOnFree() override { Assert(false); }

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

virtual void \* GetTrackerData(void \* address) override { Assert(false); return nullptr; }

virtual void SetTrackerData(void \* address, void \* data) override { Assert(false); }

#endif

static CollectedRecyclerWeakRefHeapBlock Instance;

private:

CollectedRecyclerWeakRefHeapBlock() : HeapBlock(BlockTypeCount) { isPendingConcurrentSweep = false; }

};

class AutoIdleDecommit

{

public:

AutoIdleDecommit(Recycler \* recycler) : recycler(recycler) { recycler->EnterIdleDecommit(); }

~AutoIdleDecommit() { recycler->LeaveIdleDecommit(); }

private:

Recycler \* recycler;

};

template <typename SmallHeapBlockAllocatorType>

void

Recycler::AddSmallAllocator(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat)

{

autoHeap.AddSmallAllocator(allocator, sizeCat);

}

template <typename SmallHeapBlockAllocatorType>

void

Recycler::RemoveSmallAllocator(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat)

{

autoHeap.RemoveSmallAllocator(allocator, sizeCat);

}

template <ObjectInfoBits attributes, typename SmallHeapBlockAllocatorType>

char \*

Recycler::SmallAllocatorAlloc(SmallHeapBlockAllocatorType \* allocator, size\_t sizeCat)

{

return autoHeap.SmallAllocatorAlloc<attributes>(this, allocator, sizeCat);

}

// Dummy recycler allocator policy classes to choose the allocation function

class \_RecyclerLeafPolicy;

class \_RecyclerNonLeafPolicy;

template <typename Policy>

class \_RecyclerAllocatorFunc

{};

template <>

class \_RecyclerAllocatorFunc<\_RecyclerLeafPolicy>

{

public:

typedef char \* (Recycler::\*AllocFuncType)(size\_t);

typedef bool (Recycler::\*FreeFuncType)(void\*, size\_t);

static AllocFuncType GetAllocFunc()

{

return &Recycler::AllocLeaf;

}

static AllocFuncType GetAllocZeroFunc()

{

return &Recycler::AllocLeafZero;

}

static FreeFuncType GetFreeFunc()

{

return &Recycler::ExplicitFreeLeaf;

}

};

template <>

class \_RecyclerAllocatorFunc<\_RecyclerNonLeafPolicy>

{

public:

typedef char \* (Recycler::\*AllocFuncType)(size\_t);

typedef bool (Recycler::\*FreeFuncType)(void\*, size\_t);

static AllocFuncType GetAllocFunc()

{

return &Recycler::Alloc;

}

static AllocFuncType GetAllocZeroFunc()

{

return &Recycler::AllocZero;

}

static FreeFuncType GetFreeFunc()

{

return &Recycler::ExplicitFreeNonLeaf;

}

};

// This is used by the compiler; when T is NOT a pointer i.e. a value type - it causes leaf allocation

template <typename T>

class TypeAllocatorFunc<Recycler, T> : public \_RecyclerAllocatorFunc<\_RecyclerLeafPolicy>

{

};

// Partial template specialization; applies to T when it is a pointer

template <typename T>

class TypeAllocatorFunc<Recycler, T \*> : public \_RecyclerAllocatorFunc<\_RecyclerNonLeafPolicy>

{

};

template <bool isLeaf>

class ListTypeAllocatorFunc<Recycler, isLeaf>

{

public:

typedef bool (Recycler::\*FreeFuncType)(void\*, size\_t);

static FreeFuncType GetFreeFunc()

{

if (isLeaf)

{

return &Recycler::ExplicitFreeLeaf;

}

else

{

return &Recycler::ExplicitFreeNonLeaf;

}

}

};

// Dummy class to choose the allocation function

class RecyclerLeafAllocator;

class RecyclerNonLeafAllocator;

// Partial template specialization to allocate as non leaf

template <typename T>

class TypeAllocatorFunc<RecyclerNonLeafAllocator, T> : public \_RecyclerAllocatorFunc<\_RecyclerNonLeafPolicy>

{

};

template <typename T>

class TypeAllocatorFunc<RecyclerLeafAllocator, T> : public \_RecyclerAllocatorFunc<\_RecyclerLeafPolicy>

{

};

template <typename TAllocType>

struct AllocatorInfo<Recycler, TAllocType>

{

typedef Recycler AllocatorType;

typedef TypeAllocatorFunc<Recycler, TAllocType> AllocatorFunc;

typedef \_RecyclerAllocatorFunc<\_RecyclerNonLeafPolicy> InstAllocatorFunc; // By default any instance considered non-leaf

};

template <typename TAllocType>

struct AllocatorInfo<RecyclerNonLeafAllocator, TAllocType>

{

typedef Recycler AllocatorType;

typedef TypeAllocatorFunc<RecyclerNonLeafAllocator, TAllocType> AllocatorFunc;

typedef TypeAllocatorFunc<RecyclerNonLeafAllocator, TAllocType> InstAllocatorFunc; // Same as TypeAllocatorFunc

};

template <typename TAllocType>

struct AllocatorInfo<RecyclerLeafAllocator, TAllocType>

{

typedef Recycler AllocatorType;

typedef TypeAllocatorFunc<RecyclerLeafAllocator, TAllocType> AllocatorFunc;

typedef TypeAllocatorFunc<RecyclerLeafAllocator, TAllocType> InstAllocatorFunc; // Same as TypeAllocatorFunc

};

template <>

struct ForceNonLeafAllocator<Recycler>

{

typedef RecyclerNonLeafAllocator AllocatorType;

};

template <>

struct ForceNonLeafAllocator<RecyclerLeafAllocator>

{

typedef RecyclerNonLeafAllocator AllocatorType;

};

template <>

struct ForceLeafAllocator<Recycler>

{

typedef RecyclerLeafAllocator AllocatorType;

};

template <>

struct ForceLeafAllocator<RecyclerNonLeafAllocator>

{

typedef RecyclerLeafAllocator AllocatorType;

};

#ifdef PROFILE\_EXEC

#define RECYCLER\_PROFILE\_EXEC\_BEGIN(recycler, phase) if (recycler->profiler != nullptr) { recycler->profiler->Begin(phase); }

#define RECYCLER\_PROFILE\_EXEC\_END(recycler, phase) if (recycler->profiler != nullptr) { recycler->profiler->End(phase); }

#define RECYCLER\_PROFILE\_EXEC\_BEGIN2(recycler, phase1, phase2) if (recycler->profiler != nullptr) { recycler->profiler->Begin(phase1); recycler->profiler->Begin(phase2);}

#define RECYCLER\_PROFILE\_EXEC\_END2(recycler, phase1, phase2) if (recycler->profiler != nullptr) { recycler->profiler->End(phase1); recycler->profiler->End(phase2);}

#define RECYCLER\_PROFILE\_EXEC\_CHANGE(recydler, phase1, phase2) if (recycler->profiler != nullptr) { recycler->profiler->End(phase1); recycler->profiler->Begin(phase2); }

#define RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(recycler, phase) if (recycler->backgroundProfiler != nullptr) { recycler->backgroundProfiler->Begin(phase); }

#define RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(recycler, phase) if (recycler->backgroundProfiler != nullptr) { recycler->backgroundProfiler->End(phase); }

#define RECYCLER\_PROFILE\_EXEC\_THREAD\_BEGIN(background, recycler, phase) if (background) { RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(recycler, phase); } else { RECYCLER\_PROFILE\_EXEC\_BEGIN(recycler, phase); }

#define RECYCLER\_PROFILE\_EXEC\_THREAD\_END(background, recycler, phase) if (background) { RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(recycler, phase); } else { RECYCLER\_PROFILE\_EXEC\_END(recycler, phase); }

#else

#define RECYCLER\_PROFILE\_EXEC\_BEGIN(recycler, phase)

#define RECYCLER\_PROFILE\_EXEC\_END(recycler, phase)

#define RECYCLER\_PROFILE\_EXEC\_BEGIN2(recycler, phase1, phase2)

#define RECYCLER\_PROFILE\_EXEC\_END2(recycler, phase1, phase2)

#define RECYCLER\_PROFILE\_EXEC\_CHANGE(recydler, phase1, phase2)

#define RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_BEGIN(recycler, phase)

#define RECYCLER\_PROFILE\_EXEC\_BACKGROUND\_END(recycler, phase)

#define RECYCLER\_PROFILE\_EXEC\_THREAD\_BEGIN(background, recycler, phase)

#define RECYCLER\_PROFILE\_EXEC\_THREAD\_END(background, recycler, phase)

#endif

}

\_Ret\_notnull\_ inline void \* \_\_cdecl

operator new(size\_t byteSize, Recycler \* alloc, HeapInfo \* heapInfo)

{

return alloc->HeapAllocR(heapInfo, byteSize);

}

inline void \_\_cdecl

operator delete(void \* obj, Recycler \* alloc, HeapInfo \* heapInfo)

{

alloc->HeapFree(heapInfo, obj);

}

\_Ret\_notnull\_ inline void \* \_\_cdecl

operator new(size\_t byteSize, Recycler \* recycler, ObjectInfoBits enumClassBits)

{

AssertCanHandleOutOfMemory();

Assert(byteSize != 0);

Assert(enumClassBits == EnumClass\_1\_Bit);

void \* buffer = recycler->AllocEnumClass<EnumClass\_1\_Bit>(byteSize);

// All of our allocation should throw on out of memory

Assume(buffer != nullptr);

return buffer;

}

template<ObjectInfoBits infoBits>

\_Ret\_notnull\_ inline void \* \_\_cdecl

operator new(size\_t byteSize, Recycler \* recycler, const InfoBitsWrapper<infoBits>&)

{

AssertCanHandleOutOfMemory();

Assert(byteSize != 0);

void \* buffer = recycler->AllocWithInfoBits<infoBits>(byteSize);

// All of our allocation should throw on out of memory

Assume(buffer != nullptr);

return buffer;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

template <ObjectInfoBits attributes>

bool

Recycler::IntegrateBlock(char \* blockAddress, PageSegment \* segment, size\_t allocSize, size\_t objectSize)

{

// We only support no bit and leaf bit right now, where we don't need to set the object info in either case

CompileAssert(attributes == NoBit || attributes == LeafBit);

// Currently only small object is supported

Assert(HeapInfo::IsSmallObject(allocSize));

Assert(HeapInfo::GetAlignedSizeNoCheck(allocSize) == allocSize);

// This should only happen during a pre-collection callback.

Assert(this->collectionState == Collection\_PreCollection);

bool success = autoHeap.IntegrateBlock<attributes>(blockAddress, segment, this, allocSize);

#ifdef PROFILE\_RECYCLER\_ALLOC

if (success)

{

TrackAllocData trackAllocData;

ClearTrackAllocInfo(&trackAllocData);

TrackIntegrate(blockAddress, SmallAllocationBlockAttributes::PageCount \* AutoSystemInfo::PageSize, allocSize, objectSize, trackAllocData);

}

#endif

return success;

}

namespace Memory

{

class DummyVTableObject : public FinalizableObject

{

public:

virtual void Finalize(bool isShutdown) {}

virtual void Dispose(bool isShutdown) {}

virtual void Mark(Recycler \* recycler) {}

};

}

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char \*

Recycler::AllocWithAttributesInlined(size\_t size)

{

// All tracked objects are client tracked objects

CompileAssert((attributes & TrackBit) == 0 || (attributes & ClientTrackedBit) != 0);

Assert(this->enableScanImplicitRoots || (attributes & ImplicitRootBit) == 0);

AssertMsg(this->disableThreadAccessCheck || this->mainThreadId == GetCurrentThreadContextId(),

"Allocating from the recycler can only be done on the main thread");

Assert(size != 0);

AssertMsg(collectionState != Collection\_PreCollection, "we cannot have allocation in precollection callback");

// We shouldn't be allocating memory when we are running GC in thread, including finalizers

Assert(this->IsConcurrentState() || !this->CollectionInProgress() || this->collectionState == CollectionStatePostCollectionCallback);

// There are some cases where we allow allocation during heap enum that doesn't affect the enumeration

// Those should be really rare and not rely upon.

Assert(!isHeapEnumInProgress || allowAllocationDuringHeapEnum);

#ifdef PROFILE\_RECYCLER\_ALLOC

TrackAllocData trackAllocData;

ClearTrackAllocInfo(&trackAllocData);

#endif

size\_t allocSize = size;

#ifdef RECYCLER\_MEMORY\_VERIFY

if (this->VerifyEnabled())

{

allocSize += verifyPad + sizeof(size\_t);

if (allocSize < size)

{

// An overflow occurred- if nothrow is false, we can throw here

// Otherwise, return null

if (nothrow == false)

{

this->OutOfMemory();

}

else

{

return nullptr;

}

}

}

#endif

char\* memBlock = RealAlloc<(ObjectInfoBits)(attributes & InternalObjectInfoBitMask), nothrow>(&autoHeap, allocSize);

if (nothrow)

{

// If we aren't allowed to throw, then the memblock returned could be null

// so we should check for that and bail out early here

if (memBlock == nullptr)

{

return nullptr;

}

}

#ifdef PROFILE\_RECYCLER\_ALLOC

TrackAlloc(memBlock, size, trackAllocData, (CUSTOM\_CONFIG\_ISENABLED(GetRecyclerFlagsTable(), Js::TraceObjectAllocationFlag) && (attributes & TraceBit) == TraceBit));

#endif

RecyclerMemoryTracking::ReportAllocation(this, memBlock, size);

RECYCLER\_PERF\_COUNTER\_INC(LiveObject);

RECYCLER\_PERF\_COUNTER\_ADD(LiveObjectSize, HeapInfo::GetAlignedSizeNoCheck(allocSize));

RECYCLER\_PERF\_COUNTER\_SUB(FreeObjectSize, HeapInfo::GetAlignedSizeNoCheck(allocSize));

if (HeapInfo::IsSmallBlockAllocation(HeapInfo::GetAlignedSizeNoCheck(allocSize)))

{

RECYCLER\_PERF\_COUNTER\_INC(SmallHeapBlockLiveObject);

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockLiveObjectSize, HeapInfo::GetAlignedSizeNoCheck(allocSize));

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockFreeObjectSize, HeapInfo::GetAlignedSizeNoCheck(allocSize));

}

else

{

RECYCLER\_PERF\_COUNTER\_INC(LargeHeapBlockLiveObject);

RECYCLER\_PERF\_COUNTER\_ADD(LargeHeapBlockLiveObjectSize, HeapInfo::GetAlignedSizeNoCheck(allocSize));

RECYCLER\_PERF\_COUNTER\_SUB(LargeHeapBlockFreeObjectSize, HeapInfo::GetAlignedSizeNoCheck(allocSize));

}

#ifdef RECYCLER\_MEMORY\_VERIFY

size\_t alignedSize = HeapInfo::GetAlignedSizeNoCheck(allocSize);

if (HeapInfo::IsMediumObject(allocSize))

{

#if SMALLBLOCK\_MEDIUM\_ALLOC

alignedSize = HeapInfo::GetMediumObjectAlignedSizeNoCheck(allocSize);

#else

HeapBlock\* heapBlock = this->FindHeapBlock(memBlock);

Assert(heapBlock->IsLargeHeapBlock());

LargeHeapBlock\* largeHeapBlock = (LargeHeapBlock\*) heapBlock;

LargeObjectHeader\* header = nullptr;

if (largeHeapBlock->GetObjectHeader(memBlock, &header))

{

size = header->objectSize - (verifyPad + sizeof(size\_t));

alignedSize = HeapInfo::GetAlignedSizeNoCheck(header->objectSize);

}

#endif

}

this->FillCheckPad(memBlock, size, alignedSize);

#endif

#pragma prefast(suppress:6313, "attributes is a template parameter and can be 0")

if (attributes & (FinalizeBit | TrackBit))

{

// Make sure a valid vtable is installed in case of OOM before the real vtable is set

memBlock = (char \*)new (memBlock) DummyVTableObject();

}

#ifdef RECYCLER\_WRITE\_BARRIER

SwbVerboseTrace(this->GetRecyclerFlagsTable(), L"Allocated SWB memory: 0x%p\n", memBlock);

#pragma prefast(suppress:6313, "attributes is a template parameter and can be 0")

if (attributes & (NewTrackBit))

{

// For objects allocated with NewTrackBit, we need to trigger the write barrier since

// there could be a GC triggered by an allocation in the constructor, and we'd miss

// calling track on the partially constructed object. To deal with this, we set the write

// barrier on all the pages of objects allocated with the NewTrackBit

RecyclerWriteBarrierManager::WriteBarrier(memBlock, size / sizeof(void\*));

}

#endif

#ifdef PARTIAL\_GC\_ENABLED

#pragma prefast(suppress:6313, "attributes is a template parameter and can be 0")

if (attributes & ClientTrackedBit)

{

if (this->inPartialCollectMode)

{

// with partial GC, we don't traverse ITrackable

// So we have to mark all objects that could be in the ITrackable graph

// This includes JavascriptDispatch and HostVariant

this->clientTrackedObjectList.Prepend(&this->clientTrackedObjectAllocator, memBlock);

}

else

{

Assert(this->hasBackgroundFinishPartial || this->clientTrackedObjectList.Empty());

}

}

#endif

return memBlock;

}

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char \*

Recycler::AllocZeroWithAttributesInlined(size\_t size)

{

char\* obj = AllocWithAttributesInlined<attributes, nothrow>(size);

if (nothrow)

{

// If we aren't allowed to throw, then the obj returned could be null

// so we should check for that and bail out early here

if (obj == nullptr)

{

return nullptr;

}

}

#ifdef RECYCLER\_MEMORY\_VERIFY

if (this->VerifyEnabled())

{

memset(obj, 0, min(size, sizeof(FreeObject \*)));

}

else

#endif

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_THREAD\_PAGE

if ((((attributes & LeafBit) == LeafBit) || ((attributes & WithBarrierBit) == WithBarrierBit)) && HeapInfo::IsSmallObjectAllocation(size))

#else

if (((attributes & LeafBit) == LeafBit) && HeapInfo::IsSmallBlockAllocation(size))

#endif

{

// If this was allocated from the small heap block, it's not

// guaranteed to be zero so we should zero out here.

memset((void\*) obj, 0, size);

}

else

{

// All recycler memory are allocated with zero except for the first word,

// which store the next pointer for the free list. Just zero that one out

((FreeObject \*)obj)->ZeroNext();

}

return obj;

}

template <ObjectInfoBits attributes, bool isSmallAlloc, bool nothrow>

\_\_inline char\*

Recycler::RealAllocFromBucket(HeapInfo\* heap, size\_t size)

{

// Align the size

Assert(HeapInfo::GetAlignedSizeNoCheck(size) <= UINT\_MAX);

uint sizeCat;

char \* memBlock;

if (isSmallAlloc)

{

sizeCat = (uint)HeapInfo::GetAlignedSizeNoCheck(size);

memBlock = heap->RealAlloc<attributes, nothrow>(this, sizeCat);

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

else

{

sizeCat = (uint)HeapInfo::GetMediumObjectAlignedSizeNoCheck(size);

memBlock = heap->MediumAlloc<attributes, nothrow>(this, sizeCat);

}

#endif

// If we are not allowed to throw, then the memory returned here could be null so check for that

// If we are allowed to throw, then memBlock is not allowed to null so assert that

if (nothrow)

{

if (memBlock == nullptr)

{

return nullptr;

}

}

else

{

Assert(memBlock != nullptr);

}

#ifdef RECYCLER\_ZERO\_MEM\_CHECK

// Don't bother checking leaf allocations for zeroing out- they're not guaranteed to be so

if ((attributes & LeafBit) == 0

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC\_THREAD\_PAGE

&& (attributes & WithBarrierBit) == 0

#endif

)

{

VerifyZeroFill(memBlock + sizeof(FreeObject), sizeCat - (2 \* sizeof(FreeObject)));

}

#endif

#ifdef PROFILE\_MEM

if (this->memoryData)

{

this->memoryData->requestCount++;

this->memoryData->requestBytes += size;

this->memoryData->alignmentBytes += sizeCat - size;

}

#endif

return memBlock;

}

template <ObjectInfoBits attributes, bool nothrow>

\_\_inline char\*

Recycler::RealAlloc(HeapInfo\* heap, size\_t size)

{

#ifdef RECYCLER\_STRESS

this->StressCollectNow();

#endif

if (nothrow)

{

FAULTINJECT\_MEMORY\_NOTHROW(L"Recycler", size);

}

else

{

FAULTINJECT\_MEMORY\_THROW(L"Recycler", size);

}

if (HeapInfo::IsSmallObject(size))

{

return RealAllocFromBucket<attributes, /\* isSmallAlloc = \*/ true, nothrow>(heap, size);

}

#ifdef BUCKETIZE\_MEDIUM\_ALLOCATIONS

if (HeapInfo::IsMediumObject(size))

{

return RealAllocFromBucket<attributes, /\* isSmallAlloc = \*/ false, nothrow>(heap, size);

}

#endif

return LargeAlloc<nothrow>(heap, size, attributes);

}

template<typename T>

\_\_inline RecyclerWeakReference<T>\* Recycler::CreateWeakReferenceHandle(T\* pStrongReference)

{

// Return the weak reference that calling Add on the WR map returns

// The entry returned is recycler-allocated memory

RecyclerWeakReference<T>\* weakRef = (RecyclerWeakReference<T>\*) this->weakReferenceMap.Add((char\*) pStrongReference, this);

#if DBG

if (weakRef->typeInfo == nullptr)

{

weakRef->typeInfo = &typeid(T);

#ifdef TRACK\_ALLOC

TrackAllocWeakRef(weakRef);

#endif

}

#endif

return weakRef;

}

template<typename T>

\_\_inline bool Recycler::FindOrCreateWeakReferenceHandle(T\* pStrongReference, RecyclerWeakReference<T> \*\*ppWeakRef)

{

// Ensure that the given strong ref has a weak ref in the map.

// Return a result to indicate whether a new weak ref was created.

bool ret = this->weakReferenceMap.FindOrAdd((char\*) pStrongReference, this, (RecyclerWeakReferenceBase\*\*)ppWeakRef);

#if DBG

if (!ret)

{

(\*ppWeakRef)->typeInfo = &typeid(T);

#ifdef TRACK\_ALLOC

TrackAllocWeakRef(\*ppWeakRef);

#endif

}

#endif

return ret;

}

template<typename T>

\_\_inline bool Recycler::TryGetWeakReferenceHandle(T\* pStrongReference, RecyclerWeakReference<T> \*\*weakReference)

{

return this->weakReferenceMap.TryGetValue((char\*) pStrongReference, (RecyclerWeakReferenceBase\*\*)weakReference);

}

\_\_inline HeapBlock\*

Recycler::FindHeapBlock(void\* candidate)

{

if ((size\_t)candidate < 0x10000)

{

return nullptr;

}

if (!HeapInfo::IsAlignedAddress(candidate))

{

return nullptr;

}

return heapBlockMap.GetHeapBlock(candidate);

}

\_\_inline void

Recycler::ScanObjectInline(void \*\* obj, size\_t byteCount)

{

// This is never called during parallel marking

Assert(this->collectionState != CollectionStateParallelMark);

if (this->enableScanInteriorPointers)

{

ScanObjectInlineInterior(obj, byteCount);

}

else

{

markContext.ScanObject<false, false>(obj, byteCount);

}

}

\_\_inline void

Recycler::ScanObjectInlineInterior(void \*\* obj, size\_t byteCount)

{

// This is never called during parallel marking

Assert(this->collectionState != CollectionStateParallelMark);

Assert(this->enableScanInteriorPointers);

markContext.ScanObject<false, true>(obj, byteCount);

}

\_\_inline void

Recycler::ScanMemoryInline(void \*\* obj, size\_t byteCount)

{

// This is never called during parallel marking

Assert(this->collectionState != CollectionStateParallelMark);

if (this->enableScanInteriorPointers)

{

markContext.ScanMemory<false, true>(obj, byteCount);

}

else

{

markContext.ScanMemory<false, false>(obj, byteCount);

}

}

\_\_inline bool

Recycler::AddMark(void \* candidate, size\_t byteCount) throw()

{

// This is never called during parallel marking

Assert(this->collectionState != CollectionStateParallelMark);

return markContext.AddMarkedObject(candidate, byteCount);

}

template <bool pageheap, typename T>

void

Recycler::NotifyFree(T \* heapBlock)

{

bool forceSweepObject = ForceSweepObject();

if (forceSweepObject)

{

#if DBG || defined(RECYCLER\_STATS)

this->isForceSweeping = true;

heapBlock->isForceSweeping = true;

#endif

heapBlock->SweepObjects<pageheap, SweepMode\_InThread>(this);

#if DBG || defined(RECYCLER\_STATS)

heapBlock->isForceSweeping = false;

this->isForceSweeping = false;

#endif

RECYCLER\_STATS\_INC(this, heapBlockFreeCount[heapBlock->GetHeapBlockType()]);

}

JS\_ETW(EventWriteFreeMemoryBlock(heapBlock));

#ifdef RECYCLER\_PERF\_COUNTERS

if (forceSweepObject)

{

RECYCLER\_PERF\_COUNTER\_SUB(FreeObjectSize, heapBlock->GetPageCount() \* AutoSystemInfo::PageSize);

if (heapBlock->IsLargeHeapBlock())

{

RECYCLER\_PERF\_COUNTER\_SUB(LargeHeapBlockFreeObjectSize, heapBlock->GetPageCount() \* AutoSystemInfo::PageSize);

}

else

{

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockFreeObjectSize, heapBlock->GetPageCount() \* AutoSystemInfo::PageSize);

}

}

else

{

heapBlock->UpdatePerfCountersOnFree();

}

#endif

}

template <class TBlockAttributes>

\_\_inline ushort

SmallHeapBlockT<TBlockAttributes>::GetObjectBitDelta()

{

return this->objectSize / HeapConstants::ObjectGranularity;

}

// Map any object address to it's bit index in the heap block bit vectors.

// static

template <class TBlockAttributes>

\_\_forceinline ushort

SmallHeapBlockT<TBlockAttributes>::GetAddressBitIndex(void \* objectAddress)

{

Assert(HeapInfo::IsAlignedAddress(objectAddress));

ushort offset = (ushort)(::Math::PointerCastToIntegralTruncate<uint>(objectAddress) % (TBlockAttributes::PageCount \* AutoSystemInfo::PageSize));

offset = offset >> HeapConstants::ObjectAllocationShift;

Assert(offset <= USHRT\_MAX);

Assert(offset <= TBlockAttributes::MaxAddressBit);

return (ushort) offset;

}

template <class TBlockAttributes>

\_\_forceinline ushort

SmallHeapBlockT<TBlockAttributes>::GetObjectIndexFromBitIndex(ushort bitIndex)

{

Assert(bitIndex <= TBlockAttributes::MaxAddressBit);

ushort objectIndex = validPointers.GetAddressIndex(bitIndex);

Assert(objectIndex == SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit ||

objectIndex <= TBlockAttributes::MaxAddressBit);

return objectIndex;

}

template <class TBlockAttributes>

\_\_forceinline void \*

SmallHeapBlockT<TBlockAttributes>::GetRealAddressFromInterior(void \* interiorAddress, uint objectSize, byte bucketIndex)

{

const ValidPointers<TBlockAttributes> validPointers = HeapInfo::GetValidPointersMapForBucket<TBlockAttributes>(bucketIndex);

size\_t rawInteriorAddress = reinterpret\_cast<size\_t>(interiorAddress);

size\_t baseAddress = rawInteriorAddress & ~(TBlockAttributes::PageCount \* AutoSystemInfo::PageSize - 1);

ushort offset = (ushort)(rawInteriorAddress - baseAddress);

offset = validPointers.GetInteriorAddressIndex(offset >> HeapConstants::ObjectAllocationShift);

if (offset == SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit)

{

return nullptr;

}

return reinterpret\_cast<void\*>(baseAddress + offset \* objectSize);

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

template <typename T, ObjectInfoBits attributes>

class RecyclerFastAllocator

{

typedef typename SmallHeapBlockType<(ObjectInfoBits)(attributes & GetBlockTypeBitMask), SmallAllocationBlockAttributes>::BlockType BlockType;

public:

#ifdef TRACK\_ALLOC

RecyclerFastAllocator \* TrackAllocInfo(TrackAllocData const& data)

{

#ifdef PROFILE\_RECYCLER\_ALLOC

recycler->TrackAllocInfo(data);

#endif

return this;

}

#endif

void Initialize(Recycler \* recycler)

{

this->recycler = recycler;

size\_t sizeCat = GetAlignedAllocSize();

recycler->AddSmallAllocator(&allocator, sizeCat);

}

void Uninitialize()

{

size\_t sizeCat = GetAlignedAllocSize();

this->recycler->RemoveSmallAllocator(&allocator, sizeCat);

this->recycler = nullptr;

}

Recycler \* GetRecycler() { return recycler; }

char \* Alloc(size\_t size)

{

Assert(recycler != nullptr);

Assert(!recycler->IsHeapEnumInProgress() || recycler->AllowAllocationDuringHeapEnum());

Assert(size == sizeof(T));

#ifdef PROFILE\_RECYCLER\_ALLOC

TrackAllocData trackAllocData;

recycler->ClearTrackAllocInfo(&trackAllocData);

#endif

size\_t sizeCat = GetAlignedAllocSize();

Assert(HeapInfo::IsSmallObject(sizeCat));

char \* memBlock = allocator.InlinedAlloc<(ObjectInfoBits)(attributes & InternalObjectInfoBitMask)>(recycler, sizeCat);

if (memBlock == nullptr)

{

memBlock = recycler->SmallAllocatorAlloc<attributes>(&allocator, sizeCat);

Assert(memBlock != nullptr);

}

#ifdef PROFILE\_RECYCLER\_ALLOC

recycler->TrackAlloc(memBlock, sizeof(T), trackAllocData);

#endif

RecyclerMemoryTracking::ReportAllocation(this->recycler, memBlock, sizeof(T));

RECYCLER\_PERF\_COUNTER\_INC(LiveObject);

RECYCLER\_PERF\_COUNTER\_ADD(LiveObjectSize, sizeCat);

RECYCLER\_PERF\_COUNTER\_SUB(FreeObjectSize, sizeCat);

RECYCLER\_PERF\_COUNTER\_INC(SmallHeapBlockLiveObject);

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockLiveObjectSize, sizeCat);

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockFreeObjectSize, sizeCat);

#ifdef RECYCLER\_MEMORY\_VERIFY

recycler->FillCheckPad(memBlock, sizeof(T), sizeCat);

#endif

return memBlock;

};

static uint32 GetEndAddressOffset()

{

return offsetof(RecyclerFastAllocator, allocator) + SmallHeapBlockAllocator<BlockType>::GetEndAddressOffset();

}

bool AllowNativeCodeBumpAllocation()

{

return recycler->AllowNativeCodeBumpAllocation();

}

char \*GetEndAddress()

{

return allocator.GetEndAddress();

}

static uint32 GetFreeObjectListOffset()

{

return offsetof(RecyclerFastAllocator, allocator) + SmallHeapBlockAllocator<BlockType>::GetFreeObjectListOffset();

}

FreeObject \*GetFreeObjectList()

{

return allocator.GetFreeObjectList();

}

void SetFreeObjectList(FreeObject \*freeObject)

{

allocator.SetFreeObjectList(freeObject);

}

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(RECYCLER\_MEMORY\_VERIFY) || defined(MEMSPECT\_TRACKING) || defined(ETW\_MEMORY\_TRACKING)

RecyclerFastAllocator()

{

allocator.SetTrackNativeAllocatedObjectCallBack(&TrackNativeAllocatedObject);

}

static void TrackNativeAllocatedObject(Recycler \* recycler, void \* memBlock, size\_t sizeCat)

{

#ifdef PROFILE\_RECYCLER\_ALLOC

TrackAllocData trackAllocData = { &typeid(T), 0, (size\_t)-1, NULL, 0 };

recycler->TrackAlloc(memBlock, sizeof(T), trackAllocData);

#endif

RecyclerMemoryTracking::ReportAllocation(recycler, memBlock, sizeof(T));

RECYCLER\_PERF\_COUNTER\_INC(LiveObject);

RECYCLER\_PERF\_COUNTER\_ADD(LiveObjectSize, sizeCat);

RECYCLER\_PERF\_COUNTER\_SUB(FreeObjectSize, sizeCat);

RECYCLER\_PERF\_COUNTER\_INC(SmallHeapBlockLiveObject);

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockLiveObjectSize, sizeCat);

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockFreeObjectSize, sizeCat);

#ifdef RECYCLER\_MEMORY\_VERIFY

recycler->FillCheckPad(memBlock, sizeof(T), sizeCat, true);

#endif

}

#endif

size\_t GetAlignedAllocSize() const

{

#ifdef RECYCLER\_MEMORY\_VERIFY

if (recycler->VerifyEnabled())

{

CompileAssert(sizeof(T) <= (size\_t)-1 - sizeof(size\_t));

return HeapInfo::GetAlignedSize(AllocSizeMath::Add(sizeof(T) + sizeof(size\_t), recycler->verifyPad));

}

#endif

// We should have structures large enough that would cause this to overflow

CompileAssert(((sizeof(T) + (HeapConstants::ObjectGranularity-1)) & ~(HeapConstants::ObjectGranularity-1)) != 0);

return HeapInfo::GetAlignedSizeNoCheck(sizeof(T));

}

private:

SmallHeapBlockAllocator<BlockType> allocator;

Recycler \* recycler;

CompileAssert(sizeof(T) <= HeapConstants::MaxSmallObjectSize);

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAT")

RecyclerHeuristic RecyclerHeuristic::Instance;

// static

RecyclerHeuristic::RecyclerHeuristic()

{

::MEMORYSTATUSEX mem;

mem.dwLength = sizeof(mem);

BOOL isSuccess = ::GlobalMemoryStatusEx(&mem);

Assert(isSuccess);

DWORDLONG physicalMemoryBytes = mem.ullTotalPhys;

uint baseFactor;

if (isSuccess && AutoSystemInfo::IsLowMemoryDevice() && physicalMemoryBytes <= 512 MEGABYTES)

{

// Low-end Apollo (512MB RAM) scenario.

// Note that what's specific about Apollo is that IE runs in physical memory,

// that's one reason to distinguish 512MB Apollo from 512MB desktop.

baseFactor = 16;

this->DefaultMaxFreePageCount = 16 MEGABYTES\_OF\_PAGES;

this->DefaultMaxAllocPageCount = 32;

}

else if (isSuccess && physicalMemoryBytes <= 1024 MEGABYTES)

{

// Tablet/slate/high-end Apollo scenario, including 512MB non-Apollo.

baseFactor = 64;

this->DefaultMaxFreePageCount = 64 MEGABYTES\_OF\_PAGES;

this->DefaultMaxAllocPageCount = 64;

}

else

{

// Regular desktop scenario.

baseFactor = 192;

this->DefaultMaxFreePageCount = 512 MEGABYTES\_OF\_PAGES;

this->DefaultMaxAllocPageCount = 256;

}

this->ConfigureBaseFactor(baseFactor);

}

void

RecyclerHeuristic::ConfigureBaseFactor(uint baseFactor)

{

this->MaxUncollectedAllocBytes = baseFactor MEGABYTES;

this->UncollectedAllocBytesConcurrentPriorityBoost = baseFactor MEGABYTES;

this->MaxPartialUncollectedNewPageCount = baseFactor MEGABYTES\_OF\_PAGES;

this->MaxUncollectedAllocBytesOnExit = (baseFactor / 2) MEGABYTES;

this->MaxUncollectedAllocBytesPartialCollect = this->MaxUncollectedAllocBytes - 1 MEGABYTES;

}

uint

RecyclerHeuristic::UncollectedAllocBytesCollection()

{

return DefaultUncollectedAllocBytesCollection;

}

#ifdef CONCURRENT\_GC\_ENABLED

uint

RecyclerHeuristic::MaxBackgroundFinishMarkCount(Js::ConfigFlagsTable& flags)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (flags.IsEnabled(Js::MaxBackgroundFinishMarkCountFlag))

{

return flags.MaxBackgroundFinishMarkCount;

}

#endif

return DefaultMaxBackgroundFinishMarkCount;

}

DWORD

RecyclerHeuristic::BackgroundFinishMarkWaitTime(bool backgroundFinishMarkWaitTime, Js::ConfigFlagsTable& flags)

{

if (RECYCLER\_HEURISTIC\_VERSION == 10)

{

backgroundFinishMarkWaitTime = backgroundFinishMarkWaitTime && CUSTOM\_PHASE\_ON1(flags, Js::BackgroundFinishMarkPhase);

}

else

{

backgroundFinishMarkWaitTime = backgroundFinishMarkWaitTime && !CUSTOM\_PHASE\_OFF1(flags, Js::BackgroundFinishMarkPhase);

}

if (!backgroundFinishMarkWaitTime && !CUSTOM\_PHASE\_FORCE1(flags, Js::BackgroundFinishMarkPhase))

{

return INFINITE;

}

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (flags.IsEnabled(Js::BackgroundFinishMarkWaitTimeFlag))

{

return flags.BackgroundFinishMarkWaitTime;

}

#endif

if (CUSTOM\_PHASE\_FORCE1(flags, Js::BackgroundFinishMarkPhase))

{

return INFINITE;

}

return DefaultBackgroundFinishMarkWaitTime;

}

size\_t

RecyclerHeuristic::MinBackgroundRepeatMarkRescanBytes(Js::ConfigFlagsTable& flags)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (flags.IsEnabled(Js::MinBackgroundRepeatMarkRescanBytesFlag))

{

return flags.MinBackgroundRepeatMarkRescanBytes;

}

#endif

return DefaultMinBackgroundRepeatMarkRescanBytes;

}

DWORD

RecyclerHeuristic::FinishConcurrentCollectWaitTime(Js::ConfigFlagsTable& flags)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (flags.IsEnabled(Js::RecyclerThreadCollectTimeoutFlag))

{

return flags.RecyclerThreadCollectTimeout;

}

#endif

return DefaultFinishConcurrentCollectWaitTime;

}

DWORD

RecyclerHeuristic::PriorityBoostTimeout(Js::ConfigFlagsTable& flags)

{

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

if (flags.IsEnabled(Js::RecyclerPriorityBoostTimeoutFlag))

{

return flags.RecyclerPriorityBoostTimeout;

}

#endif

return TickCountConcurrentPriorityBoost;

}

#endif

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

bool

RecyclerHeuristic::PartialConcurrentNextCollection(double ratio, Js::ConfigFlagsTable& flags)

{

if (CUSTOM\_PHASE\_FORCE1(flags, Js::ConcurrentPartialCollectPhase))

{

return true;

}

if (RECYCLER\_HEURISTIC\_VERSION == 10)

{

// Default off for version == 10

if (!CUSTOM\_PHASE\_ON1(flags, Js::ConcurrentPartialCollectPhase))

{

return false;

}

}

else

{

// Default on

if (CUSTOM\_PHASE\_OFF1(flags, Js::ConcurrentPartialCollectPhase))

{

return false;

}

}

return ratio >= 0.5;

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

#define KILOBYTES \* 1024

#define MEGABYTES \* 1024 KILOBYTES

#define MEGABYTES\_OF\_PAGES \* 1024 \* 1024 / AutoSystemInfo::PageSize;

class RecyclerHeuristic

{

private:

RecyclerHeuristic();

public:

static RecyclerHeuristic Instance;

// Heuristics that depend on hardware or environment (not constant).

uint MaxUncollectedAllocBytes;

size\_t UncollectedAllocBytesConcurrentPriorityBoost;

uint MaxPartialUncollectedNewPageCount;

uint MaxUncollectedAllocBytesOnExit;

// If we are getting close to the full GC limit, let's just get out of partial GC mode.

uint MaxUncollectedAllocBytesPartialCollect;

// Defines the PageSegment size for recycler small block page allocator.

uint DefaultMaxAllocPageCount;

// Used for RecyclerPageAllocator and determines how many free pages need to be there to trigger decommit

// (most cases, not all).

uint DefaultMaxFreePageCount;

// Constant heuristic that may be changed by switches

static uint UncollectedAllocBytesCollection();

#ifdef CONCURRENT\_GC\_ENABLED

static uint MaxBackgroundFinishMarkCount(Js::ConfigFlagsTable&);

static DWORD BackgroundFinishMarkWaitTime(bool, Js::ConfigFlagsTable&);

static size\_t MinBackgroundRepeatMarkRescanBytes(Js::ConfigFlagsTable&);

static DWORD FinishConcurrentCollectWaitTime(Js::ConfigFlagsTable&);

static DWORD PriorityBoostTimeout(Js::ConfigFlagsTable&);

#endif

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

static bool PartialConcurrentNextCollection(double ratio, Js::ConfigFlagsTable& flags);

#endif

// Constant heuristics

static const uint IdleUncollectedAllocBytesCollection = 1 MEGABYTES;

static const uint TickCountCollection = 1200; // 1.2 second

static const uint TickCountFinishCollection = 45; // 45 milliseconds

static const uint TickCountDoDispose = 300; // Allow for 300 milliseconds of script before attempting dispose

// This heuristic is currently used for dispose on stack probes

void ConfigureBaseFactor(uint baseFactor);

#ifdef CONCURRENT\_GC\_ENABLED

static const uint MaxBackgroundRepeatMarkCount = 2;

// If we rescan at least 128 pages in the first background repeat mark,

// then trigger a second repeat mark pass.

static const uint BackgroundSecondRepeatMarkThreshold = 128;

#endif

private:

#ifndef RECYCLER\_HEURISTIC\_VERSION

#define RECYCLER\_HEURISTIC\_VERSION 11

#endif

static const uint DefaultUncollectedAllocBytesCollection = 1 MEGABYTES;

#ifdef CONCURRENT\_GC\_ENABLED

static const uint TickCountConcurrentPriorityBoost = 5000; // 5 second

static const DWORD DefaultFinishConcurrentCollectWaitTime = 1000; // 1 second

static const uint DefaultMaxBackgroundFinishMarkCount = 1;

static const DWORD DefaultBackgroundFinishMarkWaitTime = 15; // ms

static const size\_t DefaultMinBackgroundRepeatMarkRescanBytes = 1 MEGABYTES;

#endif

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef PROFILE\_RECYCLER\_ALLOC

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAS")

RecyclerObjectDumper::DumpFunctionMap \* RecyclerObjectDumper::dumpFunctionMap = nullptr;

RecyclerObjectDumper RecyclerObjectDumper::Instance;

RecyclerObjectDumper::~RecyclerObjectDumper()

{

if (dumpFunctionMap)

{

NoCheckHeapDelete(dumpFunctionMap);

}

}

BOOL

RecyclerObjectDumper::EnsureDumpFunctionMap()

{

if (dumpFunctionMap == nullptr)

{

dumpFunctionMap = NoCheckHeapNew(DumpFunctionMap, &NoCheckHeapAllocator::Instance);

}

return (dumpFunctionMap != nullptr);

}

void

RecyclerObjectDumper::RegisterDumper(type\_info const \* typeinfo, DumpFunction dumperFunction)

{

if (EnsureDumpFunctionMap())

{

Assert(!dumpFunctionMap->ContainsKey(typeinfo));

dumpFunctionMap->Add(typeinfo, dumperFunction);

}

}

void

RecyclerObjectDumper::DumpObject(type\_info const \* typeinfo, bool isArray, void \* objectAddress)

{

if (typeinfo == nullptr)

{

Output::Print(L"Address %p", objectAddress);

}

else

{

DumpFunction dumpFunction;

if (dumpFunctionMap == nullptr || !dumpFunctionMap->TryGetValue(typeinfo, &dumpFunction) || !dumpFunction(typeinfo, isArray, objectAddress))

{

Output::Print(isArray? L"%S[] %p" : L"%S %p", typeinfo->name(), objectAddress);

}

}

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

#ifdef PROFILE\_RECYCLER\_ALLOC

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

class RecyclerObjectDumper

{

public:

typedef bool (\*DumpFunction)(type\_info const \* typeinfo, bool isArray, void \* objectAddress);

static void RegisterDumper(type\_info const \* typeinfo, DumpFunction dumperFunction);

static void DumpObject(type\_info const \* typeinfo, bool isArray, void \* objectAddress);

private:

RecyclerObjectDumper() {}

~RecyclerObjectDumper();

static RecyclerObjectDumper Instance;

static BOOL EnsureDumpFunctionMap();

typedef JsUtil::BaseDictionary<type\_info const \*, RecyclerObjectDumper::DumpFunction, NoCheckHeapAllocator> DumpFunctionMap;

static DumpFunctionMap \* dumpFunctionMap;

};

template <typename T, RecyclerObjectDumper::DumpFunction dumpFunction>

class AutoRegisterRecyclerObjectDumper

{

public:

static AutoRegisterRecyclerObjectDumper Instance;

private:

AutoRegisterRecyclerObjectDumper()

{

RecyclerObjectDumper::RegisterDumper(&typeid(T), dumpFunction);

}

};

template <typename T, RecyclerObjectDumper::DumpFunction dumpFunction>

AutoRegisterRecyclerObjectDumper<T, dumpFunction> AutoRegisterRecyclerObjectDumper<T, dumpFunction>::Instance;

#define AUTO\_REGISTER\_RECYCLER\_OBJECT\_DUMPER(T, func) template AutoRegisterRecyclerObjectDumper<T, func>;

#else

#define AUTO\_REGISTER\_RECYCLER\_OBJECT\_DUMPER(T, func)

#endif

void DumpRecyclerObjectGraph();

#else

#define AUTO\_REGISTER\_RECYCLER\_OBJECT\_DUMPER(T, func)

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

RecyclerObjectGraphDumper::RecyclerObjectGraphDumper(Recycler \* recycler, RecyclerObjectGraphDumper::Param \* param) :

recycler(recycler),

param(param),

dumpObjectName(nullptr),

dumpObject(nullptr),

isOutOfMemory(false)

#ifdef PROFILE\_RECYCLER\_ALLOC

, dumpObjectTypeInfo(nullptr)

#endif

{

recycler->objectGraphDumper = this;

}

RecyclerObjectGraphDumper::~RecyclerObjectGraphDumper()

{

recycler->objectGraphDumper = nullptr;

}

void RecyclerObjectGraphDumper::BeginDumpObject(wchar\_t const \* name)

{

Assert(dumpObjectName == nullptr);

Assert(dumpObject == nullptr);

dumpObjectName = name;

}

void RecyclerObjectGraphDumper::BeginDumpObject(wchar\_t const \* name, void \* address)

{

Assert(dumpObjectName == nullptr);

Assert(dumpObject == nullptr);

swprintf\_s(tempObjectName, \_countof(tempObjectName), L"%s %p", name, address);

dumpObjectName = tempObjectName;

}

void RecyclerObjectGraphDumper::BeginDumpObject(void \* objectAddress)

{

Assert(dumpObjectName == nullptr);

Assert(dumpObject == nullptr);

this->dumpObject = objectAddress;

#ifdef PROFILE\_RECYCLER\_ALLOC

if (recycler->trackerDictionary)

{

Recycler::TrackerData \* trackerData = recycler->GetTrackerData(objectAddress);

if (trackerData != nullptr)

{

this->dumpObjectTypeInfo = trackerData->typeinfo;

this->dumpObjectIsArray = trackerData->isArray;

}

else

{

Assert(false);

this->dumpObjectTypeInfo = nullptr;

this->dumpObjectIsArray = nullptr;

}

}

#endif

}

void RecyclerObjectGraphDumper::EndDumpObject()

{

Assert(this->dumpObjectName != nullptr || this->dumpObject != nullptr);

this->dumpObjectName = nullptr;

this->dumpObject = nullptr;

}

void RecyclerObjectGraphDumper::DumpObjectReference(void \* objectAddress, bool remark)

{

if (this->param == nullptr || !this->param->dumpRootOnly || recycler->collectionState == CollectionStateFindRoots)

{

if (this->param != nullptr && this->param->dumpReferenceFunc)

{

if (!this->param->dumpReferenceFunc(this->dumpObjectName, this->dumpObject, objectAddress))

return;

}

Output::Print(L"\"");

if (this->dumpObjectName)

{

Output::Print(L"%s", this->dumpObjectName);

}

else

{

Assert(this->dumpObject != nullptr);

#ifdef PROFILE\_RECYCLER\_ALLOC

RecyclerObjectDumper::DumpObject(this->dumpObjectTypeInfo, this->dumpObjectIsArray, this->dumpObject);

#else

Output::Print(L"Address %p", objectAddress);

#endif

}

Output::Print(remark? L"\" => \"" : L"\" -> \"");

recycler->DumpObjectDescription(objectAddress);

Output::Print(L"\"\n");

}

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

class RecyclerObjectGraphDumper

{

public:

struct Param

{

bool (\*dumpReferenceFunc)(wchar\_t const \*, void \*objectAddress, void \*referenceAddress);

bool dumpRootOnly;

bool skipStack;

#ifdef RECYCLER\_STATS

RecyclerCollectionStats stats;

#endif

};

RecyclerObjectGraphDumper(Recycler \* recycler, Param \* param);

~RecyclerObjectGraphDumper();

void BeginDumpObject(void \* objectAddres);

void BeginDumpObject(wchar\_t const \* name);

void BeginDumpObject(wchar\_t const \* name, void\* objectAddress);

void EndDumpObject();

void DumpObjectReference(void \* objectAddress, bool remark);

Recycler \* recycler;

Param \* param;

wchar\_t const \* dumpObjectName;

wchar\_t tempObjectName[256];

void \* dumpObject;

#ifdef PROFILE\_RECYCLER\_ALLOC

type\_info const \* dumpObjectTypeInfo;

bool dumpObjectIsArray;

#endif

bool isOutOfMemory;

};

#endif

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

#define BEGIN\_DUMP\_OBJECT(recycler, address) { if (recycler->objectGraphDumper != nullptr) { recycler->objectGraphDumper->BeginDumpObject(address); }

#define BEGIN\_DUMP\_OBJECT\_ADDRESS(name, address) { if (this->objectGraphDumper != nullptr) { this->objectGraphDumper->BeginDumpObject(name, address); }

#define DUMP\_OBJECT\_REFERENCE(recycler, address) if (recycler->objectGraphDumper != nullptr) { recycler->objectGraphDumper->DumpObjectReference(address, false); }

#define DUMP\_OBJECT\_REFERENCE\_REMARK(recycler, address) if (recycler->objectGraphDumper != nullptr && recycler->IsValidObject(address)) { recycler->objectGraphDumper->DumpObjectReference(address, true); }

#define END\_DUMP\_OBJECT(recycler) if (recycler->objectGraphDumper != nullptr) { recycler->objectGraphDumper->EndDumpObject(); } }

#define DUMP\_IMPLICIT\_ROOT(recycler, address) BEGIN\_DUMP\_OBJECT(recycler, L"Implicit Root"); DUMP\_OBJECT\_REFERENCE(recycler, address); END\_DUMP\_OBJECT(recycler);

#else

#define BEGIN\_DUMP\_OBJECT(recycler, address)

#define BEGIN\_DUMP\_OBJECT\_ADDRESS(name, address)

#define DUMP\_OBJECT\_REFERENCE(recycler, address)

#define DUMP\_OBJECT\_REFERENCE\_REMARK(recycler, address)

#define END\_DUMP\_OBJECT(recycler)

#define DUMP\_IMPLICIT\_ROOT(recycler, address)

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

RecyclerPageAllocator::RecyclerPageAllocator(Recycler\* recycler, AllocationPolicyManager \* policyManager,

#ifndef JD\_PRIVATE

Js::ConfigFlagsTable& flagTable,

#endif

uint maxFreePageCount, uint maxAllocPageCount)

: IdleDecommitPageAllocator(policyManager,

PageAllocatorType\_Recycler,

#ifndef JD\_PRIVATE

flagTable,

#endif

0, maxFreePageCount,

true, &zeroPageQueue, maxAllocPageCount)

{

this->recycler = recycler;

}

bool RecyclerPageAllocator::IsMemProtectMode()

{

return recycler->IsMemProtectMode();

}

void

RecyclerPageAllocator::EnableWriteWatch()

{

Assert(segments.Empty());

Assert(fullSegments.Empty());

Assert(emptySegments.Empty());

Assert(decommitSegments.Empty());

Assert(largeSegments.Empty());

allocFlags = MEM\_WRITE\_WATCH;

}

bool

RecyclerPageAllocator::ResetWriteWatch()

{

if (allocFlags != MEM\_WRITE\_WATCH)

{

return false;

}

GCETW(GC\_RESETWRITEWATCH\_START, (this));

SuspendIdleDecommit();

bool success = true;

// Only reset write watch on allocated pages

if (!ResetWriteWatch(&segments) ||

!ResetWriteWatch(&decommitSegments) ||

!ResetAllWriteWatch(&fullSegments) ||

!ResetAllWriteWatch(&largeSegments))

{

allocFlags = 0;

success = false;

}

ResumeIdleDecommit();

GCETW(GC\_RESETWRITEWATCH\_STOP, (this));

return success;

}

bool

RecyclerPageAllocator::ResetWriteWatch(DListBase<PageSegment> \* segmentList)

{

DListBase<PageSegment>::Iterator i(segmentList);

while (i.Next())

{

PageSegment& segment = i.Data();

size\_t pageCount = segment.GetAvailablePageCount();

Assert(pageCount <= MAXUINT32);

PageSegment::PageBitVector unallocPages = segment.GetUnAllocatedPages();

for (uint index = 0u; index < pageCount; index++)

{

if (unallocPages.Test(index))

{

continue;

}

char \* address = segment.GetAddress() + index \* AutoSystemInfo::PageSize;

if (::ResetWriteWatch(address, AutoSystemInfo::PageSize) != 0)

{

#if DBG\_DUMP

Output::Print(L"ResetWriteWatch failed for %p\n", address);

Output::Flush();

#endif

// shouldn't happen

Assert(false);

return false;

}

}

}

return true;

}

template <typename T>

bool

RecyclerPageAllocator::ResetAllWriteWatch(DListBase<T> \* segmentList)

{

DListBase<T>::Iterator i(segmentList);

while (i.Next())

{

T& segment = i.Data();

if (::ResetWriteWatch(segment.GetAddress(), segment.GetPageCount() \* AutoSystemInfo::PageSize ) != 0)

{

#if DBG\_DUMP

Output::Print(L"ResetWriteWatch failed for %p\n", segment.GetAddress());

Output::Flush();

#endif

// shouldn't happen

Assert(false);

return false;

}

}

return true;

}

#if DBG

size\_t

RecyclerPageAllocator::GetWriteWatchPageCount()

{

if (allocFlags != MEM\_WRITE\_WATCH)

{

return 0;

}

SuspendIdleDecommit();

// Only reset write watch on allocated pages

size\_t count = GetWriteWatchPageCount(&segments)

+ GetWriteWatchPageCount(&decommitSegments)

+ GetAllWriteWatchPageCount(&fullSegments)

+ GetAllWriteWatchPageCount(&largeSegments);

ResumeIdleDecommit();

return count;

}

size\_t

RecyclerPageAllocator::GetWriteWatchPageCount(DListBase<PageSegment> \* segmentList)

{

size\_t totalCount = 0;

DListBase<PageSegment>::Iterator i(segmentList);

while (i.Next())

{

PageSegment& segment = i.Data();

size\_t pageCount = segment.GetAvailablePageCount();

Assert(pageCount <= MAXUINT32);

PageSegment::PageBitVector unallocPages = segment.GetUnAllocatedPages();

for (uint index = 0u; index < pageCount; index++)

{

if (unallocPages.Test(index))

{

continue;

}

char \* address = segment.GetAddress() + index \* AutoSystemInfo::PageSize;

void \* written;

ULONG\_PTR count = 0;

DWORD pageSize = AutoSystemInfo::PageSize;

if (::GetWriteWatch(0, address, AutoSystemInfo::PageSize, &written, &count, &pageSize) == 0)

{

#if DBG\_DUMP

Output::Print(L"GetWriteWatch failed for %p\n", segment.GetAddress());

Output::Flush();

#endif

// shouldn't happen

Assert(false);

}

else

{

Assert(count <= 1);

Assert(pageSize == AutoSystemInfo::PageSize);

Assert(count == 0 || written == address);

totalCount += count;

}

}

}

return totalCount;

}

template <typename T>

size\_t

RecyclerPageAllocator::GetAllWriteWatchPageCount(DListBase<T> \* segmentList)

{

size\_t totalCount = 0;

DListBase<T>::Iterator i(segmentList);

while (i.Next())

{

T& segment = i.Data();

for (uint i = 0; i < segment.GetPageCount(); i++)

{

void \* address = segment.GetAddress() + i \* AutoSystemInfo::PageSize;

void \* written;

ULONG\_PTR count = 0;

DWORD pageSize = AutoSystemInfo::PageSize;

if (::GetWriteWatch(0, address, AutoSystemInfo::PageSize, &written, &count, &pageSize) == 0)

{

#if DBG\_DUMP

Output::Print(L"GetWriteWatch failed for %p\n", segment.GetAddress());

Output::Flush();

#endif

// shouldn't happen

Assert(false);

}

else

{

Assert(count <= 1);

Assert(pageSize == AutoSystemInfo::PageSize);

Assert(count == 0 || written == address);

totalCount += count;

}

}

}

return totalCount;

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

namespace Memory

{

class RecyclerPageAllocator : public IdleDecommitPageAllocator

{

public:

RecyclerPageAllocator(Recycler\* recycler, AllocationPolicyManager \* policyManager,

#ifndef JD\_PRIVATE

Js::ConfigFlagsTable& flagTable,

#endif

uint maxFreePageCount, uint maxAllocPageCount = PageAllocator::DefaultMaxAllocPageCount);

void EnableWriteWatch();

bool ResetWriteWatch();

static uint const DefaultPrimePageCount = 0x1000; // 16MB

#if DBG

size\_t GetWriteWatchPageCount();

#endif

private:

static bool ResetWriteWatch(DListBase<PageSegment> \* segmentList);

template <typename T>

static bool ResetAllWriteWatch(DListBase<T> \* segmentList);

#if DBG

static size\_t GetWriteWatchPageCount(DListBase<PageSegment> \* segmentList);

template <typename T>

static size\_t GetAllWriteWatchPageCount(DListBase<T> \* segmentList);

#endif

ZeroPageQueue zeroPageQueue;

Recycler\* recycler;

bool IsMemProtectMode();

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

template <typename T>

class NoWriteBarrierField

{

public:

NoWriteBarrierField() {}

NoWriteBarrierField(T const& value) : value(value) {}

// Getters

operator T const&() const { return value; }

operator T&() { return value; }

T const\* AddressOf() const { return &value; }

T\* AddressOf() { return &value; }

// Setters

NoWriteBarrierField& operator=(T const& value)

{

this->value = value;

return \*this;

}

private:

T value;

};

template <typename T>

class NoWriteBarrierPtr

{

public:

NoWriteBarrierPtr() {}

NoWriteBarrierPtr(T \* value) : value(value) {}

// Getters

T \* operator->() const { return this->value; }

operator T\*() const { return this->value; }

// Setters

NoWriteBarrierPtr& operator=(T const& value)

{

this->value = value;

return \*this;

}

private:

T \* value;

};

template <typename T>

class WriteBarrierObjectConstructorTrigger

{

public:

WriteBarrierObjectConstructorTrigger(T\* object, Recycler\* recycler):

object((char\*) object),

recycler(recycler)

{

}

~WriteBarrierObjectConstructorTrigger()

{

// WriteBarrier-TODO: trigger write barrier if the GC is in concurrent mark state

}

operator T\*()

{

return object;

}

private:

T\* object;

Recycler\* recycler;

};

template <typename T>

class WriteBarrierPtr

{

public:

WriteBarrierPtr() {}

WriteBarrierPtr(T \* ptr)

{

// WriteBarrier

NoWriteBarrierSet(ptr);

}

// Getters

T \* operator->() const { return ptr; }

operator T\*() const { return ptr; }

// Setters

WriteBarrierPtr& operator=(T \* ptr)

{

WriteBarrierSet(ptr);

return \*this;

}

void NoWriteBarrierSet(T \* ptr)

{

this->ptr = ptr;

}

void WriteBarrierSet(T \* ptr)

{

NoWriteBarrierSet(ptr);

#ifdef RECYCLER\_WRITE\_BARRIER

RecyclerWriteBarrierManager::WriteBarrier(this);

#endif

}

WriteBarrierPtr& operator=(WriteBarrierPtr const& other)

{

WriteBarrierSet(other.ptr);

return \*this;

}

static void MoveArray(WriteBarrierPtr \* dst, WriteBarrierPtr \* src, size\_t count)

{

memmove((void \*)dst, src, sizeof(WriteBarrierPtr) \* count);

#ifdef RECYCLER\_WRITE\_BARRIER

RecyclerWriteBarrierManager::WriteBarrier(dst, count);

#endif

}

static void CopyArray(WriteBarrierPtr \* dst, size\_t dstCount, T const\* src, size\_t srcCount)

{

js\_memcpy\_s((void \*)dst, sizeof(WriteBarrierPtr) \* dstCount, src, sizeof(T \*) \* srcCount);

#ifdef RECYCLER\_WRITE\_BARRIER

RecyclerWriteBarrierManager::WriteBarrier(dst, dstCount);

#endif

}

static void CopyArray(WriteBarrierPtr \* dst, size\_t dstCount, WriteBarrierPtr const\* src, size\_t srcCount)

{

js\_memcpy\_s((void \*)dst, sizeof(WriteBarrierPtr) \* dstCount, src, sizeof(WriteBarrierPtr) \* srcCount);

#ifdef RECYCLER\_WRITE\_BARRIER

RecyclerWriteBarrierManager::WriteBarrier(dst, dstCount);

#endif

}

static void ClearArray(WriteBarrierPtr \* dst, size\_t count)

{

// assigning NULL don't need write barrier, just cast it and null it out

memset((void \*)dst, 0, sizeof(WriteBarrierPtr<T>) \* count);

}

private:

T \* ptr;

};

}

template<class T> inline

const T& min(const T& a, const NoWriteBarrierField<T>& b) { return a < b ? a : b; }

template<class T> inline

const T& min(const NoWriteBarrierField<T>& a, const T& b) { return a < b ? a : b; }

template<class T> inline

const T& min(const NoWriteBarrierField<T>& a, const NoWriteBarrierField<T>& b) { return a < b ? a : b; }

template<class T> inline

const T& max(const NoWriteBarrierField<T>& a, const T& b) { return a > b ? a : b; }

// TODO: Add this method back once we figure out why OACR is tripping on it

template<class T> inline

const T& max(const T& a, const NoWriteBarrierField<T>& b) { return a > b ? a : b; }

template<class T> inline

const T& max(const NoWriteBarrierField<T>& a, const NoWriteBarrierField<T>& b) { return a > b ? a : b; }

// Disallow memcpy, memmove of WriteBarrierPtr

template <typename T>

void \* \_\_cdecl memmove(\_Out\_writes\_bytes\_all\_opt\_(\_Size) WriteBarrierPtr<T> \* \_Dst, \_In\_reads\_bytes\_opt\_(\_Size) const void \* \_Src, \_In\_ size\_t \_Size)

{

CompileAssert(false);

}

template <typename T>

void \_\_stdcall js\_memcpy\_s(\_\_bcount(sizeInBytes) WriteBarrierPtr<T> \*dst, size\_t sizeInBytes, \_\_bcount(count) const void \*src, size\_t count)

{

CompileAssert(false);

}

template <typename T>

void \* \_\_cdecl memset(\_Out\_writes\_bytes\_all\_(\_Size) WriteBarrierPtr<T> \* \_Dst, \_In\_ int \_Val, \_In\_ size\_t \_Size)

{

CompileAssert(false);

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

template <typename T>

class RecyclerRootPtr

{

public:

RecyclerRootPtr() : ptr(nullptr) {};

~RecyclerRootPtr() { Assert(ptr == nullptr); }

void Root(T \* ptr, Recycler \* recycler) { Assert(this->ptr == nullptr); recycler->RootAddRef(ptr); this->ptr = ptr; }

void Unroot(Recycler \* recycler) { Assert(this->ptr != nullptr); recycler->RootRelease(this->ptr); this->ptr = nullptr; }

T \* operator->() const { Assert(ptr != nullptr); return ptr; }

operator T\*() const { return ptr; }

protected:

T \* ptr;

private:

RecyclerRootPtr(const RecyclerRootPtr<T>& ptr); // Disable

RecyclerRootPtr& operator=(RecyclerRootPtr<T> const& ptr); // Disable

};

typedef RecyclerRootPtr<void> RecyclerRootVar;

template <typename T>

class AutoRecyclerRootPtr : public RecyclerRootPtr<T>

{

public:

AutoRecyclerRootPtr(T \* ptr, Recycler \* recycler) : recycler(recycler)

{

Root(ptr);

}

~AutoRecyclerRootPtr()

{

Unroot();

}

void Root(T \* ptr)

{

Unroot();

\_\_super::Root(ptr, recycler);

}

void Unroot()

{

if (ptr != nullptr)

{

\_\_super::Unroot(recycler);

}

}

Recycler \* GetRecycler() const

{

return recycler;

}

private:

Recycler \* const recycler;

};

typedef AutoRecyclerRootPtr<void> AutoRecyclerRootVar;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef PARTIAL\_GC\_ENABLED

#define KILOBYTES \* 1024

#define MEGABYTES \* 1024 KILOBYTES

#define MEGABYTES\_OF\_PAGES \* 1024 \* 1024 / AutoSystemInfo::PageSize;

const uint RecyclerSweep::MinPartialUncollectedNewPageCount = 4 MEGABYTES\_OF\_PAGES;

const uint RecyclerSweep::MaxPartialCollectRescanRootBytes = 5 MEGABYTES;

static const uint MinPartialCollectRescanRootBytes = 128 KILOBYTES;

// Maximum unused partial collect free bytes before we get out of partial GC mode

static const uint MaxUnusedPartialCollectFreeBytes = 16 MEGABYTES;

// Have to collected at least 10% before we would partial GC

// CONSIDER: It may be good to do partial with low efficacy once we have concurrent partial

// because old object are not getting collected as well, but without concurrent partial, we will have to mark

// new objects in thread.

static const double MinPartialCollectEfficacy = 0.1;

#endif

bool

RecyclerSweep::IsMemProtectMode()

{

return recycler->IsMemProtectMode();

}

void

RecyclerSweep::BeginSweep(Recycler \* recycler, size\_t rescanRootBytes, bool adjustPartialHeuristics)

{

{

// We are about to sweep, give the runtime a chance to see the now-immutable state of the world.

// And clean up all the cache not monitor by the GC (e.g. inline caches)

AUTO\_NO\_EXCEPTION\_REGION;

recycler->collectionWrapper->PreSweepCallback();

}

Assert(!recycler->IsSweeping());

Assert(recycler->recyclerSweep == nullptr);

memset(this, 0, sizeof(RecyclerSweep));

this->recycler = recycler;

recycler->recyclerSweep = this;

// We might still have block that has disposed but not put back into the allocable

// heap block list yet, which happens if we finish disposing object during concurrent

// reset mark and can't

// modify the heap block lists

// CONCURRENT-TODO: Consider doing it during FinishDisposeObjects to get these block

// available sooner as well. We will still need it here as we only always get to

// finish dispose before sweep.

this->FlushPendingTransferDisposedObjects();

#ifdef CONCURRENT\_GC\_ENABLED

// Take the small heap block new heap block list and store in RecyclerSweep temporary

// We get merge later before we start sweeping the bucket.

leafData.pendingMergeNewHeapBlockList = recycler->autoHeap.newLeafHeapBlockList;

normalData.pendingMergeNewHeapBlockList = recycler->autoHeap.newNormalHeapBlockList;

#ifdef RECYCLER\_WRITE\_BARRIER

withBarrierData.pendingMergeNewHeapBlockList = recycler->autoHeap.newNormalWithBarrierHeapBlockList;

finalizableWithBarrierData.pendingMergeNewHeapBlockList = recycler->autoHeap.newFinalizableWithBarrierHeapBlockList;

#endif

finalizableData.pendingMergeNewHeapBlockList = recycler->autoHeap.newFinalizableHeapBlockList;

mediumLeafData.pendingMergeNewHeapBlockList = recycler->autoHeap.newMediumLeafHeapBlockList;

mediumNormalData.pendingMergeNewHeapBlockList = recycler->autoHeap.newMediumNormalHeapBlockList;

#ifdef RECYCLER\_WRITE\_BARRIER

mediumWithBarrierData.pendingMergeNewHeapBlockList = recycler->autoHeap.newMediumNormalWithBarrierHeapBlockList;

mediumFinalizableWithBarrierData.pendingMergeNewHeapBlockList = recycler->autoHeap.newMediumFinalizableWithBarrierHeapBlockList;

#endif

mediumFinalizableData.pendingMergeNewHeapBlockList = recycler->autoHeap.newMediumFinalizableHeapBlockList;

recycler->autoHeap.newLeafHeapBlockList = nullptr;

recycler->autoHeap.newNormalHeapBlockList = nullptr;

recycler->autoHeap.newFinalizableHeapBlockList = nullptr;

#ifdef RECYCLER\_WRITE\_BARRIER

recycler->autoHeap.newNormalWithBarrierHeapBlockList = nullptr;

recycler->autoHeap.newFinalizableWithBarrierHeapBlockList = nullptr;

#endif

recycler->autoHeap.newMediumLeafHeapBlockList = nullptr;

recycler->autoHeap.newMediumNormalHeapBlockList = nullptr;

recycler->autoHeap.newMediumFinalizableHeapBlockList = nullptr;

#ifdef RECYCLER\_WRITE\_BARRIER

recycler->autoHeap.newMediumNormalWithBarrierHeapBlockList = nullptr;

recycler->autoHeap.newMediumFinalizableWithBarrierHeapBlockList = nullptr;

#endif

#endif

#ifdef PARTIAL\_GC\_ENABLED

Assert(recycler->clientTrackedObjectList.Empty());

// We should not have partialUncollectedAllocBytes unless we are in partial collect at this point

Assert(recycler->partialUncollectedAllocBytes == 0 || recycler->inPartialCollectMode);

Assert(recycler->autoHeap.uncollectedAllocBytes >= recycler->partialUncollectedAllocBytes);

// if the cost of rescan is too high, we want to disable partial GC starting from the

// upcoming Sweep. We basically move the check up from AdjustPartialHeuristics to here

// such that we can have the decision before sweep.

this->rescanRootBytes = rescanRootBytes;

RECYCLER\_STATS\_SET(recycler, rescanRootBytes, rescanRootBytes);

if (this->DoPartialCollectMode())

{

// enable partial collect for sweep & next round of GC

DebugOnly(this->partial = true);

// REVIEW: is adjustPartialHeuristicsMode the same as in PartialCollectMode?

this->adjustPartialHeuristics = adjustPartialHeuristics;

this->StartPartialCollectMode();

}

else

{

// disable partial collect

if (recycler->inPartialCollectMode)

{

recycler->FinishPartialCollect();

}

Assert(recycler->partialUncollectedAllocBytes == 0);

Assert(!recycler->inPartialCollectMode);

}

#endif

if (this->inPartialCollect)

{

// We just did a partial collect.

// We only want to count objects that survived this collect towards the next full GC.

// Thus, clear out uncollectedAllocBytes here; we will adjust to account for objects that

// survived this partial collect in EndSweep.

recycler->ResetHeuristicCounters();

}

else

{

// We just did a full collect.

// We reset uncollectedAllocBytes when we kicked off the collection,

// so don't reset it here (but do reset partial heuristics).

recycler->ResetPartialHeuristicCounters();

}

}

void

RecyclerSweep::FinishSweep()

{

#ifdef PARTIAL\_GC\_ENABLED

Assert(this->partial == recycler->inPartialCollectMode);

// Adjust heuristics

if (recycler->inPartialCollectMode)

{

if (this->AdjustPartialHeuristics())

{

GCETW(GC\_SWEEP\_PARTIAL\_REUSE\_PAGE\_START, (recycler));

// If we are doing a full concurrent GC, all allocated bytes are consider "collected".

// We only start accumulating uncollected allocate bytes during partial GC.

// FinishPartialCollect will reset it to 0 if we are not doing a partial GC

recycler->partialUncollectedAllocBytes = this->InPartialCollect()? this->nextPartialUncollectedAllocBytes : 0;

#ifdef RECYCLER\_TRACE

if (recycler->GetRecyclerFlagsTable().Trace.IsEnabled(Js::PartialCollectPhase))

{

Output::Print(L"AdjustPartialHeuristics returned true\n");

Output::Print(L" partialUncollectedAllocBytes = %d\n", recycler->partialUncollectedAllocBytes);

Output::Print(L" nextPartialUncollectedAllocBytes = %d\n", this->nextPartialUncollectedAllocBytes);

}

#endif

recycler->autoHeap.SweepPartialReusePages(\*this);

GCETW(GC\_SWEEP\_PARTIAL\_REUSE\_PAGE\_STOP, (recycler));

if (!this->IsBackground())

{

RECYCLER\_PROFILE\_EXEC\_BEGIN(recycler, Js::ResetWriteWatchPhase);

if (!recycler->recyclerPageAllocator.ResetWriteWatch() ||

!recycler->recyclerLargeBlockPageAllocator.ResetWriteWatch())

{

// Shouldn't happen

Assert(false);

recycler->enablePartialCollect = false;

recycler->FinishPartialCollect(this);

}

RECYCLER\_PROFILE\_EXEC\_END(recycler, Js::ResetWriteWatchPhase);

}

}

else

{

#ifdef RECYCLER\_TRACE

if (recycler->GetRecyclerFlagsTable().Trace.IsEnabled(Js::PartialCollectPhase))

{

Output::Print(L"AdjustPartialHeuristics returned false\n");

}

#endif

if (this->IsBackground())

{

recycler->BackgroundFinishPartialCollect(this);

}

else

{

recycler->FinishPartialCollect(this);

}

}

}

else

{

Assert(!this->adjustPartialHeuristics);

// Initial value or Sweep should have called FinishPartialCollect to these if we are not doing partial

Assert(recycler->partialUncollectedAllocBytes == 0);

}

recycler->SweepPendingObjects(\*this);

#endif

}

void

RecyclerSweep::EndSweep()

{

#ifdef PARTIAL\_GC\_ENABLED

// We clear out the old uncollectedAllocBytes, restore it now to get the adjustment for partial

// We clear it again after we are done collecting and if we are not in partial collect

if (this->inPartialCollect)

{

recycler->autoHeap.uncollectedAllocBytes += this->nextPartialUncollectedAllocBytes;

#ifdef RECYCLER\_TRACE

if (recycler->GetRecyclerFlagsTable().Trace.IsEnabled(Js::PartialCollectPhase))

{

Output::Print(L"EndSweep for partial sweep\n");

Output::Print(L" uncollectedAllocBytes = %d\n", recycler->autoHeap.uncollectedAllocBytes);

Output::Print(L" nextPartialUncollectedAllocBytes = %d\n", this->nextPartialUncollectedAllocBytes);

}

#endif

}

#endif

recycler->recyclerSweep = nullptr;

// Clean up the HeapBlockMap.

// This will release any internal structures that are no longer needed after Sweep.

recycler->heapBlockMap.Cleanup(!recycler->IsMemProtectMode());

}

void

RecyclerSweep::BackgroundSweep()

{

this->BeginBackground(forceForeground);

if (GetRecycler()->IsPageHeapEnabled())

{

// Finish the concurrent part of the first pass

this->recycler->autoHeap.SweepSmallNonFinalizable<true>(\*this);

}

else

{

// Finish the concurrent part of the first pass

this->recycler->autoHeap.SweepSmallNonFinalizable<false>(\*this);

}

// Finish the rest of the sweep

this->FinishSweep();

this->EndBackground();

}

Recycler \*

RecyclerSweep::GetRecycler() const

{

return recycler;

}

bool

RecyclerSweep::IsBackground() const

{

return this->background;

}

bool

RecyclerSweep::HasSetupBackgroundSweep() const

{

return this->IsBackground() || this->forceForeground;

}

void

RecyclerSweep::FlushPendingTransferDisposedObjects()

{

if (recycler->hasPendingTransferDisposedObjects)

{

// If recycler->inResolveExternalWeakReferences is true, the recycler isn't really disposing anymore

// so it's safe to call transferDisposedObjects

Assert(!recycler->inDispose || recycler->inResolveExternalWeakReferences);

Assert(!recycler->hasDisposableObject);

recycler->autoHeap.TransferDisposedObjects();

}

}

void

RecyclerSweep::ShutdownCleanup()

{

SmallLeafHeapBucketT<SmallAllocationBlockAttributes>::DeleteHeapBlockList(this->leafData.pendingMergeNewHeapBlockList, recycler);

SmallNormalHeapBucket::DeleteHeapBlockList(this->normalData.pendingMergeNewHeapBlockList, recycler);

#ifdef RECYCLER\_WRITE\_BARRIER

SmallNormalWithBarrierHeapBucket::DeleteHeapBlockList(this->withBarrierData.pendingMergeNewHeapBlockList, recycler);

SmallFinalizableWithBarrierHeapBucket::DeleteHeapBlockList(this->finalizableWithBarrierData.pendingMergeNewHeapBlockList, recycler);

#endif

SmallFinalizableHeapBucket::DeleteHeapBlockList(this->finalizableData.pendingMergeNewHeapBlockList, recycler);

for (uint i = 0; i < HeapConstants::BucketCount; i++)

{

// For leaf, we can always reuse the page as we don't need to rescan them for partial GC

// It should have been swept immediately during Sweep

Assert(this->leafData.bucketData[i].pendingSweepList == nullptr);

SmallNormalHeapBucket::DeleteHeapBlockList(this->normalData.bucketData[i].pendingSweepList, recycler);

SmallFinalizableHeapBucket::DeleteHeapBlockList(this->finalizableData.bucketData[i].pendingSweepList, recycler);

#ifdef RECYCLER\_WRITE\_BARRIER

SmallFinalizableWithBarrierHeapBucket::DeleteHeapBlockList(this->finalizableWithBarrierData.bucketData[i].pendingSweepList, recycler);

#endif

SmallLeafHeapBucket::DeleteEmptyHeapBlockList(this->leafData.bucketData[i].pendingEmptyBlockList);

SmallNormalHeapBucket::DeleteEmptyHeapBlockList(this->normalData.bucketData[i].pendingEmptyBlockList);

#ifdef RECYCLER\_WRITE\_BARRIER

SmallNormalWithBarrierHeapBucket::DeleteEmptyHeapBlockList(this->withBarrierData.bucketData[i].pendingEmptyBlockList);

Assert(this->finalizableWithBarrierData.bucketData[i].pendingEmptyBlockList == nullptr);

#endif

Assert(this->finalizableData.bucketData[i].pendingEmptyBlockList == nullptr);

}

MediumLeafHeapBucket::DeleteHeapBlockList(this->mediumLeafData.pendingMergeNewHeapBlockList, recycler);

MediumNormalHeapBucket::DeleteHeapBlockList(this->mediumNormalData.pendingMergeNewHeapBlockList, recycler);

#ifdef RECYCLER\_WRITE\_BARRIER

MediumNormalWithBarrierHeapBucket::DeleteHeapBlockList(this->mediumWithBarrierData.pendingMergeNewHeapBlockList, recycler);

MediumFinalizableWithBarrierHeapBucket::DeleteHeapBlockList(this->mediumFinalizableWithBarrierData.pendingMergeNewHeapBlockList, recycler);

#endif

MediumFinalizableHeapBucket::DeleteHeapBlockList(this->mediumFinalizableData.pendingMergeNewHeapBlockList, recycler);

for (uint i = 0; i < HeapConstants::MediumBucketCount; i++)

{

// For leaf, we can always reuse the page as we don't need to rescan them for partial GC

// It should have been swept immediately during Sweep

Assert(this->mediumLeafData.bucketData[i].pendingSweepList == nullptr);

MediumNormalHeapBucket::DeleteHeapBlockList(this->mediumNormalData.bucketData[i].pendingSweepList, recycler);

MediumFinalizableHeapBucket::DeleteHeapBlockList(this->mediumFinalizableData.bucketData[i].pendingSweepList, recycler);

#ifdef RECYCLER\_WRITE\_BARRIER

MediumFinalizableWithBarrierHeapBucket::DeleteHeapBlockList(this->mediumFinalizableWithBarrierData.bucketData[i].pendingSweepList, recycler);

#endif

MediumLeafHeapBucket::DeleteEmptyHeapBlockList(this->mediumLeafData.bucketData[i].pendingEmptyBlockList);

MediumNormalHeapBucket::DeleteEmptyHeapBlockList(this->mediumNormalData.bucketData[i].pendingEmptyBlockList);

#ifdef RECYCLER\_WRITE\_BARRIER

MediumNormalWithBarrierHeapBucket::DeleteEmptyHeapBlockList(this->mediumWithBarrierData.bucketData[i].pendingEmptyBlockList);

Assert(this->mediumFinalizableWithBarrierData.bucketData[i].pendingEmptyBlockList == nullptr);

#endif

Assert(this->mediumFinalizableData.bucketData[i].pendingEmptyBlockList == nullptr);

}

}

#ifdef CONCURRENT\_GC\_ENABLED

template <typename TBlockType>

void

RecyclerSweep::MergePendingNewHeapBlockList()

{

TBlockType \*& blockList = this->GetData<TBlockType>().pendingMergeNewHeapBlockList;

TBlockType \* list = blockList;

blockList = nullptr;

HeapInfo& heapInfo = recycler->autoHeap;

HeapBlockList::ForEachEditing(list, [&heapInfo](TBlockType \* heapBlock)

{

auto& bucket = heapInfo.GetBucket<TBlockType::RequiredAttributes>(heapBlock->GetObjectSize());

bucket.MergeNewHeapBlock(heapBlock);

});

}

template void RecyclerSweep::MergePendingNewHeapBlockList<SmallLeafHeapBlock>();

template void RecyclerSweep::MergePendingNewHeapBlockList<SmallNormalHeapBlock>();

template void RecyclerSweep::MergePendingNewHeapBlockList<SmallFinalizableHeapBlock>();

#ifdef RECYCLER\_WRITE\_BARRIER

template void RecyclerSweep::MergePendingNewHeapBlockList<SmallNormalWithBarrierHeapBlock>();

template void RecyclerSweep::MergePendingNewHeapBlockList<SmallFinalizableWithBarrierHeapBlock>();

#endif

template <typename TBlockType>

void

RecyclerSweep::MergePendingNewMediumHeapBlockList()

{

TBlockType \*& blockList = this->GetData<TBlockType>().pendingMergeNewHeapBlockList;

TBlockType \* list = blockList;

blockList = nullptr;

HeapInfo& heapInfo = recycler->autoHeap;

HeapBlockList::ForEachEditing(list, [&heapInfo](TBlockType \* heapBlock)

{

auto& bucket = heapInfo.GetMediumBucket<TBlockType::RequiredAttributes>(heapBlock->GetObjectSize());

bucket.MergeNewHeapBlock(heapBlock);

});

}

template void RecyclerSweep::MergePendingNewMediumHeapBlockList<MediumLeafHeapBlock>();

template void RecyclerSweep::MergePendingNewMediumHeapBlockList<MediumNormalHeapBlock>();

template void RecyclerSweep::MergePendingNewMediumHeapBlockList<MediumFinalizableHeapBlock>();

#ifdef RECYCLER\_WRITE\_BARRIER

template void RecyclerSweep::MergePendingNewMediumHeapBlockList<MediumNormalWithBarrierHeapBlock>();

template void RecyclerSweep::MergePendingNewMediumHeapBlockList<MediumFinalizableWithBarrierHeapBlock>();

#endif

bool

RecyclerSweep::HasPendingEmptyBlocks() const

{

return this->hasPendingEmptyBlocks;

}

bool

RecyclerSweep::HasPendingSweepSmallHeapBlocks() const

{

return this->hasPendingSweepSmallHeapBlocks;

}

void

RecyclerSweep::SetHasPendingSweepSmallHeapBlocks()

{

this->hasPendingSweepSmallHeapBlocks = true;

}

void

RecyclerSweep::BeginBackground(bool forceForeground)

{

Assert(!background);

this->background = !forceForeground;

this->forceForeground = forceForeground;

}

void

RecyclerSweep::EndBackground()

{

Assert(this->background || this->forceForeground);

this->background = false;

}

#if DBG

bool

RecyclerSweep::HasPendingNewHeapBlocks() const

{

return leafData.pendingMergeNewHeapBlockList != nullptr

|| normalData.pendingMergeNewHeapBlockList != nullptr

|| finalizableData.pendingMergeNewHeapBlockList != nullptr

#ifdef RECYCLER\_WRITE\_BARRIER

|| withBarrierData.pendingMergeNewHeapBlockList != nullptr

|| finalizableWithBarrierData.pendingMergeNewHeapBlockList != nullptr

#endif

|| mediumLeafData.pendingMergeNewHeapBlockList != nullptr

|| mediumNormalData.pendingMergeNewHeapBlockList != nullptr

|| mediumFinalizableData.pendingMergeNewHeapBlockList != nullptr

#ifdef RECYCLER\_WRITE\_BARRIER

|| mediumWithBarrierData.pendingMergeNewHeapBlockList != nullptr

|| mediumFinalizableWithBarrierData.pendingMergeNewHeapBlockList != nullptr

#endif

;

}

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t

RecyclerSweep::SetPendingMergeNewHeapBlockCount()

{

return HeapBlockList::Count(leafData.pendingMergeNewHeapBlockList)

+ HeapBlockList::Count(normalData.pendingMergeNewHeapBlockList)

+ HeapBlockList::Count(finalizableData.pendingMergeNewHeapBlockList)

#ifdef RECYCLER\_WRITE\_BARRIER

+ HeapBlockList::Count(withBarrierData.pendingMergeNewHeapBlockList)

+ HeapBlockList::Count(finalizableWithBarrierData.pendingMergeNewHeapBlockList)

#endif

+ HeapBlockList::Count(mediumLeafData.pendingMergeNewHeapBlockList)

+ HeapBlockList::Count(mediumNormalData.pendingMergeNewHeapBlockList)

+ HeapBlockList::Count(mediumFinalizableData.pendingMergeNewHeapBlockList)

#ifdef RECYCLER\_WRITE\_BARRIER

+ HeapBlockList::Count(mediumWithBarrierData.pendingMergeNewHeapBlockList)

+ HeapBlockList::Count(mediumFinalizableWithBarrierData.pendingMergeNewHeapBlockList)

#endif

;

}

#endif

#endif

#ifdef PARTIAL\_GC\_ENABLED

bool

RecyclerSweep::InPartialCollectMode() const

{

return recycler->inPartialCollectMode;

}

bool

RecyclerSweep::InPartialCollect() const

{

return this->inPartialCollect;

}

void

RecyclerSweep::StartPartialCollectMode()

{

// Save the in partial collect, the main thread reset it after returning to the script

// and the background thread still needs it

this->inPartialCollect = recycler->inPartialCollectMode;

recycler->inPartialCollectMode = true;

// Tracks the unallocated alloc bytes for partial GC

// Keep a copy Last collection's uncollected allocation bytes, so we can use it to calculate

// the new object that is allocated since the last GC

Assert(recycler->partialUncollectedAllocBytes == 0 || this->inPartialCollect);

this->lastPartialUncollectedAllocBytes = recycler->partialUncollectedAllocBytes;

size\_t currentUncollectedAllocBytes = recycler->autoHeap.uncollectedAllocBytes;

Assert(currentUncollectedAllocBytes >= this->lastPartialUncollectedAllocBytes);

if (!this->inPartialCollect)

{

// If we did a full collect, then we need to include lastUncollectedAllocBytes

// in the partialUncollectedAllocBytes calculation, because all objects allocated

// since the previous GC are considered new, but we cleared uncollectedAllocBytes

// when we kicked off the GC.

currentUncollectedAllocBytes += recycler->autoHeap.lastUncollectedAllocBytes;

}

// Initially, the partial uncollected alloc bytes is the current uncollectedAllocBytes

recycler->partialUncollectedAllocBytes = currentUncollectedAllocBytes;

this->nextPartialUncollectedAllocBytes = currentUncollectedAllocBytes;

#ifdef RECYCLER\_TRACE

if (recycler->GetRecyclerFlagsTable().Trace.IsEnabled(Js::PartialCollectPhase))

{

Output::Print(L"StartPartialCollectMode\n");

Output::Print(L" was inPartialCollectMode = %d\n", this->inPartialCollect);

Output::Print(L" lastPartialUncollectedAllocBytes = %d\n", this->lastPartialUncollectedAllocBytes);

Output::Print(L" uncollectedAllocBytes = %d\n", recycler->autoHeap.uncollectedAllocBytes);

Output::Print(L" nextPartialUncollectedAllocBytes = %d\n", this->nextPartialUncollectedAllocBytes);

}

#endif

}

// Called by prepare sweep to track the new allocated bytes on block that is not fully allocated yet.

template <typename TBlockAttributes>

void

RecyclerSweep::AddUnaccountedNewObjectAllocBytes(SmallHeapBlockT<TBlockAttributes> \* heapBlock)

{

#ifdef PARTIAL\_GC\_ENABLED

// Only need to update the unaccounted alloc bytes if we are in partial collect mode

if (recycler->inPartialCollectMode)

{

uint unaccountedAllocBytes = heapBlock->GetAndClearUnaccountedAllocBytes();

Assert(heapBlock->lastUncollectedAllocBytes == 0 || unaccountedAllocBytes == 0);

DebugOnly(heapBlock->lastUncollectedAllocBytes += unaccountedAllocBytes);

recycler->partialUncollectedAllocBytes += unaccountedAllocBytes;

this->nextPartialUncollectedAllocBytes += unaccountedAllocBytes;

}

else

#endif

{

// We don't care, clear the unaccounted to start tracking for new object for next GC

heapBlock->ClearAllAllocBytes();

}

}

template void RecyclerSweep::AddUnaccountedNewObjectAllocBytes<SmallAllocationBlockAttributes>(SmallHeapBlock \* heapBlock);

template void RecyclerSweep::AddUnaccountedNewObjectAllocBytes<MediumAllocationBlockAttributes>(MediumHeapBlock \* heapBlock);

void

RecyclerSweep::SubtractSweepNewObjectAllocBytes(size\_t newObjectExpectSweepByteCount)

{

Assert(recycler->inPartialCollectMode);

// We shouldn't free more then we allocated

Assert(this->nextPartialUncollectedAllocBytes >= newObjectExpectSweepByteCount);

Assert(this->nextPartialUncollectedAllocBytes >= this->lastPartialUncollectedAllocBytes + newObjectExpectSweepByteCount);

this->nextPartialUncollectedAllocBytes -= newObjectExpectSweepByteCount;

}

/\*--------------------------------------------------------------------------------------------

\* Determine we want to go into partial collect mode for the next GC before we sweep,

\* based on the number bytes needed to rescan (<= 5MB)

\*--------------------------------------------------------------------------------------------\*/

bool

RecyclerSweep::DoPartialCollectMode()

{

if (!recycler->enablePartialCollect)

{

return false;

}

// If we exceed 16MB of unused memory in partial blocks, get out of partial collect to avoid

// memory fragmentation.

if (recycler->autoHeap.unusedPartialCollectFreeBytes > MaxUnusedPartialCollectFreeBytes)

{

return false;

}

return this->rescanRootBytes <= MaxPartialCollectRescanRootBytes;

}

// Heuristic ratio is ((c \* e + (1 - e)) \* (1 - p)) + p and use that to linearly scale between min and max

// This give cost/efficacy/pressure equal weight, while each can push it pass where partial GC is not

// beneficial

bool

RecyclerSweep::AdjustPartialHeuristics()

{

Assert(recycler->inPartialCollectMode);

Assert(this->adjustPartialHeuristics);

Assert(this->InPartialCollect() || recycler->autoHeap.unusedPartialCollectFreeBytes == 0);

// DoPartialCollectMode should have rejected these already

Assert(this->rescanRootBytes <= (size\_t)MaxPartialCollectRescanRootBytes);

Assert(recycler->autoHeap.unusedPartialCollectFreeBytes <= MaxUnusedPartialCollectFreeBytes);

// Page reuse Heuristics

double collectEfficacy;

const size\_t allocBytes = this->GetNewObjectAllocBytes();

if (allocBytes == 0)

{

// We may get collections without allocating memory (e.g. unpin heuristics).

collectEfficacy = 1.0; // assume 100% efficacy

this->partialCollectSmallHeapBlockReuseMinFreeBytes = 0; // reuse all pages

}

else

{

const size\_t freedBytes = this->GetNewObjectFreeBytes();

Assert(freedBytes <= allocBytes);

collectEfficacy = (double)freedBytes / (double)allocBytes;

// If we collected less then 10% of the memory, let's not do partial GC.

// CONSIDER: It may be good to do partial with low efficacy once we have concurrent partial

// because old object are not getting collected as well, but without concurrent partial, we will have to mark

// new objects in thread.

if (collectEfficacy < MinPartialCollectEfficacy)

{

return false;

}

// Scale the efficacy linearly such that an efficacy of MinPartialCollectEfficacy translates to an adjusted efficacy of

// 0.0, and an efficacy of 1.0 translates to an adjusted efficacy of 1.0

collectEfficacy = (collectEfficacy - MinPartialCollectEfficacy) / (1.0 - MinPartialCollectEfficacy);

Assert(collectEfficacy <= 1.0);

this->partialCollectSmallHeapBlockReuseMinFreeBytes = (size\_t)(AutoSystemInfo::PageSize \* collectEfficacy);

}

#ifdef RECYCLER\_STATS

recycler->collectionStats.collectEfficacy = collectEfficacy;

recycler->collectionStats.partialCollectSmallHeapBlockReuseMinFreeBytes = this->partialCollectSmallHeapBlockReuseMinFreeBytes;

#endif

// Blocks which are being reused are likely to be touched again from allocation and contribute to Rescan cost.

// If there are many of these, adjust rescanRootBytes to account for this.

const size\_t estimatedPartialReuseBlocks = (size\_t)((double)this->reuseHeapBlockCount \* (1.0 - collectEfficacy));

const size\_t estimatedPartialReuseBytes = estimatedPartialReuseBlocks \* AutoSystemInfo::PageSize;

const size\_t newRescanRootBytes = max(this->rescanRootBytes, estimatedPartialReuseBytes);

RECYCLER\_STATS\_SET(recycler, estimatedPartialReuseBytes, estimatedPartialReuseBytes);

// Recheck the rescanRootBytes

if (newRescanRootBytes > MaxPartialCollectRescanRootBytes)

{

return false;

}

double collectCost = (double)newRescanRootBytes / MaxPartialCollectRescanRootBytes;

RECYCLER\_STATS\_SET(recycler, collectCost, collectCost);

// Include the efficacy in equal portion, which is related to the cost of marking through new objects.

// r = c \* e + 1 - e;

const double reuseRatio = 1.0 - collectEfficacy;

double ratio = collectCost \* collectEfficacy + reuseRatio;

if (this->InPartialCollect())

{

// Avoid ratio of uncollectedBytesPressure > 1.0

if (this->nextPartialUncollectedAllocBytes > RecyclerHeuristic::Instance.MaxUncollectedAllocBytesPartialCollect)

{

return false;

}

// Only add full collect pressure if we are doing partial collect,

// account for the amount of uncollected bytes and unused bytes to increase

// pressure to do a full GC by rising the partial GC new page heuristic

double uncollectedBytesPressure = (double)this->nextPartialUncollectedAllocBytes / (double)RecyclerHeuristic::Instance.MaxUncollectedAllocBytesPartialCollect;

double collectFullCollectPressure =

(double)recycler->autoHeap.unusedPartialCollectFreeBytes / (double)MaxUnusedPartialCollectFreeBytes

\* (1.0 - uncollectedBytesPressure) + uncollectedBytesPressure;

ratio = ratio \* (1.0 - collectFullCollectPressure) + collectFullCollectPressure;

}

Assert(0.0 <= ratio && ratio <= 1.0);

// Linear scale the partial GC new page heuristic using the ratio calculated

recycler->uncollectedNewPageCountPartialCollect = MinPartialUncollectedNewPageCount

+ (size\_t)((double)(RecyclerHeuristic::Instance.MaxPartialUncollectedNewPageCount - MinPartialUncollectedNewPageCount) \* ratio);

Assert(recycler->uncollectedNewPageCountPartialCollect >= MinPartialUncollectedNewPageCount &&

recycler->uncollectedNewPageCountPartialCollect <= RecyclerHeuristic::Instance.MaxPartialUncollectedNewPageCount);

// If the number of new page to reach the partial heuristics plus the existing uncollectedAllocBytes

// and the memory we are going to reuse (assume we use it all) is greater then the full GC max size heuristic

// (with 1M fudge factor), we trigger a full GC anyways, so let's not get into partial GC

const size\_t estimatedPartialReusedFreeByteCount = (size\_t)((double)this->reuseByteCount \* reuseRatio);

if (recycler->uncollectedNewPageCountPartialCollect \* AutoSystemInfo::PageSize

+ this->nextPartialUncollectedAllocBytes + estimatedPartialReusedFreeByteCount >= RecyclerHeuristic::Instance.MaxUncollectedAllocBytesPartialCollect)

{

return false;

}

recycler->partialConcurrentNextCollection = RecyclerHeuristic::PartialConcurrentNextCollection(ratio, recycler->GetRecyclerFlagsTable());

return true;

}

size\_t

RecyclerSweep::GetNewObjectAllocBytes() const

{

Assert(recycler->inPartialCollectMode);

Assert(recycler->partialUncollectedAllocBytes >= this->lastPartialUncollectedAllocBytes);

return recycler->partialUncollectedAllocBytes - this->lastPartialUncollectedAllocBytes;

}

size\_t

RecyclerSweep::GetNewObjectFreeBytes() const

{

Assert(recycler->inPartialCollectMode);

Assert(recycler->partialUncollectedAllocBytes >= this->nextPartialUncollectedAllocBytes);

return recycler->partialUncollectedAllocBytes - this->nextPartialUncollectedAllocBytes;

}

size\_t

RecyclerSweep::GetPartialUnusedFreeByteCount() const

{

return partialUnusedFreeByteCount;

}

size\_t

RecyclerSweep::GetPartialCollectSmallHeapBlockReuseMinFreeBytes() const

{

return partialCollectSmallHeapBlockReuseMinFreeBytes;

}

template <typename TBlockAttributes>

void

RecyclerSweep::NotifyAllocableObjects(SmallHeapBlockT<TBlockAttributes> \* heapBlock)

{

this->reuseByteCount += heapBlock->GetExpectedFreeBytes();

if (!heapBlock->IsLeafBlock())

{

this->reuseHeapBlockCount++;

}

}

template void RecyclerSweep::NotifyAllocableObjects<SmallAllocationBlockAttributes>(SmallHeapBlock\* heapBlock);

template void RecyclerSweep::NotifyAllocableObjects<MediumAllocationBlockAttributes>(MediumHeapBlock\* heapBlock);

void

RecyclerSweep::AddUnusedFreeByteCount(uint expectFreeByteCount)

{

this->partialUnusedFreeByteCount += expectFreeByteCount;

}

bool

RecyclerSweep::DoAdjustPartialHeuristics() const

{

return this->adjustPartialHeuristics;

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

// RecyclerSweep - Sweeping algorithm and state

class RecyclerSweep

#if DBG

: public RecyclerVerifyListConsistencyData

#endif

{

public:

void BeginSweep(Recycler \* recycler, size\_t rescanRootBytes, bool adjustPartialHeuristics);

void FinishSweep();

void EndSweep();

void ShutdownCleanup();

Recycler \* GetRecycler() const;

bool IsBackground() const;

bool HasSetupBackgroundSweep() const;

void FlushPendingTransferDisposedObjects();

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

bool HasPendingSweepSmallHeapBlocks() const;

void SetHasPendingSweepSmallHeapBlocks();

template <typename TBlockType>

TBlockType \*& GetPendingSweepBlockList(HeapBucketT<TBlockType> const \* heapBucket);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

bool HasPendingEmptyBlocks() const;

template <typename TBlockType, bool pageheap> void QueueEmptyHeapBlock(HeapBucketT<TBlockType> const \*heapBucket, TBlockType \* heapBlock);

template <typename TBlockType> void TransferPendingEmptyHeapBlocks(HeapBucketT<TBlockType> \* heapBucket);

void BackgroundSweep();

void BeginBackground(bool forceForeground);

void EndBackground();

template <typename TBlockType> void SetPendingMergeNewHeapBlockList(TBlockType \* heapBlockList);

template <typename TBlockType> void MergePendingNewHeapBlockList();

template <typename TBlockType> void MergePendingNewMediumHeapBlockList();

#if DBG

bool HasPendingNewHeapBlocks() const;

template <typename TBlockType> TBlockType \* GetSavedNextAllocableBlockHead(HeapBucketT<TBlockType> const \* heapBucket);

template <typename TBlockType> void SaveNextAllocableBlockHead(HeapBucketT<TBlockType> const \* heapBucket);

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

template < typename TBlockType> size\_t GetHeapBlockCount(HeapBucketT<TBlockType> const \* heapBucket);

size\_t SetPendingMergeNewHeapBlockCount();

#endif

#endif

#ifdef PARTIAL\_GC\_ENABLED

bool InPartialCollectMode() const;

bool InPartialCollect() const;

void StartPartialCollectMode();

bool DoPartialCollectMode();

bool DoAdjustPartialHeuristics() const;

bool AdjustPartialHeuristics();

template <typename TBlockAttributes>

void AddUnaccountedNewObjectAllocBytes(SmallHeapBlockT<TBlockAttributes> \* smallHeapBlock);

void SubtractSweepNewObjectAllocBytes(size\_t newObjectExpectSweepByteCount);

size\_t GetNewObjectAllocBytes() const;

size\_t GetNewObjectFreeBytes() const;

size\_t GetPartialUnusedFreeByteCount() const;

size\_t GetPartialCollectSmallHeapBlockReuseMinFreeBytes() const;

template <typename TBlockAttributes>

void NotifyAllocableObjects(SmallHeapBlockT<TBlockAttributes> \* smallHeapBlock);

void AddUnusedFreeByteCount(uint expectedFreeByteCount);

static const uint MinPartialUncollectedNewPageCount; // 4MB pages

static const uint MaxPartialCollectRescanRootBytes; // 5MB

#endif

private:

template <typename TBlockType>

struct BucketData

{

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

TBlockType \* pendingSweepList;

TBlockType \* pendingFinalizableSweptList;

#endif

#ifdef CONCURRENT\_GC\_ENABLED

TBlockType \* pendingEmptyBlockList;

TBlockType \* pendingEmptyBlockListTail;

#if DBG

TBlockType \* savedNextAllocableBlockHead;

#endif

#endif

};

template <typename TBlockType>

BucketData<TBlockType>& GetBucketData(HeapBucketT<TBlockType> const \* bucket)

{

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

return this->GetData<TBlockType>().bucketData[bucket->GetBucketIndex()];

}

else

{

Assert(TBlockType::HeapBlockAttributes::IsMediumBlock);

return this->GetData<TBlockType>().bucketData[bucket->GetMediumBucketIndex()];

}

}

template <typename TBlockType>

struct Data

{

BucketData<TBlockType> bucketData[TBlockType::HeapBlockAttributes::BucketCount];

#ifdef CONCURRENT\_GC\_ENABLED

TBlockType \* pendingMergeNewHeapBlockList;

#endif

};

template <typename TBlockType> Data<TBlockType>& GetData();

template <> Data<SmallLeafHeapBlock>& GetData<SmallLeafHeapBlock>() { return leafData; }

template <> Data<SmallNormalHeapBlock>& GetData<SmallNormalHeapBlock>() { return normalData; }

template <> Data<SmallFinalizableHeapBlock>& GetData<SmallFinalizableHeapBlock>() { return finalizableData; }

#ifdef RECYCLER\_WRITE\_BARRIER

template <> Data<SmallNormalWithBarrierHeapBlock>& GetData<SmallNormalWithBarrierHeapBlock>() { return withBarrierData; }

template <> Data<SmallFinalizableWithBarrierHeapBlock>& GetData<SmallFinalizableWithBarrierHeapBlock>() { return finalizableWithBarrierData; }

#endif

template <> Data<MediumLeafHeapBlock>& GetData<MediumLeafHeapBlock>() { return mediumLeafData; }

template <> Data<MediumNormalHeapBlock>& GetData<MediumNormalHeapBlock>() { return mediumNormalData; }

template <> Data<MediumFinalizableHeapBlock>& GetData<MediumFinalizableHeapBlock>() { return mediumFinalizableData; }

#ifdef RECYCLER\_WRITE\_BARRIER

template <> Data<MediumNormalWithBarrierHeapBlock>& GetData<MediumNormalWithBarrierHeapBlock>() { return mediumWithBarrierData; }

template <> Data<MediumFinalizableWithBarrierHeapBlock>& GetData<MediumFinalizableWithBarrierHeapBlock>() { return mediumFinalizableWithBarrierData; }

#endif

private:

bool IsMemProtectMode();

Recycler \* recycler;

Data<SmallLeafHeapBlock> leafData;

Data<SmallNormalHeapBlock> normalData;

Data<SmallFinalizableHeapBlock> finalizableData;

#ifdef RECYCLER\_WRITE\_BARRIER

Data<SmallNormalWithBarrierHeapBlock> withBarrierData;

Data<SmallFinalizableWithBarrierHeapBlock> finalizableWithBarrierData;

#endif

Data<MediumLeafHeapBlock> mediumLeafData;

Data<MediumNormalHeapBlock> mediumNormalData;

Data<MediumFinalizableHeapBlock> mediumFinalizableData;

#ifdef RECYCLER\_WRITE\_BARRIER

Data<MediumNormalWithBarrierHeapBlock> mediumWithBarrierData;

Data<MediumFinalizableWithBarrierHeapBlock> mediumFinalizableWithBarrierData;

#endif

bool background;

bool forceForeground;

bool hasPendingSweepSmallHeapBlocks;

bool hasPendingEmptyBlocks;

bool inPartialCollect;

#ifdef PARTIAL\_GC\_ENABLED

bool adjustPartialHeuristics;

size\_t lastPartialUncollectedAllocBytes;

size\_t nextPartialUncollectedAllocBytes;

// Sweep data for partial activation heuristic

size\_t rescanRootBytes;

size\_t reuseHeapBlockCount;

size\_t reuseByteCount;

// Partial reuse Heuristic

size\_t partialCollectSmallHeapBlockReuseMinFreeBytes;

// Data to update unusedPartialCollectFreeBytes

size\_t partialUnusedFreeByteCount;

#if DBG

bool partial;

#endif

#endif

};

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

template <typename TBlockType>

TBlockType \*&

RecyclerSweep::GetPendingSweepBlockList(HeapBucketT<TBlockType> const \* heapBucket)

{

return this->GetBucketData<TBlockType>(heapBucket).pendingSweepList;

}

#endif

#ifdef CONCURRENT\_GC\_ENABLED

template <typename TBlockType, bool pageheap>

void

RecyclerSweep::QueueEmptyHeapBlock(HeapBucketT<TBlockType> const \*heapBucket, TBlockType \* heapBlock)

{

auto& bucketData = this->GetBucketData(heapBucket);

Assert(heapBlock->heapBucket == heapBucket);

heapBlock->BackgroundReleasePagesSweep<pageheap>(recycler);

TBlockType \* list = bucketData.pendingEmptyBlockList;

if (list == nullptr)

{

Assert(bucketData.pendingEmptyBlockListTail == nullptr);

bucketData.pendingEmptyBlockListTail = heapBlock;

this->hasPendingEmptyBlocks = true;

}

heapBlock->SetNextBlock(list);

bucketData.pendingEmptyBlockList = heapBlock;

}

template <typename TBlockType>

void

RecyclerSweep::TransferPendingEmptyHeapBlocks(HeapBucketT<TBlockType> \* heapBucket)

{

Assert(!this->IsBackground());

Assert(!heapBucket->IsAllocationStopped());

RECYCLER\_SLOW\_CHECK(heapBucket->VerifyHeapBlockCount(false));

auto& bucketData = this->GetBucketData(heapBucket);

Assert(bucketData.pendingSweepList == nullptr);

TBlockType \* list = bucketData.pendingEmptyBlockList;

if (list)

{

TBlockType \* tail = bucketData.pendingEmptyBlockListTail;

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t count = 0;

HeapBlockList::ForEach(list, [tail, &count](TBlockType \* heapBlock)

{

Assert(heapBlock->GetAddress() == nullptr);

Assert(heapBlock->GetSegment() == nullptr);

Assert(heapBlock->GetNextBlock() != nullptr || heapBlock == tail);

count++;

});

RECYCLER\_SLOW\_CHECK(heapBucket->emptyHeapBlockCount += count);

RECYCLER\_SLOW\_CHECK(heapBucket->heapBlockCount -= count);

#endif

tail->SetNextBlock(heapBucket->emptyBlockList);

heapBucket->emptyBlockList = list;

bucketData.pendingEmptyBlockList = nullptr;

RECYCLER\_SLOW\_CHECK(heapBucket->VerifyHeapBlockCount(false));

}

else

{

Assert(bucketData.pendingEmptyBlockListTail == nullptr);

}

}

template <typename TBlockType>

void

RecyclerSweep::SetPendingMergeNewHeapBlockList(TBlockType \* heapBlockList)

{

this->GetData<TBlockType>().pendingMergeNewHeapBlockList = heapBlockList;

}

#if DBG

template <typename TBlockType>

TBlockType \*

RecyclerSweep::GetSavedNextAllocableBlockHead(HeapBucketT<TBlockType> const \* heapBucket)

{

return this->GetBucketData(heapBucket).savedNextAllocableBlockHead;

}

template <typename TBlockType>

void

RecyclerSweep::SaveNextAllocableBlockHead(HeapBucketT<TBlockType> const \* heapBucket)

{

this->GetBucketData(heapBucket).savedNextAllocableBlockHead = heapBucket->nextAllocableBlockHead;

}

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

template < typename TBlockType>

size\_t

RecyclerSweep::GetHeapBlockCount(HeapBucketT<TBlockType> const \* heapBucket)

{

auto& bucketData = this->GetBucketData(heapBucket);

return HeapBlockList::Count(bucketData.pendingSweepList)

+ HeapBlockList::Count(bucketData.pendingEmptyBlockList);

}

#endif

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

namespace Memory

{

/\*

\* Telemetry timestamp macros

\*

\* To record a particular timestamp, use RECORD\_TIMESTAMP. This overwrites the previous timestamp.

\* To have auto-start/end events logged, use the AUTO\_TIMESTAMP macro.

\*/

class AutoTimestamp

{

public:

AutoTimestamp(FILETIME \* startTimestamp, FILETIME \* endTimestamp) : endTimestamp(endTimestamp)

{

::GetSystemTimeAsFileTime(startTimestamp);

}

~AutoTimestamp()

{

::GetSystemTimeAsFileTime(endTimestamp);

}

private:

FILETIME \* endTimestamp;

};

#define RECORD\_TIMESTAMP(Field) ::GetSystemTimeAsFileTime(&telemetryBlock->Field);

#define INC\_TIMESTAMP\_FIELD(Field) telemetryBlock->Field++;

#define AUTO\_TIMESTAMP(Field) Memory::AutoTimestamp timestamp\_##Field(&telemetryBlock->Field##StartTime, &telemetryBlock->Field##EndTime);

struct RecyclerWatsonTelemetryBlock

{

FILETIME initialCollectionStartTime;

DWORDLONG initialCollectionStartProcessUsedBytes;

FILETIME currentCollectionStartTime;

DWORDLONG currentCollectionStartProcessUsedBytes;

FILETIME concurrentMarkFinishTime;

FILETIME disposeStartTime;

FILETIME disposeEndTime;

FILETIME externalWeakReferenceObjectResolveStartTime;

FILETIME externalWeakReferenceObjectResolveEndTime;

FILETIME currentCollectionEndTime;

FILETIME lastCollectionEndTime;

DWORD exhaustiveRepeatedCount;

};

};

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Js

{

template <typename TStrongRef> class RemoteWeakReference;

};

namespace Memory

{

// Forward declarations

template <typename SizePolicy> class WeakReferenceHashTable;

class Recycler;

///

/// This class is represents a weak reference handle object

/// It's a proxy to the actual strong reference itself

/// This is an entry in the weak reference hash table.

/// When a weak reference is created, the strong reference is set to

/// point to an object.

/// When the referenced object is collected, the strong reference is set to NULL

///

/// Clients should not use this class but instead use RecyclerWeakReference<type>

/// which provides for automatic type conversion of the underlying reference

class RecyclerWeakReferenceBase

{

public:

friend class Recycler;

template <typename SizePolicy> friend class WeakReferenceHashTable;

template <typename TStrongRef> friend class Js::RemoteWeakReference;

protected:

char\* strongRef;

HeapBlock \* strongRefHeapBlock;

SmallHeapBlock \* weakRefHeapBlock;

RecyclerWeakReferenceBase\* next;

#if DBG

type\_info const \* typeInfo;

#if defined TRACK\_ALLOC && defined(PERF\_COUNTERS)

PerfCounter::Counter \* counter;

#endif

#endif

};

/// Wrapper class template that can be used to acquire the underlying strong reference from the weak reference

template<typename T>

class RecyclerWeakReference: public RecyclerWeakReferenceBase

{

public:

// Fast get of the strong reference- this might return a wrong result if the recycler is in sweep so callers

// should never call this during sweep

\_\_inline T\* FastGet() const

{

return ((T\*) strongRef);

}

\_\_inline T\* Get() const

{

char \* ref = this->strongRef;

return (T\*)ref;

}

\_\_inline T\*\* GetAddressOfStrongRef()

{

return (T\*\*)&strongRef;

}

};

template <class T>

class WeakReferenceCache

{

private:

RecyclerWeakReference<T> \* weakReference;

public:

WeakReferenceCache() : weakReference(nullptr) {};

RecyclerWeakReference<T> \* GetWeakReference(Recycler \* recycler)

{

RecyclerWeakReference<T> \* weakRef = this->weakReference;

if (weakRef == nullptr)

{

weakRef = recycler->CreateWeakReferenceHandle((T\*)this);

this->weakReference = weakRef;

}

return weakRef;

}

};

///

/// Hashtable class that maps strong references to weak references

/// This is slightly unique in that the weak reference is a complete entry in the hashtable

/// but is treated as the value for the client. The key is the strong reference.

/// The hash table is a standard closed-addressing hash table where the strong references are

/// hashed into buckets, and then stored in that buckets corresponding doubly linked list

/// The hash-table is resized when its load factor exceeds a constant

/// The buckets are allocated using the HeapAllocator. Individual entries are recycler allocated.

///

template <typename SizePolicy>

class WeakReferenceHashTable

{

static const int MaxAverageChainLength = 1;

HeapAllocator\* allocator;

RecyclerWeakReferenceBase\*\* buckets;

uint count;

uint size;

RecyclerWeakReferenceBase\* freeList;

public:

WeakReferenceHashTable(uint size, HeapAllocator\* allocator):

count(0),

size(0),

allocator(allocator),

freeList(nullptr)

{

this->size = SizePolicy::GetSize(size);

buckets = AllocatorNewArrayZ(HeapAllocator, allocator, RecyclerWeakReferenceBase\*, this->size);

}

~WeakReferenceHashTable()

{

AllocatorDeleteArray(HeapAllocator, allocator, size, buckets);

}

RecyclerWeakReferenceBase\* Add(char\* strongReference, Recycler \* recycler)

{

uint targetBucket = HashKeyToBucket(strongReference, size);

RecyclerWeakReferenceBase\* entry = FindEntry(strongReference, targetBucket);

if (entry != nullptr)

{

return entry;

}

return Create(strongReference, targetBucket, recycler);

}

bool FindOrAdd(char\* strongReference, Recycler \*recycler, RecyclerWeakReferenceBase \*\*ppWeakRef)

{

Assert(ppWeakRef);

uint targetBucket = HashKeyToBucket(strongReference, size);

RecyclerWeakReferenceBase\* entry = FindEntry(strongReference, targetBucket);

if (entry != nullptr)

{

\*ppWeakRef = entry;

return false;

}

entry = Create(strongReference, targetBucket, recycler);

\*ppWeakRef = entry;

return true;

}

#ifdef RECYCLER\_TRACE\_WEAKREF

void DumpNode(RecyclerWeakReferenceBase\* node)

{

Output::Print(L"[ 0x%08x { 0x%08x, 0x%08x }]", node, node->strongRef, mode->next);

}

void Dump()

{

RecyclerWeakReferenceBase \*current;

Output::Print(L"HashTable with %d buckets and %d nodes\n", this->size, this->count);

for (uint i=0;i<size;i++) {

Output::Print(L"Bucket %d (0x%08x) ==> ", i, &buckets[i]);

for (current = buckets[i] ; current != nullptr; current = current->next) {

DumpNode(current);

}

Output::Print(L"\n");

}

}

#endif

bool TryGetValue(char\* strongReference, RecyclerWeakReferenceBase\*\* weakReference)

{

RecyclerWeakReferenceBase\* current = FindEntry(strongReference, HashKeyToBucket(strongReference, size));

if (current != nullptr)

{

\*weakReference = current;

return true;

}

return false;

}

void Remove(char\* key, RecyclerWeakReferenceBase\*\* pOut)

{

uint val = HashKeyToBucket(key, size);

RecyclerWeakReferenceBase \*\* pprev = &buckets[val];

RecyclerWeakReferenceBase \*current = \*pprev;

while (current)

{

if (DefaultComparer<char\*>::Equals(key, current->strongRef))

{

\*pprev = current->next;

if (pOut != nullptr)

{

(\*pOut) = current;

}

count--;

#ifdef RECYCLER\_TRACE\_WEAKREF

Output::Print(L"Remove 0x%08x to bucket %d, count is %d\n", current, val, count);

#endif

break;

}

pprev = &current->next;

current = \*pprev;

}

}

void Remove(char\* key)

{

Remove(key, nullptr);

}

template <class Func>

void Map(Func fn)

{

uint removed = 0;

#if DEBUG

uint countedEntries = 0;

#endif

for (uint i=0;i<size;i++)

{

RecyclerWeakReferenceBase \*\* pprev = &buckets[i];

RecyclerWeakReferenceBase \*current = \*pprev;

while (current)

{

if (fn(current))

{

pprev = &current->next;

#if DEBUG

countedEntries++;

#endif

}

else

{

// remove

\*pprev = current->next;

removed++;

}

current = \*pprev;

}

}

Assert(removed <= count);

count -= removed;

#if DEBUG

Assert(countedEntries == count);

#endif

}

private:

// If density is a compile-time constant, then we can optimize (avoids division)

// Sometimes the compiler can also make this optimization, but this way is guaranteed.

template< uint density > bool IsDenserThan() const

{

return count > (size \* density);

}

RecyclerWeakReferenceBase \* FindEntry(char\* strongReference, uint targetBucket)

{

for (RecyclerWeakReferenceBase \* current = buckets[targetBucket] ; current != nullptr; current = current->next)

{

if (DefaultComparer<char\*>::Equals(strongReference, current->strongRef))

{

return current;

}

}

return nullptr;

}

uint HashKeyToBucket(char\* strongReference, int size)

{

uint hashCode = DefaultComparer<char\*>::GetHashCode(strongReference);

return SizePolicy::GetBucket(hashCode, size);

}

void AddEntry(RecyclerWeakReferenceBase\* entry, RecyclerWeakReferenceBase\*\* bucket)

{

RecyclerWeakReferenceBase\* first = (\*bucket);

entry->next = first;

(\*bucket) = entry;

}

void Resize(int newSize)

{

#if DEBUG

uint copiedEntries = 0;

#endif

RecyclerWeakReferenceBase\*\* newBuckets = AllocatorNewArrayZ(HeapAllocator, allocator, RecyclerWeakReferenceBase\*, newSize);

for (uint i=0; i < size; i++)

{

RecyclerWeakReferenceBase\* current = buckets[i];

while (current != nullptr)

{

int targetBucket = HashKeyToBucket(current->strongRef, newSize);

RecyclerWeakReferenceBase\* next = current->next; // Cache the next pointer

AddEntry(current, &newBuckets[targetBucket]);

#if DEBUG

copiedEntries++;

#endif

current = next;

}

}

AllocatorDeleteArray(HeapAllocator, allocator, size, buckets);

size = newSize;

buckets = newBuckets;

#if DEBUG

Assert(this->count == copiedEntries);

#endif

}

RecyclerWeakReferenceBase\* Create(char\* strongReference, uint targetBucket, Recycler \* recycler)

{

Assert(HashKeyToBucket(strongReference, size) == targetBucket);

Assert(!FindEntry(strongReference, targetBucket));

if (IsDenserThan<MaxAverageChainLength>())

{

#ifdef RECYCLER\_TRACE\_WEAKREF

Output::Print(L"Count is %d\n", this->count);

#endif

Resize(SizePolicy::GetSize(size\*2));

// After resize - we will need to recalculate the bucket

targetBucket = HashKeyToBucket(strongReference, size);

}

RecyclerWeakReferenceBase\* entry;

entry = AllocatorNewBase(Recycler, recycler, AllocWeakReferenceEntry, RecyclerWeakReferenceBase);

entry->strongRef = strongReference;

entry->strongRefHeapBlock = recycler->FindHeapBlock(strongReference);

Assert(entry->strongRefHeapBlock != nullptr);

HeapBlock \* weakRefHeapBlock = recycler->FindHeapBlock(entry);

Assert(!weakRefHeapBlock->IsLargeHeapBlock());

entry->weakRefHeapBlock = (SmallHeapBlock \*)weakRefHeapBlock;

#ifdef RECYCLER\_TRACE\_WEAKREF

Output::Print(L"Add 0x%08x to bucket %d\n", entry, targetBucket);

#endif

AddEntry(entry, &buckets[targetBucket]);

count++;

#if DBG

entry->typeInfo = nullptr;

#if defined(TRACK\_ALLOC) && defined(PERF\_COUNTERS)

entry->counter = nullptr;

#endif

#endif

return entry;

}

};

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

// Initialization order

// AB AutoSystemInfo

// AD PerfCounter

// AE PerfCounterSet

// AM Output/Configuration

// AN MemProtectHeap

// AP DbgHelpSymbolManager

// AQ CFGLogger

// AR LeakReport

// AS JavascriptDispatch/RecyclerObjectDumper

// AT HeapAllocator/RecyclerHeuristic

// AU RecyclerWriteBarrierManager

#pragma warning(disable:4075) // initializers put in unrecognized initialization area on purpose

#pragma init\_seg(".CRT$XCAU")

#ifdef RECYCLER\_WRITE\_BARRIER

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

#ifdef \_M\_X64\_OR\_ARM64

X64WriteBarrierCardTableManager RecyclerWriteBarrierManager::x64CardTableManager;

X64WriteBarrierCardTableManager::CommitedSectionBitVector X64WriteBarrierCardTableManager::commitedSections(&HeapAllocator::Instance);

BYTE\* RecyclerWriteBarrierManager::cardTable = RecyclerWriteBarrierManager::x64CardTableManager.Initialize();

#else

// Each byte in the card table covers 4096 bytes so the range covered by the table is 4GB

BYTE RecyclerWriteBarrierManager::cardTable[1 \* 1024 \* 1024];

#endif

#else

// Each \*bit\* in the card table covers 128 bytes. So each DWORD covers 4096 bytes and therefore the cardTable covers 4GB

DWORD RecyclerWriteBarrierManager::cardTable[1 \* 1024 \* 1024];

#endif

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

#ifdef \_M\_X64\_OR\_ARM64

bool

X64WriteBarrierCardTableManager::OnThreadInit()

{

// We page in the card table sections for the current threads stack reservation

// So any writes to stack allocated vars can also have the write barrier set

NT\_TIB\* teb = (NT\_TIB\*) ::NtCurrentTeb();

char\* stackBase = (char\*) teb->StackBase;

char\* stackEnd = (char\*) teb->StackLimit;

size\_t numPages = (stackBase - stackEnd) / AutoSystemInfo::PageSize;

// stackEnd is the lower boundary

return OnSegmentAlloc(stackEnd, numPages);

}

bool

X64WriteBarrierCardTableManager::OnSegmentAlloc(\_In\_ char\* segmentAddress, size\_t numPages)

{

Assert(\_cardTable);

SetCommitState(OnSegmentAlloc);

if (segmentAddress >= AutoSystemInfo::Data.lpMaximumApplicationAddress)

{

Assert(false); // How did this happen?

SetCommitState(FailedMaxAddressExceeded);

Js::Throw::FatalInternalError();

}

AutoCriticalSection critSec(&\_cardTableInitCriticalSection);

size\_t pageSize = AutoSystemInfo::PageSize;

// First, check if the pages for this segment have already been committed

// If they have, there is nothing for us to do here.

void\* segmentEndAddress = segmentAddress + (numPages \* pageSize);

void\* segmentLastWritableAddress = (char\*)segmentEndAddress - 1;

BVIndex sectionStartIndex = GetSectionIndex(segmentAddress);

BVIndex sectionLastIndex = GetSectionIndex(segmentLastWritableAddress);

#ifdef X64\_WB\_DIAG

this->\_lastSegmentAddress = segmentAddress;

this->\_lastSegmentNumPages = numPages;

this->\_lastSectionIndexStart = sectionStartIndex;

this->\_lastSectionIndexLast = sectionLastIndex;

#endif

bool needCommit = false;

for (BVIndex i = sectionStartIndex; i <= sectionLastIndex; i++)

{

if (!commitedSections.Test(i))

{

needCommit = true;

break;

}

}

if (!needCommit)

{

// The pages for this segment have already been committed.

// We don't need to do anything more, since write barriers can

// already be set for writes to this segment

return true;

}

SetCommitState(OnNeedCommit);

// There are uncommitted pages in this range. We'll commit the full range

// We might commit some pages that are already committed but that's okay

const uintptr\_t startIndex = RecyclerWriteBarrierManager::GetCardTableIndex(segmentAddress);

const uintptr\_t endIndex = RecyclerWriteBarrierManager::GetCardTableIndex(segmentEndAddress);

Assert(startIndex <= endIndex);

// Section Start is the card table's starting entry aligned \*down\* to the page boundary

// Section End is the card table's ending entry aligned \*up\* to the page boundary

BYTE\* sectionStart = (BYTE\*) (((uintptr\_t) &\_cardTable[startIndex]) & ~(pageSize - 1));

BYTE\* sectionEnd = (BYTE\*) Math::Align<uintptr\_t>((uintptr\_t)&\_cardTable[endIndex], pageSize);

size\_t commitSize = (sectionEnd - sectionStart);

#ifdef X64\_WB\_DIAG

\_lastSectionStart = sectionStart;

\_lastSectionEnd = sectionEnd;

#endif

Assert(commitSize > 0);

Assert(commitSize % pageSize == 0);

Assert(commitSize / pageSize == sectionLastIndex - sectionStartIndex + 1);

LPVOID ret = ::VirtualAlloc((LPVOID) sectionStart, commitSize, MEM\_COMMIT, PAGE\_READWRITE);

if (!ret)

{

// If this is the error that occurred while trying to commit the page, this likely means

// that the page we tried to commit is outside out reservation, which means that our reservation

// was too small. This can happen if Windows increases the maximum process address space size

// If this happens, X64WriteBarrierCardTableManager::Initialize will have to be updated

Assert(::GetLastError() != ERROR\_INVALID\_ADDRESS);

SetCommitState(FailedVirtualAlloc);

return false;

}

SetCommitState(OnSectionCommitted);

BVIndex sectionIndex = sectionStartIndex;

try

{

#ifdef EXCEPTION\_CHECK

AUTO\_NESTED\_HANDLED\_EXCEPTION\_TYPE(ExceptionType\_DisableCheck);

#endif

for (; sectionIndex <= sectionLastIndex; sectionIndex++)

{

commitedSections.Set(sectionIndex);

}

SetCommitState(OnCommitBitSet);

}

catch (Js::OutOfMemoryException)

{

SetCommitState(FailedCommitBitSet);

// We ran out of memory allocating a node for the sparse bit vector, so clean up

// and return false

// Since setting sectionIndex threw the exception, we don't clear it, we clear until the index before it

for (BVIndex i = sectionStartIndex; i < sectionIndex; i++)

{

BOOLEAN wasSet = commitedSections.TestAndClear(i);

Assert(wasSet == TRUE);

}

#pragma prefast(suppress:6250, "This method decommits memory")

BOOL result = ::VirtualFree((LPVOID)sectionStart, commitSize, MEM\_DECOMMIT);

Assert(result != 0);

return false;

}

return true;

}

bool

X64WriteBarrierCardTableManager::OnSegmentFree(\_In\_ char\* segmentAddress, size\_t numPages)

{

Assert(\_cardTable);

return true;

}

X64WriteBarrierCardTableManager::~X64WriteBarrierCardTableManager()

{

if (\_cardTable != nullptr)

{

BOOL fSuccess = ::VirtualFree(\_cardTable, 0, MEM\_RELEASE);

Assert(fSuccess == TRUE);

}

}

BVIndex

X64WriteBarrierCardTableManager::GetSectionIndex(void\* address)

{

size\_t pageSize = AutoSystemInfo::PageSize;

size\_t sectionSize = (pageSize \* pageSize);

BVIndex sectionIndex = (BVIndex)(((uintptr\_t)address) / sectionSize);

return sectionIndex;

}

BYTE \*

X64WriteBarrierCardTableManager::Initialize()

{

AutoCriticalSection critSec(&\_cardTableInitCriticalSection);

if (\_cardTable == nullptr)

{

// We have two sizes for the card table on 64 bit builds

// On Win8.1 and later, the process address space size is 128 TB, so we reserve 32 GB for the card table

// On Win7, the max address space size is 192 GB, so we reserve 48 MB for the card table.

// On Win8, reserving 32 GB is fine since reservations don't incur a cost. On Win7, the cost

// of a reservation can be approximated as 2KB per MB of reserved size. In our case, we take

// an overhead of 96KB for our card table.

const unsigned \_\_int64 maxUmProcessAddressSpace = (\_\_int64) AutoSystemInfo::Data.lpMaximumApplicationAddress;

\_cardTableNumEntries = Math::Align<size\_t>(maxUmProcessAddressSpace / AutoSystemInfo::PageSize, AutoSystemInfo::PageSize) /\* s\_writeBarrierPageSize \*/;

LPVOID cardTableSpace = ::VirtualAlloc(NULL, \_cardTableNumEntries, MEM\_RESERVE, PAGE\_READWRITE);

if (cardTableSpace == nullptr)

{

return false;

}

\_cardTable = (BYTE\*) cardTableSpace;

}

return \_cardTable;

}

bool

RecyclerWriteBarrierManager::OnThreadInit()

{

return x64CardTableManager.OnThreadInit();

}

bool

RecyclerWriteBarrierManager::OnSegmentAlloc(\_In\_ char\* segmentAddress, size\_t numPages)

{

return x64CardTableManager.OnSegmentAlloc(segmentAddress, numPages);

}

bool

RecyclerWriteBarrierManager::OnSegmentFree(\_In\_ char\* segmentAddress, size\_t numPages)

{

return x64CardTableManager.OnSegmentFree(segmentAddress, numPages);

}

#endif

#else

#error Not implemented for bit-array card table

#endif

void

RecyclerWriteBarrierManager::WriteBarrier(void \* address)

{

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

const uintptr\_t index = GetCardTableIndex(address);

cardTable[index] = 1;

#else

uint bitShift = (((uint)address) >> s\_BitArrayCardTableShift);

uint bitMask = 1 << bitShift;

const uint cardIndex = ((uint) address) / (s\_BytesPerCard);

cardTable[cardIndex] |= bitMask;

#endif

#if DBG\_DUMP

// Global to process, use global configuration here

if (PHASE\_VERBOSE\_TRACE1(Js::SWBPhase))

{

Output::Print(L"Writing to 0x%p (CIndex: %u)\n", address, index);

}

#endif

}

void

RecyclerWriteBarrierManager::WriteBarrier(void \* address, size\_t ptrCount)

{

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

uintptr\_t startIndex = GetCardTableIndex(address);

char \* endAddress = (char \*)Math::Align<INT\_PTR>((INT\_PTR)((char \*)address + sizeof(void \*) \* ptrCount), s\_WriteBarrierPageSize);

uintptr\_t endIndex = GetCardTableIndex(endAddress);

Assert(startIndex <= endIndex);

memset(cardTable + startIndex, 1, endIndex - startIndex);

GlobalSwbVerboseTrace(L"Writing to 0x%p (CIndex: %u-%u)\n", address, startIndex, endIndex);

#else

uint bitShift = (((uint)address) >> s\_BitArrayCardTableShift);

uint bitMask = 0xFFFFFFFF << bitShift;

uint cardIndex = ((uint)address) / s\_BytesPerCard);

char \* endAddress = (char \*)Math::Align((INT\_PTR)((char \*)address + sizeof(void \*) \* ptrCount), s\_BytesPerCardBit);

char \* alignedAddress = (char \*)Math::Align((INT\_PTR)address, s\_WriteBarrierPageSize);

if (alignedAddress > endAddress)

{

uint endAddressShift = (((uint)endAddress) >> s\_BitArrayCardTableShift);

uint endAddressBitMask = 0xFFFFFFFF << endAddressShift;

bitMask &= ~endAddressBitMask;

cardTable[cardIndex] |= bitMask;

return;

}

cardTable[cardIndex] |= bitMask;

size\_t remainingBytes = endAddress - alignedAddress;

size\_t fullMaskCount = remainingBytes / g\_WriteBarrierPageSize;

memset(&cardTable[cardIndex + 1], 0xFFFFFFFF, fullMaskCount \* sizeof(DWORD));

uint endAddressShift = (((uint)endAddress) >> s\_BitArrayCardTableShift);

uint endAddressBitMask = 0xFFFFFFFF << endAddressShift;

cardTable[cardIndex + 1 + fullMaskCount] |= ~endAddressBitMask;

#endif

}

uintptr\_t

RecyclerWriteBarrierManager::GetCardTableIndex(void \*address)

{

return ((uintptr\_t)address) / s\_BytesPerCard;

}

void

RecyclerWriteBarrierManager::ResetWriteBarrier(void \* address, size\_t pageCount)

{

uintptr\_t cardIndex = GetCardTableIndex(address);

if (pageCount == 1)

{

cardTable[cardIndex] = 0;

}

else

{

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

memset(&cardTable[cardIndex], 0, pageCount);

#else

memset(&cardTable[cardIndex], 0, sizeof(DWORD) \* pageCount);

#endif

}

#if DBG\_DUMP

// Global to process, use global configuration here

if (PHASE\_VERBOSE\_TRACE1(Js::SWBPhase))

{

Output::Print(L"Resetting %u pages at CIndex: %u\n", address, pageCount, cardIndex);

}

#endif

}

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

BYTE

#else

DWORD

#endif

RecyclerWriteBarrierManager::GetWriteBarrier(void \* address)

{

return cardTable[GetCardTableIndex(address)];

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

//#define RECYCLER\_WRITE\_BARRIER\_INLINE\_RECYCLER

#ifdef RECYCLER\_WRITE\_BARRIER

// Controls whether we're using a 128 byte granularity card table or a 4K granularity byte array to indicate that a range of memory is dirty

#define RECYCLER\_WRITE\_BARRIER\_BYTE

// Controls whether the JIT is software write barrier aware

// #define RECYCLER\_WRITE\_BARRIER\_JIT

// Controls whether we can allocate SWB memory or not

// Turning this on leaves the rest of the SWB infrastructure intact,

// it'll simply allocate SWB objects are regular objects

#define RECYCLER\_WRITE\_BARRIER\_ALLOC

#ifdef RECYCLER\_WRITE\_BARRIER\_ALLOC

// Controls which page allocator to allocate SWB objects in

// By default, they use the recycler page allocator

// The following switches allow the user to either use the thread (leaf) page allocator or an entirely new page allocator for SWB objects

//#define RECYCLER\_WRITE\_BARRIER\_ALLOC\_THREAD\_PAGE

#define RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE

#if defined(RECYCLER\_WRITE\_BARRIER\_ALLOC\_THREAD\_PAGE) && defined(RECYCLER\_WRITE\_BARRIER\_ALLOC\_SEPARATE\_PAGE)

#error Not supported

#endif

#endif // RECYCLER\_WRITE\_BARRIER\_ALLOC

#ifdef \_M\_X64\_OR\_ARM64

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

#define X64\_WB\_DIAG 1

class X64WriteBarrierCardTableManager

{

public:

X64WriteBarrierCardTableManager() :

\_cardTable(nullptr)

{

}

~X64WriteBarrierCardTableManager();

BYTE \* Initialize();

// Called when a script thread is initialized

bool OnThreadInit();

// Called when a page allocator segment is allocated

bool OnSegmentAlloc(\_In\_ char\* segmentAddress, size\_t numPages);

// Called when a page allocator segment is freed

bool OnSegmentFree(\_In\_ char\* segmentAddress, size\_t numPages);

// Get the card table for the 64 bit address space

BYTE \* GetAddressOfCardTable() { return \_cardTable; }

#if DBG

void AssertWriteToAddress(\_In\_ void\* address)

{

Assert(\_cardTable);

Assert(commitedSections.Test(GetSectionIndex(address)));

}

#endif

private:

BVIndex GetSectionIndex(void\* address);

// In our card table scheme, 4KB of memory is mapped to 1 byte indicating whether

// it's written to or not. A page in Windows is also 4KB. Therefore, a page's worth

// of memory in the card table will track the state of 16MB of memory, which is what

// we term as a section. A bit set in this bit vector tracks whether a page corresponding

// to said 16MB is committed in the card table.

typedef BVSparse<HeapAllocator> CommitedSectionBitVector;

BYTE\* \_cardTable;

size\_t \_cardTableNumEntries;

CriticalSection \_cardTableInitCriticalSection;

#ifdef X64\_WB\_DIAG

enum CommitState

{

CommitStateOnSegmentAlloc,

CommitStateOnNeedCommit,

CommitStateOnSectionCommitted,

CommitStateOnCommitBitSet,

CommitStateFailedMaxAddressExceeded,

CommitStateFailedVirtualAlloc,

CommitStateFailedCommitBitSet

};

char\* \_lastSegmentAddress;

BYTE\* \_lastSectionStart;

BYTE\* \_lastSectionEnd;

size\_t \_lastSegmentNumPages;

size\_t \_lastSectionIndexStart;

size\_t \_lastSectionIndexLast;

CommitState \_lastCommitState;

#endif

#ifdef X64\_WB\_DIAG

#define SetCommitState(state) this->\_lastCommitState = CommitState##state

#else

#define SetCommitState(state)

#endif

static CommitedSectionBitVector commitedSections;

};

#else

#error Not implemented

#endif

#endif

class RecyclerWriteBarrierManager

{

public:

static void WriteBarrier(void \* address);

static void WriteBarrier(void \* address, size\_t ptrCount);

// For JIT

static uintptr\_t GetCardTableIndex(void \* address);

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

#ifdef \_M\_X64\_OR\_ARM64

static BYTE \* GetAddressOfCardTable() { return x64CardTableManager.GetAddressOfCardTable(); }

#else

static BYTE \* GetAddressOfCardTable() { return cardTable; }

#endif

#else

static DWORD \* GetAddressOfCardTable() { return cardTable; }

#endif

// For GC

#ifdef \_M\_X64\_OR\_ARM64

static bool OnThreadInit();

static bool OnSegmentAlloc(\_In\_ char\* segment, size\_t pageCount);

static bool OnSegmentFree(\_In\_ char\* segment, size\_t pageCount);

#endif

static void ResetWriteBarrier(void \* address, size\_t pageCount);

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

static BYTE GetWriteBarrier(void \* address);

#else

static DWORD GetWriteBarrier(void \* address);

#endif

static size\_t const s\_WriteBarrierPageSize = 4096;

static uint const s\_BitArrayCardTableShift = 7;

static uint const s\_BytesPerCardBit = 1 << s\_BitArrayCardTableShift; // 128 = 1 << 7

static uint const s\_BytesPerCard = s\_BytesPerCardBit \* 32; // 4K = 1 << 12 = 128 << 5

private:

#ifdef RECYCLER\_WRITE\_BARRIER\_BYTE

#ifdef \_M\_X64\_OR\_ARM64

// On AMD64, we use a different scheme

// As of Windows 8.1, the process user-mode address space is 128TB

// We still use a write barrier page size of 4KB

// If we used a static card table, the address space needed for the array would be 32 GB

// To get around this, we instead reserve 32 GB of address space for the card table

// The page allocator will commit the relevant parts of the card table when

// needed. Since the card table is dynamically allocated at runtime, we need one additional

// indirection to look up the card.

static X64WriteBarrierCardTableManager x64CardTableManager;

static BYTE\* cardTable; // 1 byte per 4096

#else

static BYTE cardTable[1 \* 1024 \* 1024]; // 1 byte per 4096

#endif

#else

static DWORD cardTable[1 \* 1024 \* 1024]; // 128 bytes per bit, 4096 per DWORD

#endif

};

#ifdef RECYCLER\_TRACE

#define SwbTrace(flags, ...) \

if (flags.Trace.IsEnabled(Js::MemoryAllocationPhase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#define GlobalSwbVerboseTrace(...) \

if (Js::Configuration::Global.flags.Verbose && \

Js::Configuration::Global.flags.Trace.IsEnabled(Js::MemoryAllocationPhase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#define SwbVerboseTrace(flags, ...) \

if (flags.Verbose && \

flags.Trace.IsEnabled(Js::MemoryAllocationPhase)) \

{ \

Output::Print(\_\_VA\_ARGS\_\_); \

}

#else

#define SwbTrace(...)

#define SwbVerboseTrace(...)

#define GlobalSwbVerboseTrace(...)

#endif

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#ifndef TBlockTypeAttributes

#error Need to define block type attributes before including this file

#endif

template void SmallHeapBlockT<TBlockTypeAttributes>::ReleasePages<true>(Recycler \* recycler);

template void SmallHeapBlockT<TBlockTypeAttributes>::ReleasePages<false>(Recycler \* recycler);

template void SmallHeapBlockT<TBlockTypeAttributes>::BackgroundReleasePagesSweep<true>(Recycler\* recycler);

template void SmallHeapBlockT<TBlockTypeAttributes>::BackgroundReleasePagesSweep<false>(Recycler\* recycler);

template void SmallHeapBlockT<TBlockTypeAttributes>::ReleasePagesSweep<true>(Recycler \* recycler);

template void SmallHeapBlockT<TBlockTypeAttributes>::ReleasePagesSweep<false>(Recycler \* recycler);

template BOOL SmallHeapBlockT<TBlockTypeAttributes>::ReassignPages<true>(Recycler \* recycler);

template BOOL SmallHeapBlockT<TBlockTypeAttributes>::ReassignPages<false>(Recycler \* recycler);

template BOOL SmallHeapBlockT<TBlockTypeAttributes>::SetPage<true>(\_\_in\_ecount\_pagesize char \* baseAddress, PageSegment \* pageSegment, Recycler \* recycler);

template BOOL SmallHeapBlockT<TBlockTypeAttributes>::SetPage<false>(\_\_in\_ecount\_pagesize char \* baseAddress, PageSegment \* pageSegment, Recycler \* recycler);

template const uint SmallHeapBlockT<TBlockTypeAttributes>::GetPageHeapModePageCount<true>() const;

template const uint SmallHeapBlockT<TBlockTypeAttributes>::GetPageHeapModePageCount<false>() const;

template SweepState SmallHeapBlockT<TBlockTypeAttributes>::Sweep<true>(RecyclerSweep& recyclerSweep, bool queuePendingSweep, bool allocable, ushort finalizeCount, bool hasPendingDispose);

template SweepState SmallHeapBlockT<TBlockTypeAttributes>::Sweep<false>(RecyclerSweep& recyclerSweep, bool queuePendingSweep, bool allocable, ushort finalizeCount, bool hasPendingDispose);

template SmallNormalHeapBlockT<TBlockTypeAttributes>\* HeapBlock::AsNormalBlock<TBlockTypeAttributes>();

template SmallLeafHeapBlockT<TBlockTypeAttributes>\* HeapBlock::AsLeafBlock<TBlockTypeAttributes>();

template SmallFinalizableHeapBlockT<TBlockTypeAttributes>\* HeapBlock::AsFinalizableBlock<TBlockTypeAttributes>();

#ifdef RECYCLER\_WRITE\_BARRIER

template SmallNormalWithBarrierHeapBlockT<TBlockTypeAttributes>\* HeapBlock::AsNormalWriteBarrierBlock<TBlockTypeAttributes>();

template SmallFinalizableWithBarrierHeapBlockT<TBlockTypeAttributes>\* HeapBlock::AsFinalizableWriteBarrierBlock<TBlockTypeAttributes>();

#endif

template bool SmallHeapBlockT<TBlockTypeAttributes>::FindHeapObjectImpl<SmallLeafHeapBlockT<TBlockTypeAttributes>>(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject);

template bool SmallHeapBlockT<TBlockTypeAttributes>::FindHeapObjectImpl<SmallNormalHeapBlockT<TBlockTypeAttributes>>(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject);

template bool SmallHeapBlockT<TBlockTypeAttributes>::FindHeapObjectImpl<SmallFinalizableHeapBlockT<TBlockTypeAttributes>>(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject);

#ifdef RECYCLER\_WRITE\_BARRIER

template bool SmallHeapBlockT<TBlockTypeAttributes>::FindHeapObjectImpl<SmallNormalWithBarrierHeapBlockT<TBlockTypeAttributes>>(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject);

template bool SmallHeapBlockT<TBlockTypeAttributes>::FindHeapObjectImpl<SmallFinalizableWithBarrierHeapBlockT<TBlockTypeAttributes>>(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject);

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template bool SmallHeapBlockT<TBlockTypeAttributes>::GetFreeObjectListOnAllocatorImpl<SmallNormalHeapBlockT<TBlockTypeAttributes>>(FreeObject \*\* freeObjectList);

template bool SmallHeapBlockT<TBlockTypeAttributes>::GetFreeObjectListOnAllocatorImpl<SmallLeafHeapBlockT<TBlockTypeAttributes>>(FreeObject \*\* freeObjectList);

template bool SmallHeapBlockT<TBlockTypeAttributes>::GetFreeObjectListOnAllocatorImpl<SmallFinalizableHeapBlockT<TBlockTypeAttributes>>(FreeObject \*\* freeObjectList);

#ifdef RECYCLER\_WRITE\_BARRIER

template bool SmallHeapBlockT<TBlockTypeAttributes>::GetFreeObjectListOnAllocatorImpl<SmallNormalWithBarrierHeapBlockT<TBlockTypeAttributes>>(FreeObject \*\* freeObjectList);

template bool SmallHeapBlockT<TBlockTypeAttributes>::GetFreeObjectListOnAllocatorImpl<SmallFinalizableWithBarrierHeapBlockT<TBlockTypeAttributes>>(FreeObject \*\* freeObjectList);

#endif

#endif

// template const SmallHeapBlockT<TBlockTypeAttributes>::SmallHeapBlockBitVector \* HeapInfo::ValidPointersMap<TBlockTypeAttributes>::GetInvalidBitVector(uint index) const;

// Explicit instantiate all the sweep mode

template void SmallHeapBlockT<TBlockTypeAttributes>::SweepObjects<true, SweepMode\_InThread>(Recycler \* recycler);

template void SmallHeapBlockT<TBlockTypeAttributes>::SweepObjects<false, SweepMode\_InThread>(Recycler \* recycler);

#ifdef CONCURRENT\_GC\_ENABLED

template <>

template <>

void

SmallHeapBlockT<TBlockTypeAttributes>::SweepObject<SweepMode\_Concurrent>(Recycler \* recycler, uint i, void \* addr)

{

AssertMsg(!(ObjectInfo(i) & FinalizeBit), "Finalize object should not be concurrent swept");

EnqueueProcessedObject(&freeObjectList, addr, i);

}

// Explicit instantiate all the sweep mode

template void SmallHeapBlockT<TBlockTypeAttributes>::SweepObjects<false, SweepMode\_Concurrent>(Recycler \* recycler);

#ifdef PARTIAL\_GC\_ENABLED

template <>

template <>

void

SmallHeapBlockT<TBlockTypeAttributes>::SweepObject<SweepMode\_ConcurrentPartial>(Recycler \* recycler, uint i, void \* addr)

{

Assert(recycler->inPartialCollectMode);

AssertMsg(!this->IsLeafBlock(), "Leaf pages should not do partial sweep");

AssertMsg(!(ObjectInfo(i) & FinalizeBit), "Finalize object should not be concurrent swept");

// This is a partial swept block; i.e. we're not reusing it.

// Just leave the object as-is; we will collect it in a future Sweep.

// However, we do clear out the ObjectInfo

// this keeps us from getting confused in certain situations, e.g. function enumeration for the debugger.

ObjectInfo(i) = 0;

}

// Explicit instantiate all the sweep mode

template void SmallHeapBlockT<TBlockTypeAttributes>::SweepObjects<false, SweepMode\_ConcurrentPartial>(Recycler \* recycler);

#endif

#endif

template <>

template <>

void

SmallHeapBlockT<TBlockTypeAttributes>::SweepObject<SweepMode\_InThread>(Recycler \* recycler, uint i, void \* addr)

{

if (ObjectInfo(i) & FinalizeBit)

{

Assert(this->IsAnyFinalizableBlock());

#ifdef CONCURRENT\_GC\_ENABLED

Assert(!recycler->IsConcurrentExecutingState());

#endif

// Call prepare finalize to do clean up that needs to be done immediately

// (e.g. Clear the ITrackable alias reference, so it can't be revived during

// other finalizers or concurrent sweep)

((FinalizableObject \*)addr)->Finalize(false);

// Set ObjectInfo to indicate a pending dispose object

ObjectInfo(i) = PendingDisposeObjectBits;

this->AsFinalizableBlock<TBlockTypeAttributes>()->AddPendingDisposeObject();

#ifdef RECYCLER\_FINALIZE\_CHECK

recycler->autoHeap.pendingDisposableObjectCount++;

#endif

}

else

{

EnqueueProcessedObject(&freeObjectList, addr, i);

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#ifdef RECYCLER\_WRITE\_BARRIER

template <class TBlockAttributes>

SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes>\*

SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes>::New(HeapBucketT<SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes>> \* bucket)

{

CompileAssert(TBlockAttributes::MaxObjectSize <= USHRT\_MAX);

Assert(bucket->sizeCat <= TBlockAttributes::MaxObjectSize);

Assert((TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / bucket->sizeCat <= USHRT\_MAX);

ushort objectSize = (ushort)bucket->sizeCat;

ushort objectCount = (ushort)(TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / objectSize;

return NoMemProtectHeapNewNoThrowPlusPrefixZ(GetAllocPlusSize(objectCount), SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes>, bucket, objectSize, objectCount);

}

template <class TBlockAttributes>

void

SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes>::Delete(SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes>\* heapBlock)

{

Assert(heapBlock->IsAnyFinalizableBlock());

Assert(heapBlock->IsWithBarrier());

NoMemProtectHeapDeletePlusPrefix(GetAllocPlusSize(heapBlock->objectCount), heapBlock);

}

#endif

template <class TBlockAttributes>

SmallFinalizableHeapBlockT<TBlockAttributes> \*

SmallFinalizableHeapBlockT<TBlockAttributes>::New(HeapBucketT<SmallFinalizableHeapBlockT<TBlockAttributes>> \* bucket)

{

CompileAssert(TBlockAttributes::MaxObjectSize <= USHRT\_MAX);

Assert(bucket->sizeCat <= TBlockAttributes::MaxObjectSize);

Assert((TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / bucket->sizeCat <= USHRT\_MAX);

ushort objectSize = (ushort)bucket->sizeCat;

ushort objectCount = (ushort)(TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / objectSize;

return NoMemProtectHeapNewNoThrowPlusPrefixZ(GetAllocPlusSize(objectCount), SmallFinalizableHeapBlockT<TBlockAttributes>, bucket, objectSize, objectCount);

}

template <class TBlockAttributes>

void

SmallFinalizableHeapBlockT<TBlockAttributes>::Delete(SmallFinalizableHeapBlockT<TBlockAttributes> \* heapBlock)

{

Assert(heapBlock->IsFinalizableBlock());

NoMemProtectHeapDeletePlusPrefix(GetAllocPlusSize(heapBlock->objectCount), heapBlock);

}

template <class SmallAllocationBlockAttributes>

SmallFinalizableHeapBlockT<SmallAllocationBlockAttributes>::SmallFinalizableHeapBlockT(HeapBucketT<SmallFinalizableHeapBlockT<SmallAllocationBlockAttributes>> \* bucket, ushort objectSize, ushort objectCount)

: SmallNormalHeapBlockT(bucket, objectSize, objectCount, SmallFinalizableBlockType)

{

// We used AllocZ

Assert(this->finalizeCount == 0);

Assert(this->pendingDisposeCount == 0);

Assert(this->disposedObjectList == nullptr);

Assert(this->disposedObjectListTail == nullptr);

Assert(!this->isPendingDispose);

}

template <>

SmallFinalizableHeapBlockT<MediumAllocationBlockAttributes>::SmallFinalizableHeapBlockT(HeapBucketT<SmallFinalizableHeapBlockT<MediumAllocationBlockAttributes>> \* bucket, ushort objectSize, ushort objectCount)

: SmallNormalHeapBlockT(bucket, objectSize, objectCount, MediumFinalizableBlockType)

{

// We used AllocZ

Assert(this->finalizeCount == 0);

Assert(this->pendingDisposeCount == 0);

Assert(this->disposedObjectList == nullptr);

Assert(this->disposedObjectListTail == nullptr);

Assert(!this->isPendingDispose);

}

#ifdef RECYCLER\_WRITE\_BARRIER

template <class TBlockAttributes>

SmallFinalizableHeapBlockT<TBlockAttributes>::SmallFinalizableHeapBlockT(HeapBucketT<SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes>> \* bucket, ushort objectSize, ushort objectCount, HeapBlockType blockType)

: SmallNormalHeapBlockT(bucket, objectSize, objectCount, blockType)

{

// We used AllocZ

Assert(this->finalizeCount == 0);

Assert(this->pendingDisposeCount == 0);

Assert(this->disposedObjectList == nullptr);

Assert(this->disposedObjectListTail == nullptr);

Assert(!this->isPendingDispose);

}

#endif

template <class TBlockAttributes>

void

SmallFinalizableHeapBlockT<TBlockAttributes>::SetAttributes(void \* address, unsigned char attributes)

{

Assert((attributes & FinalizeBit) != 0);

\_\_super::SetAttributes(address, attributes);

finalizeCount++;

#ifdef RECYCLER\_FINALIZE\_CHECK

HeapInfo \* heapInfo = this->heapBucket->heapInfo;

heapInfo->liveFinalizableObjectCount++;

heapInfo->newFinalizableObjectCount++;

#endif

}

template <class TBlockAttributes>

\_\_declspec(noinline)

void

SmallFinalizableHeapBlockT<TBlockAttributes>::ProcessMarkedObject(void\* objectAddress, MarkContext \* markContext)

{

ushort objectIndex = GetAddressIndex(objectAddress);

if (objectIndex == SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit)

{

// Not a valid offset within the block. No further processing necessary.

return;

}

unsigned char \* attributes = &ObjectInfo(objectIndex);

if (!UpdateAttributesOfMarkedObjects(markContext, objectAddress, objectSize, \*attributes,

[&](unsigned char \_attributes) { \*attributes = \_attributes; }))

{

// Couldn't mark children- bail out and come back later

this->SetNeedOOMRescan(markContext->GetRecycler());

}

}

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

// static

template <class TBlockAttributes>

bool

SmallFinalizableHeapBlockT<TBlockAttributes>::CanRescanFullBlock()

{

// Finalizable block need to rescan object one at a time.

return false;

}

// static

template <class TBlockAttributes>

bool

SmallFinalizableHeapBlockT<TBlockAttributes>::RescanObject(SmallFinalizableHeapBlockT<TBlockAttributes> \* block, \_\_in\_ecount(localObjectSize) char \* objectAddress, uint localObjectSize,

uint objectIndex, Recycler \* recycler)

{

unsigned char const attributes = block->ObjectInfo(objectIndex);

Assert(block->IsAnyFinalizableBlock());

if ((attributes & LeafBit) == 0)

{

Assert(block->GetAddressIndex(objectAddress) != SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit);

if (!recycler->AddMark(objectAddress, localObjectSize))

{

// Failed to add to the mark stack due to OOM.

return false;

}

RECYCLER\_STATS\_INC(recycler, markData.rescanObjectCount);

RECYCLER\_STATS\_ADD(recycler, markData.rescanObjectByteCount, localObjectSize);

}

// Since we mark through unallocated objects, we might have marked an object before it

// is allocated as an tracked object. The object will not be queue up in the

// tracked object list, and NewTrackBit will still be on. Queue it up now.

// NewTrackBit will also be on for tracked object that we weren't able to queue

// because of OOM. In those case, the page is forced to be rescan, and we will

// try to process those again here.

if ((attributes & (TrackBit | NewTrackBit)) == (TrackBit | NewTrackBit))

{

if (!block->RescanTrackedObject((FinalizableObject\*) objectAddress, objectIndex, recycler))

{

// Failed to add to the mark stack due to OOM.

return false;

}

}

#ifdef RECYCLER\_STATS

else if (attributes & FinalizeBit)

{

// Concurrent thread mark the object before the attribute is set and missed the finalize count

// For finalized object, we will always write a dummy vtable before returning to the call,

// so the page will always need to be rescanned, and we can count those here.

// NewFinalizeBit is cleared if the background thread has already counted the object.

// So if it is still set here, we need to count it

RECYCLER\_STATS\_INC\_IF(attributes & NewFinalizeBit, recycler, finalizeCount);

block->ObjectInfo(objectIndex) &= ~NewFinalizeBit;

}

#endif

return true;

}

template <class TBlockAttributes>

bool

SmallFinalizableHeapBlockT<TBlockAttributes>::RescanTrackedObject(FinalizableObject \* object, uint objectIndex, Recycler \* recycler)

{

RecyclerVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Marking 0x%08x during rescan\n", object);

#ifdef PARTIAL\_GC\_ENABLED

if (recycler->inPartialCollectMode)

{

Assert(!recycler->DoQueueTrackedObject());

}

else

#endif

{

Assert(recycler->DoQueueTrackedObject());

if (!recycler->QueueTrackedObject(object))

{

// Failed to add to track stack due to OOM.

return false;

}

}

RECYCLER\_STATS\_INC(recycler, trackCount);

RECYCLER\_STATS\_INC\_IF(ObjectInfo(objectIndex) & FinalizeBit, recycler, finalizeCount);

// We have processed this object as tracked, we can clear the NewTrackBit

ObjectInfo(objectIndex) &= ~NewTrackBit;

return true;

}

#endif

template SweepState SmallFinalizableHeapBlockT<SmallAllocationBlockAttributes>::Sweep<true>(RecyclerSweep& recyclerSweep, bool queuePendingSweep, bool allocable);

template SweepState SmallFinalizableHeapBlockT<SmallAllocationBlockAttributes>::Sweep<false>(RecyclerSweep& recyclerSweep, bool queuePendingSweep, bool allocable);

template SweepState SmallFinalizableHeapBlockT<MediumAllocationBlockAttributes>::Sweep<true>(RecyclerSweep& recyclerSweep, bool queuePendingSweep, bool allocable);

template SweepState SmallFinalizableHeapBlockT<MediumAllocationBlockAttributes>::Sweep<false>(RecyclerSweep& recyclerSweep, bool queuePendingSweep, bool allocable);

template <class TBlockAttributes>

template<bool pageheap>

SweepState

SmallFinalizableHeapBlockT<TBlockAttributes>::Sweep(RecyclerSweep& recyclerSweep, bool queuePendingSweep, bool allocable)

{

Assert(!recyclerSweep.IsBackground());

Assert(!queuePendingSweep);

// If there are finalizable objects in this heap block, they need to be swept

// in-thread and not in the concurrent thread, so don't queue pending sweep

return SmallNormalHeapBlockT<TBlockAttributes>::Sweep<pageheap>(recyclerSweep, false, allocable, this->finalizeCount, HasAnyDisposeObjects());

}

template <class TBlockAttributes>

void

SmallFinalizableHeapBlockT<TBlockAttributes>::DisposeObjects()

{

Assert(this->isPendingDispose);

Assert(HasAnyDisposeObjects());

// PARTIALGC-CONSIDER: page with finalizable/disposable object will always be modified

// because calling dispose probably will modify object itself, and it may call other

// script that might touch the page as well. We can't distinguish between these two kind

// of write to the page.

//

// Possible mitigation include:

// - allocating finalizable/disposable object on separate pages

// - some of the object only need finalize, but not dispose. mark them separately

//

// For now, we always touch the page by zeroing out disposed object which should be moved as well.

ForEachPendingDisposeObject([&] (uint index) {

void \* objectAddress = address + (objectSize \* index);

// Dispose the object.

// Note that Dispose can cause reentrancy, which can cause allocation, which can cause collection.

// The object we're disposing is still considered PendingDispose until the Dispose call completes.

// So in case we call CheckFreeBitVector or similar, we should still see correct state re this object.

((FinalizableObject \*)objectAddress)->Dispose(false);

Assert(finalizeCount != 0);

finalizeCount--;

Assert(pendingDisposeCount != 0);

pendingDisposeCount--;

// Properly enqueue the processed object

// This will also clear the ObjectInfo bits so it's not marked as PendingDispose anymore

EnqueueProcessedObject(&disposedObjectList, &disposedObjectListTail, objectAddress, index);

RECYCLER\_STATS\_INC(this->heapBucket->heapInfo->recycler, finalizeSweepCount);

#ifdef RECYCLER\_FINALIZE\_CHECK

this->heapBucket->heapInfo->liveFinalizableObjectCount--;

this->heapBucket->heapInfo->pendingDisposableObjectCount--;

#endif

});

// Dispose could have re-entered and caused new pending dispose objects on this block.

// If so, recycler->hasDisposableObject will have been set again, and we will do another

// round of Dispose to actually dispose these objects.

Assert(this->pendingDisposeCount == 0 || this->heapBucket->heapInfo->recycler->hasDisposableObject);

}

template <class TBlockAttributes>

void

SmallFinalizableHeapBlockT<TBlockAttributes>::TransferDisposedObjects()

{

// CONCURRENT-TODO: we don't allocate on pending disposed blocks during concurrent sweep or disable dispose

// So the free bit vector must be valid

Assert(this->IsFreeBitsValid());

Assert(this->isPendingDispose);

Assert(this->pendingDisposeCount == 0);

DebugOnly(this->isPendingDispose = false);

TransferProcessedObjects(this->disposedObjectList, this->disposedObjectListTail);

this->disposedObjectList = nullptr;

this->disposedObjectListTail = nullptr;

// We already updated the bit vector on TransferSweptObjects

// So just update the free object head.

this->lastFreeObjectHead = this->freeObjectList;

RECYCLER\_SLOW\_CHECK(CheckFreeBitVector(true));

}

template <class TBlockAttributes>

ushort

SmallFinalizableHeapBlockT<TBlockAttributes>::AddDisposedObjectFreeBitVector(SmallHeapBlockBitVector \* free)

{

// all the finalized object are considered freed, but not allocable yet

ushort freeCount = 0;

FreeObject \* freeObject = this->disposedObjectList;

if (freeObject != nullptr)

{

while (true)

{

uint bitIndex = this->GetAddressBitIndex(freeObject);

Assert(IsValidBitIndex(bitIndex));

// not allocable yet

Assert(!this->GetDebugFreeBitVector()->Test(bitIndex));

// but in the free list to mark can skip scanning the object

free->Set(bitIndex);

freeCount++;

if (freeObject == this->disposedObjectListTail)

{

break;

}

freeObject = freeObject->GetNext();

}

}

return freeCount;

}

template <class TBlockAttributes>

void

SmallFinalizableHeapBlockT<TBlockAttributes>::FinalizeAllObjects()

{

if (this->finalizeCount != 0)

{

DebugOnly(uint processedCount = 0);

ForEachAllocatedObject(FinalizeBit, [&](uint index, void \* objectAddress)

{

FinalizableObject \* finalizableObject = ((FinalizableObject \*)objectAddress);

finalizableObject->Finalize(true);

finalizableObject->Dispose(true);

#ifdef RECYCLER\_FINALIZE\_CHECK

this->heapBucket->heapInfo->liveFinalizableObjectCount --;

#endif

DebugOnly(processedCount++);

});

ForEachPendingDisposeObject([&] (uint index) {

void \* objectAddress = address + (objectSize \* index);

((FinalizableObject \*)objectAddress)->Dispose(true);

#ifdef RECYCLER\_FINALIZE\_CHECK

this->heapBucket->heapInfo->liveFinalizableObjectCount--;

this->heapBucket->heapInfo->pendingDisposableObjectCount--;

#endif

DebugOnly(processedCount++);

});

Assert(this->finalizeCount == processedCount);

}

}

#if DBG

template <class TBlockAttributes>

void

SmallFinalizableHeapBlockT<TBlockAttributes>::Init(ushort objectSize, ushort objectCount)

{

Assert(this->disposedObjectList == nullptr);

Assert(this->disposedObjectListTail == nullptr);

Assert(this->finalizeCount == 0);

Assert(this->pendingDisposeCount == 0);

\_\_super::Init(objectSize, objectCount);

}

#ifdef PARTIAL\_GC\_ENABLED

template <class TBlockAttributes>

void

SmallFinalizableHeapBlockT<TBlockAttributes>::FinishPartialCollect()

{

Assert(this->disposedObjectList == nullptr);

Assert(this->disposedObjectListTail == nullptr);

\_\_super::FinishPartialCollect();

}

#endif

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <class TBlockAttributes>

uint

SmallFinalizableHeapBlockT<TBlockAttributes>::CheckDisposedObjectFreeBitVector()

{

uint verifyFreeCount = 0;

// all the finalized object are considered freed, but not allocable yet

FreeObject \*freeObject = this->disposedObjectList;

if (freeObject != nullptr)

{

SmallHeapBlockBitVector \* free = this->GetFreeBitVector();

while (true)

{

uint bitIndex = this->GetAddressBitIndex(freeObject);

Assert(IsValidBitIndex(bitIndex));

Assert(!this->GetDebugFreeBitVector()->Test(bitIndex));

Assert(free->Test(bitIndex));

verifyFreeCount++;

if (freeObject == this->disposedObjectListTail)

{

break;

}

freeObject = freeObject->GetNext();

}

}

return verifyFreeCount;

}

template <class TBlockAttributes>

bool

SmallFinalizableHeapBlockT<TBlockAttributes>::GetFreeObjectListOnAllocator(FreeObject \*\* freeObjectList)

{

return GetFreeObjectListOnAllocatorImpl<SmallFinalizableHeapBlockT<TBlockAttributes>>(freeObjectList);

}

#endif

template class SmallFinalizableHeapBlockT<SmallAllocationBlockAttributes>;

template class SmallFinalizableHeapBlockT<MediumAllocationBlockAttributes>;

#ifdef RECYCLER\_WRITE\_BARRIER

template class SmallFinalizableWithBarrierHeapBlockT<SmallAllocationBlockAttributes>;

template class SmallFinalizableWithBarrierHeapBlockT<MediumAllocationBlockAttributes>;

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

namespace Memory

{

template <class TBlockAttributes> class SmallFinalizableHeapBucketT;

#ifdef RECYCLER\_WRITE\_BARRIER

template <class TBlockAttributes> class SmallFinalizableWithBarrierHeapBlockT;

#endif

template <class TBlockAttributes>

class SmallFinalizableHeapBlockT : public SmallNormalHeapBlockT<TBlockAttributes>

{

friend class HeapBucketT<SmallFinalizableHeapBlockT>;

public:

typedef TBlockAttributes HeapBlockAttributes;

static const ObjectInfoBits RequiredAttributes = FinalizeBit;

static SmallFinalizableHeapBlockT \* New(HeapBucketT<SmallFinalizableHeapBlockT> \* bucket);

static void Delete(SmallFinalizableHeapBlockT \* block);

SmallFinalizableHeapBlockT \* GetNextBlock() const { return SmallHeapBlockT<TBlockAttributes>::GetNextBlock()->AsFinalizableBlock<TBlockAttributes>(); }

void SetNextBlock(SmallFinalizableHeapBlockT \* next) { \_\_super::SetNextBlock(next); }

void ProcessMarkedObject(void\* candidate, MarkContext \* markContext);

void SetAttributes(void \* address, unsigned char attributes);

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

static bool CanRescanFullBlock();

static bool RescanObject(SmallFinalizableHeapBlockT<TBlockAttributes> \* block, \_\_in\_ecount(localObjectSize) char \* objectAddress, uint localObjectSize, uint objectIndex, Recycler \* recycler);

bool RescanTrackedObject(FinalizableObject \* object, uint objectIndex, Recycler \* recycler);

#endif

template<bool pageheap>

SweepState Sweep(RecyclerSweep& sweepeData, bool queuePendingSweep, bool allocable);

void DisposeObjects();

void TransferDisposedObjects();

bool HasPendingDisposeObjects() const

{

return (this->pendingDisposeCount != 0);

}

void AddPendingDisposeObject()

{

this->pendingDisposeCount++;

Assert(this->pendingDisposeCount <= this->objectCount);

}

bool HasDisposedObjects() const { return this->disposedObjectList != nullptr; }

bool HasAnyDisposeObjects() const { return this->HasPendingDisposeObjects() || this->HasDisposedObjects(); }

template <typename Fn>

void ForEachPendingDisposeObject(Fn fn)

{

if (this->HasPendingDisposeObjects())

{

for (uint i = 0; i < this->objectCount; i++)

{

if ((ObjectInfo(i) & PendingDisposeBit) != 0)

{

// When pending dispose, exactly the PendingDisposeBits should be set

Assert(ObjectInfo(i) == PendingDisposeObjectBits);

fn(i);

}

}

}

else

{

#if DBG

for (uint i = 0; i < this->objectCount; i++)

{

Assert((ObjectInfo(i) & PendingDisposeBit) == 0);

}

#endif

}

}

ushort AddDisposedObjectFreeBitVector(SmallHeapBlockBitVector \* free);

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

uint CheckDisposedObjectFreeBitVector();

virtual bool GetFreeObjectListOnAllocator(FreeObject \*\* freeObjectList) override;

#endif

#if DBG

#ifdef PARTIAL\_GC\_ENABLED

void FinishPartialCollect();

#endif

bool IsPendingDispose() const { return isPendingDispose; }

void SetIsPendingDispose() { isPendingDispose = true; }

#endif

void FinalizeAllObjects();

#ifdef DUMP\_FRAGMENTATION\_STATS

ushort GetFinalizeCount() {

return finalizeCount;

}

#endif

virtual bool FindHeapObject(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject) override

{

return FindHeapObjectImpl<SmallFinalizableHeapBlockT>(objectAddress, recycler, flags, heapObject);

}

protected:

SmallFinalizableHeapBlockT(HeapBucketT<SmallFinalizableHeapBlockT> \* bucket, ushort objectSize, ushort objectCount);

#ifdef RECYCLER\_WRITE\_BARRIER

SmallFinalizableHeapBlockT(HeapBucketT<SmallFinalizableWithBarrierHeapBlockT<TBlockAttributes>> \* bucket, ushort objectSize, ushort objectCount, HeapBlockType blockType);

#endif

#if DBG

void Init(ushort objectSize, ushort objectCount);

bool isPendingDispose;

#endif

ushort finalizeCount;

ushort pendingDisposeCount;

FreeObject\* disposedObjectList;

FreeObject\* disposedObjectListTail;

friend class ScriptMemoryDumper;

#ifdef RECYCLER\_MEMORY\_VERIFY

friend void SmallHeapBlockT<TBlockAttributes>::Verify(bool pendingDispose);

#endif

};

#ifdef RECYCLER\_WRITE\_BARRIER

template <class TBlockAttributes>

class SmallFinalizableWithBarrierHeapBlockT : public SmallFinalizableHeapBlockT<TBlockAttributes>

{

friend class HeapBucketT<SmallFinalizableWithBarrierHeapBlockT>;

public:

typedef TBlockAttributes HeapBlockAttributes;

static const ObjectInfoBits RequiredAttributes = FinalizableWithBarrierBit;

static const bool IsLeafOnly = false;

static SmallFinalizableWithBarrierHeapBlockT \* New(HeapBucketT<SmallFinalizableWithBarrierHeapBlockT> \* bucket);

static void Delete(SmallFinalizableWithBarrierHeapBlockT \* block);

SmallFinalizableWithBarrierHeapBlockT \* GetNextBlock() const { return ((SmallHeapBlock\*) this)->GetNextBlock()->AsFinalizableWriteBarrierBlock<TBlockAttributes>(); }

virtual bool FindHeapObject(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject) override sealed

{

return FindHeapObjectImpl<SmallFinalizableWithBarrierHeapBlockT>(objectAddress, recycler, flags, heapObject);

}

protected:

SmallFinalizableWithBarrierHeapBlockT(HeapBucketT<SmallFinalizableWithBarrierHeapBlockT> \* bucket, ushort objectSize, ushort objectCount)

: SmallFinalizableHeapBlockT(bucket, objectSize, objectCount, TBlockAttributes::IsSmallBlock ? SmallFinalizableBlockWithBarrierType : MediumFinalizableBlockWithBarrierType)

{

}

};

#endif

extern template class SmallFinalizableHeapBlockT<SmallAllocationBlockAttributes>;

extern template class SmallFinalizableHeapBlockT<MediumAllocationBlockAttributes>;

typedef SmallFinalizableHeapBlockT<SmallAllocationBlockAttributes> SmallFinalizableHeapBlock;

typedef SmallFinalizableHeapBlockT<MediumAllocationBlockAttributes> MediumFinalizableHeapBlock;

#ifdef RECYCLER\_WRITE\_BARRIER

extern template class SmallFinalizableWithBarrierHeapBlockT<SmallAllocationBlockAttributes>;

extern template class SmallFinalizableWithBarrierHeapBlockT<MediumAllocationBlockAttributes>;

typedef SmallFinalizableWithBarrierHeapBlockT<SmallAllocationBlockAttributes> SmallFinalizableWithBarrierHeapBlock;

typedef SmallFinalizableWithBarrierHeapBlockT<MediumAllocationBlockAttributes> MediumFinalizableWithBarrierHeapBlock;

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

template <class TBlockType>

SmallFinalizableHeapBucketBaseT<TBlockType>::SmallFinalizableHeapBucketBaseT() :

pendingDisposeList(nullptr)

{

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

tempPendingDisposeList = nullptr;

#endif

}

template <class TBlockType>

SmallFinalizableHeapBucketBaseT<TBlockType>::~SmallFinalizableHeapBucketBaseT()

{

Assert(this->AllocatorsAreEmpty());

DeleteHeapBlockList(this->pendingDisposeList);

Assert(this->tempPendingDisposeList == nullptr);

}

template <class TBlockType>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::FinalizeAllObjects()

{

// Finalize all objects on shutdown.

// Clear allocators to update the information on the heapblock

// Walk through the allocated object and call finalize and dispose on them

this->ClearAllocators();

FinalizeHeapBlockList(this->pendingDisposeList);

#ifdef PARTIAL\_GC\_ENABLED

FinalizeHeapBlockList(this->partialHeapBlockList);

#endif

FinalizeHeapBlockList(this->heapBlockList);

FinalizeHeapBlockList(this->fullBlockList);

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

FinalizeHeapBlockList(this->partialSweptHeapBlockList);

#endif

}

template <class TBlockType>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::FinalizeHeapBlockList(TBlockType \* list)

{

HeapBlockList::ForEach(list, [](TBlockType \* heapBlock)

{

heapBlock->FinalizeAllObjects();

});

}

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

template <class TBlockType>

size\_t

SmallFinalizableHeapBucketBaseT<TBlockType>::GetNonEmptyHeapBlockCount(bool checkCount) const

{

size\_t currentHeapBlockCount = \_\_super::GetNonEmptyHeapBlockCount(false)

+ HeapBlockList::Count(pendingDisposeList)

+ HeapBlockList::Count(tempPendingDisposeList);

RECYCLER\_SLOW\_CHECK(Assert(!checkCount || heapBlockCount == currentHeapBlockCount));

return currentHeapBlockCount;

}

#endif

template <class TBlockType>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::ResetMarks(ResetMarkFlags flags)

{

\_\_super::ResetMarks(flags);

if ((flags & ResetMarkFlags\_ScanImplicitRoot) != 0)

{

HeapBlockList::ForEach(this->pendingDisposeList, [flags](TBlockType \* heapBlock)

{

heapBlock->MarkImplicitRoots();

});

}

}

#ifdef DUMP\_FRAGMENTATION\_STATS

template <class TBlockType>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::AggregateBucketStats(HeapBucketStats& stats)

{

\_\_super::AggregateBucketStats(stats);

HeapBlockList::ForEach(pendingDisposeList, [&stats](TBlockType\* heapBlock) {

heapBlock->AggregateBlockStats(stats);

});

}

#endif

template void SmallFinalizableHeapBucketBaseT<SmallFinalizableHeapBlock>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallFinalizableHeapBucketBaseT<SmallFinalizableHeapBlock>::Sweep<false>(RecyclerSweep& recyclerSweep);

template void SmallFinalizableHeapBucketBaseT<MediumFinalizableHeapBlock>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallFinalizableHeapBucketBaseT<MediumFinalizableHeapBlock>::Sweep<false>(RecyclerSweep& recyclerSweep);

#ifdef RECYCLER\_WRITE\_BARRIER

template void SmallFinalizableHeapBucketBaseT<SmallFinalizableWithBarrierHeapBlock>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallFinalizableHeapBucketBaseT<SmallFinalizableWithBarrierHeapBlock>::Sweep<false>(RecyclerSweep& recyclerSweep);

template void SmallFinalizableHeapBucketBaseT<MediumFinalizableWithBarrierHeapBlock>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallFinalizableHeapBucketBaseT<MediumFinalizableWithBarrierHeapBlock>::Sweep<false>(RecyclerSweep& recyclerSweep);

#endif

template<class TBlockType>

template<bool pageheap>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::Sweep(RecyclerSweep& recyclerSweep)

{

Assert(!recyclerSweep.IsBackground());

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

Assert(this->tempPendingDisposeList == nullptr);

this->tempPendingDisposeList = pendingDisposeList;

#endif

TBlockType \* currentDisposeList = pendingDisposeList;

this->pendingDisposeList = nullptr;

BaseT::SweepBucket<pageheap>(recyclerSweep, [=](RecyclerSweep& recyclerSweep)

{

#if DBG

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

recyclerSweep.SetupVerifyListConsistencyDataForSmallBlock(nullptr, false, true);

}

else if (TBlockType::HeapBlockAttributes::IsMediumBlock)

{

recyclerSweep.SetupVerifyListConsistencyDataForMediumBlock(nullptr, false, true);

}

else

{

Assert(false);

}

#endif

HeapBucketT<TBlockType>::SweepHeapBlockList<pageheap>(recyclerSweep, currentDisposeList, false);

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

Assert(this->tempPendingDisposeList == currentDisposeList);

this->tempPendingDisposeList = nullptr;

#endif

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount(recyclerSweep.IsBackground()));

});

}

template <class TBlockType>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::DisposeObjects()

{

HeapBlockList::ForEach(this->pendingDisposeList, [](TBlockType \* heapBlock)

{

Assert(heapBlock->HasAnyDisposeObjects());

heapBlock->DisposeObjects();

});

}

template <class TBlockType>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::TransferDisposedObjects()

{

Assert(!this->IsAllocationStopped());

TBlockType \* currentPendingDisposeList = this->pendingDisposeList;

if (currentPendingDisposeList != nullptr)

{

this->pendingDisposeList = nullptr;

HeapBlockList::ForEach(currentPendingDisposeList, [=](TBlockType \* heapBlock)

{

heapBlock->TransferDisposedObjects();

// in pageheap, we actually always have free object

Assert(heapBlock->HasFreeObject<false>());

});

#ifdef RECYCLER\_PAGE\_HEAP

if (this->IsPageHeapEnabled())

{

// In pageheap mode, we can't reuse the empty blocks since they are not close enough to the guard page

// treat it as full and ready to be released

HeapBlockList::Tail(currentPendingDisposeList)->SetNextBlock(this->fullBlockList);

this->fullBlockList = currentPendingDisposeList;

}

else

#endif

{

// For partial collect, dispose will modify the object, and we

// also touch the page by chaining the object through the free list

// might as well reuse the block for partial collect

this->AppendAllocableHeapBlockList(currentPendingDisposeList);

}

}

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount(false));

}

template <class TBlockType>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::EnumerateObjects(ObjectInfoBits infoBits, void(\*CallBackFunction)(void \* address, size\_t size))

{

\_\_super::EnumerateObjects(infoBits, CallBackFunction);

HeapBucket::EnumerateObjects(this->pendingDisposeList, infoBits, CallBackFunction);

}

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <class TBlockType>

size\_t

SmallFinalizableHeapBucketBaseT<TBlockType>::Check()

{

size\_t smallHeapBlockCount = \_\_super::Check(false) + HeapInfo::Check(false, true, this->pendingDisposeList);

Assert(this->heapBlockCount == smallHeapBlockCount);

return smallHeapBlockCount;

}

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

template <class TBlockType>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::Verify()

{

BaseT::Verify();

#if DBG

RecyclerVerifyListConsistencyData recyclerVerifyListConsistencyData;

if (TBlockType::HeapBlockAttributes::IsSmallBlock)

{

recyclerVerifyListConsistencyData.SetupVerifyListConsistencyDataForSmallBlock(nullptr, false, true);

}

else if (TBlockType::HeapBlockAttributes::IsMediumBlock)

{

recyclerVerifyListConsistencyData.SetupVerifyListConsistencyDataForMediumBlock(nullptr, false, true);

}

else

{

Assert(false);

}

HeapBlockList::ForEach(this->pendingDisposeList, [&recyclerVerifyListConsistencyData](TBlockType \* heapBlock)

{

DebugOnly(VerifyBlockConsistencyInList(heapBlock, recyclerVerifyListConsistencyData));

heapBlock->Verify(true);

});

#endif

}

#endif

#ifdef RECYCLER\_VERIFY\_MARK

template <class TBlockType>

void

SmallFinalizableHeapBucketBaseT<TBlockType>::VerifyMark()

{

\_\_super::VerifyMark();

HeapBlockList::ForEach(this->pendingDisposeList, [](TBlockType \* heapBlock)

{

Assert(heapBlock->HasAnyDisposeObjects());

heapBlock->VerifyMark();

});

}

#endif

template class SmallFinalizableHeapBucketBaseT<SmallFinalizableHeapBlock>;

#ifdef RECYCLER\_WRITE\_BARRIER

template class SmallFinalizableHeapBucketBaseT<SmallFinalizableWithBarrierHeapBlock>;

#endif

template class SmallFinalizableHeapBucketBaseT<MediumFinalizableHeapBlock>;

#ifdef RECYCLER\_WRITE\_BARRIER

template class SmallFinalizableHeapBucketBaseT<MediumFinalizableWithBarrierHeapBlock>;

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

namespace Memory

{

template <class THeapBlockType>

class SmallFinalizableHeapBucketBaseT : public SmallNormalHeapBucketBase<THeapBlockType>

{

typedef SmallNormalHeapBucketBase<THeapBlockType> BaseT;

public:

typedef typename THeapBlockType::HeapBlockAttributes TBlockAttributes;

SmallFinalizableHeapBucketBaseT();

~SmallFinalizableHeapBucketBaseT();

void DisposeObjects();

void TransferDisposedObjects();

void FinalizeAllObjects();

static void FinalizeHeapBlockList(THeapBlockType \* list);

#ifdef DUMP\_FRAGMENTATION\_STATS

void AggregateBucketStats(HeapBucketStats& stats);

#endif

protected:

void EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size));

friend class HeapBucket;

template <class TBlockAttributes>

friend class HeapBucketGroup;

void ResetMarks(ResetMarkFlags flags);

template<bool pageheap>

void Sweep(RecyclerSweep& recyclerSweep);

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t GetNonEmptyHeapBlockCount(bool checkCount) const;

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t Check();

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void Verify();

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void VerifyMark();

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

friend class ScriptMemoryDumper;

#endif

template <typename TBlockType>

friend class HeapBucketT;

protected:

THeapBlockType \* pendingDisposeList; // list of block that has finalizable object that needs to be disposed

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

THeapBlockType \* tempPendingDisposeList;

#endif

};

#define DeclareFinalizableHeapBucket(type) \

template <class TBlockAttributes> class Small##type##HeapBucketT : public SmallFinalizableHeapBucketBaseT<Small##type##HeapBlockT<TBlockAttributes> >{}; \

extern template class SmallFinalizableHeapBucketBaseT<Small##type##HeapBlockT<SmallAllocationBlockAttributes>>; \

extern template class SmallFinalizableHeapBucketBaseT<Small##type##HeapBlockT<MediumAllocationBlockAttributes>>;

DeclareFinalizableHeapBucket(Finalizable);

#ifdef RECYCLER\_WRITE\_BARRIER

DeclareFinalizableHeapBucket(FinalizableWithBarrier);

#endif

typedef SmallFinalizableHeapBucketT<MediumAllocationBlockAttributes> MediumFinalizableHeapBucket;

typedef SmallFinalizableHeapBucketT<SmallAllocationBlockAttributes> SmallFinalizableHeapBucket;

#ifdef RECYCLER\_WRITE\_BARRIER

typedef SmallFinalizableWithBarrierHeapBucketT<MediumAllocationBlockAttributes> MediumFinalizableWithBarrierHeapBucket;

typedef SmallFinalizableWithBarrierHeapBucketT<SmallAllocationBlockAttributes> SmallFinalizableWithBarrierHeapBucket;

#endif

// GC-TODO: Move this away

template <ObjectInfoBits attributes, class TBlockAttributes>

class SmallHeapBlockType

{

public:

CompileAssert(attributes & FinalizeBit);

typedef SmallFinalizableHeapBlockT<TBlockAttributes> BlockType;

typedef SmallFinalizableHeapBucketT<TBlockAttributes> BucketType;

};

template <>

class SmallHeapBlockType<LeafBit, SmallAllocationBlockAttributes>

{

public:

typedef SmallLeafHeapBlock BlockType;

typedef SmallLeafHeapBucketT<SmallAllocationBlockAttributes> BucketType;

};

template <>

class SmallHeapBlockType<NoBit, SmallAllocationBlockAttributes>

{

public:

typedef SmallNormalHeapBlock BlockType;

typedef SmallNormalHeapBucket BucketType;

};

#ifdef RECYCLER\_WRITE\_BARRIER

template <>

class SmallHeapBlockType<WithBarrierBit, SmallAllocationBlockAttributes>

{

public:

typedef SmallNormalWithBarrierHeapBlock BlockType;

typedef SmallNormalWithBarrierHeapBucket BucketType;

};

template <>

class SmallHeapBlockType<FinalizableWithBarrierBit, SmallAllocationBlockAttributes>

{

public:

typedef SmallFinalizableWithBarrierHeapBlock BlockType;

typedef SmallFinalizableWithBarrierHeapBucket BucketType;

};

#endif

#if SMALLBLOCK\_MEDIUM\_ALLOC

template <>

class SmallHeapBlockType<FinalizeBit, MediumAllocationBlockAttributes>

{

public:

typedef MediumFinalizableHeapBlock BlockType;

typedef MediumFinalizableHeapBucket BucketType;

};

template <>

class SmallHeapBlockType<LeafBit, MediumAllocationBlockAttributes>

{

public:

typedef MediumLeafHeapBlock BlockType;

typedef SmallLeafHeapBucketT<MediumAllocationBlockAttributes> BucketType;

};

template <>

class SmallHeapBlockType<NoBit, MediumAllocationBlockAttributes>

{

public:

typedef MediumNormalHeapBlock BlockType;

typedef MediumNormalHeapBucket BucketType;

};

#ifdef RECYCLER\_WRITE\_BARRIER

template <>

class SmallHeapBlockType<WithBarrierBit, MediumAllocationBlockAttributes>

{

public:

typedef MediumNormalWithBarrierHeapBlock BlockType;

typedef MediumNormalWithBarrierHeapBucket BucketType;

};

template <>

class SmallHeapBlockType<FinalizableWithBarrierBit, MediumAllocationBlockAttributes>

{

public:

typedef MediumFinalizableWithBarrierHeapBlock BlockType;

typedef MediumFinalizableWithBarrierHeapBucket BucketType;

};

#endif

#endif

template <class TBlockAttributes>

class HeapBucketGroup

{

public:

template <ObjectInfoBits objectAttributes>

typename SmallHeapBlockType<objectAttributes, TBlockAttributes>::BucketType& GetBucket()

{

CompileAssert(objectAttributes & FinalizeBit);

return this->finalizableHeapBucket;

}

template <>

typename SmallHeapBlockType<LeafBit, TBlockAttributes>::BucketType& GetBucket<LeafBit>()

{

return this->leafHeapBucket;

}

template <>

typename SmallHeapBlockType<NoBit, TBlockAttributes>::BucketType& GetBucket<NoBit>()

{

return this->heapBucket;

}

#ifdef RECYCLER\_WRITE\_BARRIER

template <>

typename SmallHeapBlockType<WithBarrierBit, TBlockAttributes>::BucketType& GetBucket<WithBarrierBit>()

{

return this->smallNormalWithBarrierHeapBucket;

}

template <>

typename SmallHeapBlockType<FinalizableWithBarrierBit, TBlockAttributes>::BucketType& GetBucket<FinalizableWithBarrierBit>()

{

return this->smallFinalizableWithBarrierHeapBucket;

}

#endif

void Initialize(HeapInfo \* heapInfo, uint sizeCat);

void ResetMarks(ResetMarkFlags flags);

void ScanInitialImplicitRoots(Recycler \* recycler);

void ScanNewImplicitRoots(Recycler \* recycler);

template<bool pageheap>

void Sweep(RecyclerSweep& recyclerSweep);

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

uint Rescan(Recycler \* recycler, RescanFlags flags);

void SweepPendingObjects(RecyclerSweep& recyclerSweep);

#endif

#ifdef PARTIAL\_GC\_ENABLED

void SweepPartialReusePages(RecyclerSweep& recyclerSweep);

void FinishPartialCollect(RecyclerSweep \* recyclerSweep);

#endif

#ifdef CONCURRENT\_GC\_ENABLED

void PrepareSweep();

void SetupBackgroundSweep(RecyclerSweep& recyclerSweep);

void TransferPendingEmptyHeapBlocks(RecyclerSweep& recyclerSweep);

#endif

template<bool pageheap>

void SweepFinalizableObjects(RecyclerSweep& recyclerSweep);

void DisposeObjects();

void TransferDisposedObjects();

void EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size));

void FinalizeAllObjects();

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t GetNonEmptyHeapBlockCount(bool checkCount) const;

size\_t GetEmptyHeapBlockCount() const;

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t Check();

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void Verify();

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void VerifyMark();

#endif

#if DBG

bool AllocatorsAreEmpty();

#endif

private:

SmallNormalHeapBucketT<TBlockAttributes> heapBucket;

SmallLeafHeapBucketT<TBlockAttributes> leafHeapBucket;

SmallFinalizableHeapBucketT<TBlockAttributes> finalizableHeapBucket;

#ifdef RECYCLER\_WRITE\_BARRIER

SmallNormalWithBarrierHeapBucketT<TBlockAttributes> smallNormalWithBarrierHeapBucket;

SmallFinalizableWithBarrierHeapBucketT<TBlockAttributes> smallFinalizableWithBarrierHeapBucket;

#endif

};

extern template class HeapBucketGroup<SmallAllocationBlockAttributes>;

extern template class HeapBucketGroup<MediumAllocationBlockAttributes>;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

EXPLICIT\_INSTANTIATE\_WITH\_SMALL\_HEAP\_BLOCK\_TYPE(SmallHeapBlockAllocator)

template \_\_forceinline char\* SmallHeapBlockAllocator<SmallNormalHeapBlock>::InlinedAllocImpl</\*canFaultInject\*/true>(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes);

template <typename TBlockType>

SmallHeapBlockAllocator<TBlockType>::SmallHeapBlockAllocator() :

freeObjectList(nullptr),

endAddress(nullptr),

heapBlock(nullptr),

prev(nullptr),

next(nullptr)

{

#ifdef RECYCLER\_TRACK\_NATIVE\_ALLOCATED\_OBJECTS

this->lastNonNativeBumpAllocatedBlock = nullptr;

#endif

}

template <typename TBlockType>

void

SmallHeapBlockAllocator<TBlockType>::Initialize()

{

Assert(this->freeObjectList == nullptr);

Assert(this->endAddress == nullptr);

Assert(this->heapBlock == nullptr);

this->prev = this;

this->next = this;

}

template <typename TBlockType>

void

SmallHeapBlockAllocator<TBlockType>::UpdateHeapBlock()

{

if (heapBlock != nullptr)

{

if (this->endAddress == nullptr)

{

heapBlock->freeObjectList = this->freeObjectList;

}

else

{

Assert(heapBlock->freeObjectList == nullptr);

}

}

}

template <typename TBlockType>

void

SmallHeapBlockAllocator<TBlockType>::Clear()

{

TBlockType \* heapBlock = this->heapBlock;

if (heapBlock != nullptr)

{

Assert(heapBlock->isInAllocator);

heapBlock->isInAllocator = false;

FreeObject \* remainingFreeObjectList = nullptr;

if (this->endAddress != nullptr)

{

#ifdef RECYCLER\_TRACK\_NATIVE\_ALLOCATED\_OBJECTS

TrackNativeAllocatedObjects();

lastNonNativeBumpAllocatedBlock = nullptr;

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

// Need to tell the tracker

this->bucket->heapInfo->recycler->TrackUnallocated((char \*)this->freeObjectList, this->endAddress, this->bucket->sizeCat);

#endif

RecyclerMemoryTracking::ReportUnallocated(this->heapBlock->heapBucket->heapInfo->recycler, (char \*)this->freeObjectList, this->endAddress, heapBlock->heapBucket->sizeCat);

#ifdef RECYCLER\_PERF\_COUNTERS

size\_t unallocatedObjects = heapBlock->objectCount - ((char \*)this->freeObjectList - heapBlock->address) / heapBlock->objectSize;

size\_t unallocatedObjectBytes = unallocatedObjects \* heapBlock->GetObjectSize();

RECYCLER\_PERF\_COUNTER\_ADD(LiveObject, unallocatedObjects);

RECYCLER\_PERF\_COUNTER\_ADD(LiveObjectSize, unallocatedObjectBytes);

RECYCLER\_PERF\_COUNTER\_SUB(FreeObjectSize, unallocatedObjectBytes);

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockLiveObject, unallocatedObjects);

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockLiveObjectSize, unallocatedObjectBytes);

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockFreeObjectSize, unallocatedObjectBytes);

#endif

Assert(heapBlock->freeObjectList == nullptr);

this->endAddress = nullptr;

}

else

{

remainingFreeObjectList = this->freeObjectList;

heapBlock->freeObjectList = remainingFreeObjectList;

}

this->freeObjectList = nullptr;

// this->freeObjectList and this->lastFreeCount are accessed in SmallHeapBlock::ResetMarks

// the order of access there is first we see if lastFreeCount = 0, and if it is, we assert

// that freeObjectList = null. Because of ARM's memory model, we need to insert barriers

// so that the two variables can be accessed correctly across threads. Here, after we write

// to this->freeObjectList, we insert a write barrier so that if this->lastFreeCount is 0,

// this->freeObjectList must have been set to null. On the other end, we stick a read barrier

// We use the MemoryBarrier macro because of ARMs lack of a separate read barrier

#if defined(\_M\_ARM32\_OR\_ARM64)

#if DBG

MemoryBarrier();

#endif

#endif

if (remainingFreeObjectList == nullptr)

{

uint lastFreeCount = heapBlock->GetAndClearLastFreeCount();

heapBlock->heapBucket->heapInfo->uncollectedAllocBytes += lastFreeCount \* heapBlock->GetObjectSize();

Assert(heapBlock->lastUncollectedAllocBytes == 0);

DebugOnly(heapBlock->lastUncollectedAllocBytes = lastFreeCount \* heapBlock->GetObjectSize());

}

else

{

DebugOnly(heapBlock->SetIsClearedFromAllocator(true));

}

this->heapBlock = nullptr;

RECYCLER\_SLOW\_CHECK(heapBlock->CheckDebugFreeBitVector(false));

}

else if (this->freeObjectList != nullptr)

{

// Explicit Free Object List

#ifdef RECYCLER\_MEMORY\_VERIFY

FreeObject\* freeObject = this->freeObjectList;

while (freeObject)

{

HeapBlock\* heapBlock = this->bucket->GetRecycler()->FindHeapBlock((void\*) freeObject);

Assert(heapBlock != nullptr);

Assert(!heapBlock->IsLargeHeapBlock());

TBlockType\* smallBlock = (TBlockType\*)heapBlock;

smallBlock->ClearExplicitFreeBitForObject((void\*) freeObject);

freeObject = freeObject->GetNext();

}

#endif

this->freeObjectList = nullptr;

}

}

template <typename TBlockType>

void

SmallHeapBlockAllocator<TBlockType>::SetNew(BlockType \* heapBlock)

{

Assert(this->endAddress == nullptr);

Assert(this->heapBlock == nullptr);

Assert(this->freeObjectList == nullptr);

Assert(heapBlock != nullptr);

Assert(heapBlock->freeObjectList == nullptr);

Assert(heapBlock->lastFreeCount != 0);

Assert(!heapBlock->isInAllocator);

heapBlock->isInAllocator = true;

this->heapBlock = heapBlock;

this->freeObjectList = (FreeObject \*)heapBlock->GetAddress();

this->endAddress = heapBlock->GetEndAddress();

}

template <typename TBlockType>

void

SmallHeapBlockAllocator<TBlockType>::Set(BlockType \* heapBlock)

{

Assert(this->endAddress == nullptr);

Assert(this->heapBlock == nullptr);

Assert(this->freeObjectList == nullptr);

Assert(heapBlock != nullptr);

Assert(heapBlock->freeObjectList != nullptr);

Assert(heapBlock->lastFreeCount != 0);

Assert(!heapBlock->isInAllocator);

heapBlock->isInAllocator = true;

this->heapBlock = heapBlock;

RECYCLER\_SLOW\_CHECK(this->heapBlock->CheckDebugFreeBitVector(true));

this->freeObjectList = this->heapBlock->freeObjectList;

}

template <typename TBlockType>

void

SmallHeapBlockAllocator<TBlockType>::SetExplicitFreeList(FreeObject\* list)

{

Assert(list != nullptr);

Assert(this->heapBlock == nullptr);

Assert(this->freeObjectList == nullptr);

this->freeObjectList = list;

}

#ifdef RECYCLER\_TRACK\_NATIVE\_ALLOCATED\_OBJECTS

template <typename TBlockType>

void

SmallHeapBlockAllocator<TBlockType>::TrackNativeAllocatedObjects()

{

Assert(this->freeObjectList != nullptr && endAddress != nullptr);

Assert(this->heapBlock != nullptr);

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(RECYCLER\_MEMORY\_VERIFY) || defined(MEMSPECT\_TRACKING) || defined(ETW\_MEMORY\_TRACKING)

if (pfnTrackNativeAllocatedObjectCallBack == nullptr)

{

return;

}

if (lastNonNativeBumpAllocatedBlock == nullptr)

{

#ifdef RECYCLER\_PAGE\_HEAP

Assert((char \*)this->freeObjectList == this->heapBlock->GetAddress() || ((SmallHeapBlock\*) this->heapBlock)->InPageHeapMode());

#else

Assert((char \*)this->freeObjectList == this->heapBlock->GetAddress());

#endif

return;

}

Recycler \* recycler = this->heapBlock->heapBucket->heapInfo->recycler;

size\_t sizeCat = this->heapBlock->heapBucket->sizeCat;

char \* curr = lastNonNativeBumpAllocatedBlock + sizeCat;

Assert(curr <= (char \*)this->freeObjectList);

#if DBG\_DUMP

AllocationVerboseTrace(recycler->GetRecyclerFlagsTable(), L"TrackNativeAllocatedObjects: recycler = 0x%p, sizeCat = %u, lastRuntimeAllocatedBlock = 0x%p, freeObjectList = 0x%p, nativeAllocatedObjectCount = %u\n",

recycler, sizeCat, this->lastNonNativeBumpAllocatedBlock, this->freeObjectList, ((char \*)this->freeObjectList - curr) / sizeCat);

#endif

while (curr < (char \*)this->freeObjectList)

{

pfnTrackNativeAllocatedObjectCallBack(recycler, curr, sizeCat);

curr += sizeCat;

}

#elif defined(RECYCLER\_PERF\_COUNTERS)

if (lastNonNativeBumpAllocatedBlock == nullptr)

{

return;

}

size\_t sizeCat = this->heapBlock->heapBucket->sizeCat;

char \* curr = lastNonNativeBumpAllocatedBlock + sizeCat;

Assert(curr <= (char \*)this->freeObjectList);

size\_t byteCount = ((char \*)this->freeObjectList - curr);

#if DBG\_DUMP

AllocationVerboseTrace(L"TrackNativeAllocatedObjects: recycler = 0x%p, sizeCat = %u, lastRuntimeAllocatedBlock = 0x%p, freeObjectList = 0x%p, nativeAllocatedObjectCount = %u\n",

recycler, sizeCat, this->lastNonNativeBumpAllocatedBlock, this->freeObjectList, ((char \*)this->freeObjectList - curr) / sizeCat);

#endif

RECYCLER\_PERF\_COUNTER\_ADD(LiveObject, byteCount / sizeCat);

RECYCLER\_PERF\_COUNTER\_ADD(LiveObjectSize, byteCount);

RECYCLER\_PERF\_COUNTER\_SUB(FreeObjectSize, byteCount);

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockLiveObject, byteCount / sizeCat);

RECYCLER\_PERF\_COUNTER\_ADD(SmallHeapBlockLiveObjectSize, byteCount);

RECYCLER\_PERF\_COUNTER\_SUB(SmallHeapBlockFreeObjectSize, byteCount);

#else

#error Not implemented

#endif

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

template <typename TBlockType>

class SmallHeapBlockAllocator

{

public:

typedef TBlockType BlockType;

SmallHeapBlockAllocator();

void Initialize();

template <ObjectInfoBits attributes>

\_\_inline char \* InlinedAlloc(Recycler \* recycler, size\_t sizeCat);

// Pass through template parameter to InlinedAllocImpl

template <bool canFaultInject>

\_\_inline char \* SlowAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes);

// There are paths where we simply can't OOM here, so we shouldn't fault inject as it creates a bit of a mess

template <bool canFaultInject>

\_\_inline char\* InlinedAllocImpl(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes);

#ifdef RECYCLER\_PAGE\_HEAP

\_\_inline char \*PageHeapAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes, PageHeapMode mode);

#endif

TBlockType \* GetHeapBlock() const { return heapBlock; }

SmallHeapBlockAllocator \* GetNext() const { return next; }

void Set(TBlockType \* heapBlock);

void SetNew(TBlockType \* heapBlock);

void Clear();

void UpdateHeapBlock();

void SetExplicitFreeList(FreeObject\* list);

static uint32 GetEndAddressOffset() { return offsetof(SmallHeapBlockAllocator, endAddress); }

char \*GetEndAddress() { return endAddress; }

static uint32 GetFreeObjectListOffset() { return offsetof(SmallHeapBlockAllocator, freeObjectList); }

FreeObject \*GetFreeObjectList() { return freeObjectList; }

void SetFreeObjectList(FreeObject \*freeObject) { freeObjectList = freeObject; }

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(RECYCLER\_MEMORY\_VERIFY) || defined(MEMSPECT\_TRACKING) || defined(ETW\_MEMORY\_TRACKING)

void SetTrackNativeAllocatedObjectCallBack(void (\*pfnCallBack)(Recycler \*, void \*, size\_t))

{

pfnTrackNativeAllocatedObjectCallBack = pfnCallBack;

}

#endif

#if DBG

FreeObject \* GetExplicitFreeList() const

{

Assert(IsExplicitFreeObjectListAllocMode());

return this->freeObjectList;

}

#endif

bool IsBumpAllocMode() const

{

return endAddress != nullptr;

}

bool IsExplicitFreeObjectListAllocMode() const

{

return this->heapBlock == nullptr;

}

bool IsFreeListAllocMode() const

{

return !IsBumpAllocMode() && !IsExplicitFreeObjectListAllocMode();

}

private:

static bool NeedSetAttributes(ObjectInfoBits attributes)

{

return attributes != LeafBit && (attributes & InternalObjectInfoBitMask) != 0;

}

char \* endAddress;

FreeObject \* freeObjectList;

TBlockType \* heapBlock;

SmallHeapBlockAllocator \* prev;

SmallHeapBlockAllocator \* next;

friend class HeapBucketT<BlockType>;

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <class TBlockAttributes>

friend class SmallHeapBlockT;

#endif

#ifdef PROFILE\_RECYCLER\_ALLOC

HeapBucket \* bucket;

#endif

#ifdef RECYCLER\_TRACK\_NATIVE\_ALLOCATED\_OBJECTS

char \* lastNonNativeBumpAllocatedBlock;

void TrackNativeAllocatedObjects();

#endif

#if defined(PROFILE\_RECYCLER\_ALLOC) || defined(RECYCLER\_MEMORY\_VERIFY) || defined(MEMSPECT\_TRACKING) || defined(ETW\_MEMORY\_TRACKING)

void (\*pfnTrackNativeAllocatedObjectCallBack)(Recycler \* recycler, void \*, size\_t sizeCat);

#endif

};

#ifdef RECYCLER\_PAGE\_HEAP

template <typename TBlockType>

\_\_inline

char \*

SmallHeapBlockAllocator<TBlockType>::PageHeapAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes, PageHeapMode mode)

{

if (this->heapBlock == nullptr)

{

return nullptr;

}

// In integrated page, we can get a heapBlock is not pageheap enabled

if (!heapBlock->InPageHeapMode())

{

return nullptr;

}

TBlockType\* smallBlock = (TBlockType\*) this->heapBlock;

// Free list allocation not supported in page heap mode

// since if we sweep, the block should be empty

Assert(this->endAddress != nullptr);

// We do one of two things:

// 1) Either bump allocate

// 2) Allocate from the explicit free list

// Explicit free list will already be at the right point in the heap block

if ((char\*)freeObjectList == smallBlock->address && !IsExplicitFreeObjectListAllocMode())

{

if (mode == PageHeapMode::PageHeapModeBlockEnd)

{

// Allocate at the last valid object

// This could cause some extra padding at the end

// e.g. For a Heap block with the range 0x1000 to 0x2000, and sizeCat 48

// This can fit 85 objects in it, the last object at 0x1FC0, causing the

// last 16 bytes to be wasted. So if there is a buffer overflow of <= 16 bytes,

// PageHeap as implemented today will not catch it.

char\* objectAddress = smallBlock->GetAddress() + ((smallBlock->GetObjectCount() - 1) \* sizeCat);

Assert(objectAddress <= this->endAddress - sizeCat);

freeObjectList = (FreeObject\*)objectAddress;

}

}

char\* memBlock = InlinedAllocImpl<true/\* allow fault injection \*/>(recycler, sizeCat, (ObjectInfoBits)attributes);

if (memBlock)

{

#if DBG

HeapBlock\* block = recycler->FindHeapBlock(memBlock);

Assert(!block->IsLargeHeapBlock());

((TBlockType\*)block)->VerifyPageHeapAllocation(memBlock, mode);

#endif

PageHeapVerboseTrace(recycler->GetRecyclerFlagsTable(), L"Allocated from block 0x%p\n", smallBlock);

// Close off allocation from this block

this->freeObjectList = (FreeObject\*) this->endAddress;

#ifdef RECYCLER\_TRACK\_NATIVE\_ALLOCATED\_OBJECTS

this->lastNonNativeBumpAllocatedBlock = (char\*) this->freeObjectList - sizeCat;

#endif

if (recycler->ShouldCapturePageHeapAllocStack())

{

smallBlock->CapturePageHeapAllocStack();

}

}

return memBlock;

}

#endif

template <typename TBlockType>

template <bool canFaultInject>

\_\_inline char\*

SmallHeapBlockAllocator<TBlockType>::InlinedAllocImpl(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

AUTO\_NO\_EXCEPTION\_REGION;

if (canFaultInject)

{

FAULTINJECT\_MEMORY\_NOTHROW(L"InlinedAllocImpl", sizeCat);

}

char \* memBlock = (char \*)freeObjectList;

char \* nextCurrentAddress = memBlock + sizeCat;

char \* endAddress = this->endAddress;

if (nextCurrentAddress <= endAddress)

{

// Bump Allocation

Assert(this->IsBumpAllocMode());

#ifdef RECYCLER\_TRACK\_NATIVE\_ALLOCATED\_OBJECTS

TrackNativeAllocatedObjects();

lastNonNativeBumpAllocatedBlock = memBlock;

#endif

freeObjectList = (FreeObject \*)nextCurrentAddress;

if (NeedSetAttributes(attributes))

{

heapBlock->SetAttributes(memBlock, (attributes & StoredObjectInfoBitMask));

}

return memBlock;

}

if (memBlock != nullptr && endAddress == nullptr)

{

// Free list allocation

Assert(!this->IsBumpAllocMode());

if (NeedSetAttributes(attributes))

{

TBlockType \* allocationHeapBlock = this->heapBlock;

if (allocationHeapBlock == nullptr)

{

Assert(this->IsExplicitFreeObjectListAllocMode());

allocationHeapBlock = (TBlockType \*)recycler->FindHeapBlock(memBlock);

Assert(allocationHeapBlock != nullptr);

Assert(!allocationHeapBlock->IsLargeHeapBlock());

}

allocationHeapBlock->SetAttributes(memBlock, (attributes & StoredObjectInfoBitMask));

}

freeObjectList = ((FreeObject \*)memBlock)->GetNext();

#ifdef RECYCLER\_MEMORY\_VERIFY

((FreeObject \*)memBlock)->DebugFillNext();

if (this->IsExplicitFreeObjectListAllocMode())

{

HeapBlock\* heapBlock = recycler->FindHeapBlock(memBlock);

Assert(heapBlock != nullptr);

Assert(!heapBlock->IsLargeHeapBlock());

TBlockType\* smallBlock = (TBlockType\*)heapBlock;

smallBlock->ClearExplicitFreeBitForObject(memBlock);

}

#endif

#if DBG || defined(RECYCLER\_STATS)

if (!IsExplicitFreeObjectListAllocMode())

{

BOOL isSet = heapBlock->GetDebugFreeBitVector()->TestAndClear(heapBlock->GetAddressBitIndex(memBlock));

Assert(isSet);

}

#endif

return memBlock;

}

return nullptr;

}

template <typename TBlockType>

template <ObjectInfoBits attributes>

\_\_inline char \*

SmallHeapBlockAllocator<TBlockType>::InlinedAlloc(Recycler \* recycler, size\_t sizeCat)

{

return InlinedAllocImpl<true /\* allow fault injection \*/>(recycler, sizeCat, attributes);

}

template <typename TBlockType>

template <bool canFaultInject>

\_\_inline

char \*

SmallHeapBlockAllocator<TBlockType>::SlowAlloc(Recycler \* recycler, size\_t sizeCat, ObjectInfoBits attributes)

{

Assert((attributes & InternalObjectInfoBitMask) == attributes);

return InlinedAllocImpl<canFaultInject>(recycler, sizeCat, attributes);

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

template <class TBlockAttributes>

SmallLeafHeapBlockT<TBlockAttributes> \*

SmallLeafHeapBlockT<TBlockAttributes>::New(HeapBucketT<SmallLeafHeapBlockT<TBlockAttributes>> \* bucket)

{

CompileAssert(TBlockAttributes::MaxObjectSize <= USHRT\_MAX);

Assert(bucket->sizeCat <= TBlockAttributes::MaxObjectSize);

Assert((TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / bucket->sizeCat <= USHRT\_MAX);

ushort objectSize = (ushort)bucket->sizeCat;

ushort objectCount = (ushort)(TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / objectSize;

return NoMemProtectHeapNewNoThrowPlusPrefixZ(GetAllocPlusSize(objectCount), SmallLeafHeapBlockT<TBlockAttributes>, bucket, objectSize, objectCount);

}

template <>

SmallLeafHeapBlockT<SmallAllocationBlockAttributes>::SmallLeafHeapBlockT(HeapBucketT<SmallLeafHeapBlockT<SmallAllocationBlockAttributes>> \* bucket, ushort objectSize, ushort objectCount)

: SmallHeapBlockT<SmallAllocationBlockAttributes>(bucket, objectSize, objectCount, HeapBlockType::SmallLeafBlockType)

{

Assert(objectCount > 1 && objectCount == (this->GetPageCount() \* AutoSystemInfo::PageSize)/ objectSize);

}

template <>

SmallLeafHeapBlockT<MediumAllocationBlockAttributes>::SmallLeafHeapBlockT(HeapBucketT<SmallLeafHeapBlockT<MediumAllocationBlockAttributes>> \* bucket, ushort objectSize, ushort objectCount)

: SmallHeapBlockT<MediumAllocationBlockAttributes>(bucket, objectSize, objectCount, HeapBlockType::MediumLeafBlockType)

{

Assert(objectCount > 1 && objectCount == (this->GetPageCount() \* AutoSystemInfo::PageSize) / objectSize);

}

template <class TBlockAttributes>

void

SmallLeafHeapBlockT<TBlockAttributes>::Delete(SmallLeafHeapBlockT<TBlockAttributes> \* heapBlock)

{

Assert(heapBlock->IsLeafBlock());

NoMemProtectHeapDeletePlusPrefix(GetAllocPlusSize(heapBlock->objectCount), heapBlock);

}

template <class TBlockAttributes>

void

SmallLeafHeapBlockT<TBlockAttributes>::ScanNewImplicitRoots(Recycler \* recycler)

{

\_\_super::ScanNewImplicitRootsBase([](void \* object, size\_t objectSize){});

}

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <class TBlockAttributes>

bool

SmallLeafHeapBlockT<TBlockAttributes>::GetFreeObjectListOnAllocator(FreeObject \*\* freeObjectList)

{

return GetFreeObjectListOnAllocatorImpl<SmallLeafHeapBlockT<TBlockAttributes>>(freeObjectList);

}

#endif

// Declare the class templates

template class SmallLeafHeapBlockT<SmallAllocationBlockAttributes>;

template class SmallLeafHeapBlockT<MediumAllocationBlockAttributes>;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

template <class TBlockAttributes>

class SmallLeafHeapBlockT : public SmallHeapBlockT<TBlockAttributes>

{

friend class HeapBucketT<SmallLeafHeapBlockT>;

public:

static const ObjectInfoBits RequiredAttributes = LeafBit;

typedef TBlockAttributes HeapBlockAttributes;

SmallLeafHeapBlockT \* GetNextBlock() const { return \_\_super::GetNextBlock()->AsLeafBlock<TBlockAttributes>(); }

void SetNextBlock(SmallLeafHeapBlockT \* next) { \_\_super::SetNextBlock(next); }

void ScanNewImplicitRoots(Recycler \* recycler);

static SmallLeafHeapBlockT \* New(HeapBucketT<SmallLeafHeapBlockT> \* bucket);

static void Delete(SmallLeafHeapBlockT \* block);

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

virtual bool GetFreeObjectListOnAllocator(FreeObject \*\* freeObjectList) override;

#endif

virtual bool FindHeapObject(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject) override sealed

{

return FindHeapObjectImpl<SmallLeafHeapBlockT>(objectAddress, recycler, flags, heapObject);

}

private:

SmallLeafHeapBlockT(HeapBucketT<SmallLeafHeapBlockT> \* bucket, ushort objectSize, ushort objectCount);

};

// Declare the class templates

extern template class SmallLeafHeapBlockT<SmallAllocationBlockAttributes>;

extern template class SmallLeafHeapBlockT<MediumAllocationBlockAttributes>;

typedef SmallLeafHeapBlockT<SmallAllocationBlockAttributes> SmallLeafHeapBlock;

typedef SmallLeafHeapBlockT<MediumAllocationBlockAttributes> MediumLeafHeapBlock;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

template void SmallLeafHeapBucketT<SmallAllocationBlockAttributes>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallLeafHeapBucketT<SmallAllocationBlockAttributes>::Sweep<false>(RecyclerSweep& recyclerSweep);

template void SmallLeafHeapBucketT<MediumAllocationBlockAttributes>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallLeafHeapBucketT<MediumAllocationBlockAttributes>::Sweep<false>(RecyclerSweep& recyclerSweep);

template <typename TBlockAttributes>

template<bool pageheap>

void

SmallLeafHeapBucketT<TBlockAttributes>::Sweep(RecyclerSweep& recyclerSweep)

{

BaseT::SweepBucket<pageheap>(recyclerSweep, [](RecyclerSweep& recyclerSweep){});

}

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

template <typename TBlockAttributes>

size\_t

SmallLeafHeapBucketT<TBlockAttributes>::GetNonEmptyHeapBlockCount(bool checkCount) const

{

return \_\_super::GetNonEmptyHeapBlockCount(checkCount);

}

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <typename TBlockAttributes>

size\_t

SmallLeafHeapBucketT<TBlockAttributes>::Check()

{

return \_\_super::Check(true);

}

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

template <typename TBlockAttributes>

void

SmallLeafHeapBucketT<TBlockAttributes>::Verify()

{

BaseT::Verify();

}

#endif

#ifdef RECYCLER\_VERIFY\_MARK

template <typename TBlockAttributes>

void

SmallLeafHeapBucketT<TBlockAttributes>::VerifyMark()

{

\_\_super::VerifyMark();

}

#endif

template class SmallLeafHeapBucketT<SmallAllocationBlockAttributes>;

template class SmallLeafHeapBucketT<MediumAllocationBlockAttributes>;

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

namespace Memory

{

template <class TBlockAttributes>

class SmallLeafHeapBucketT : public HeapBucketT<SmallLeafHeapBlockT<TBlockAttributes>>

{

typedef HeapBucketT<SmallLeafHeapBlockT<TBlockAttributes>> BaseT;

protected:

friend class HeapBucket;

template <class TBlockAttributes>

friend class HeapBucketGroup;

template<bool pageheap>

void Sweep(RecyclerSweep& recyclerSweep);

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t GetNonEmptyHeapBlockCount(bool checkCount) const;

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t Check();

friend class HeapBucketT<SmallLeafHeapBlockT<TBlockAttributes>>;

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void Verify();

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void VerifyMark();

#endif

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

friend class ScriptMemoryDumper;

#endif

};

typedef SmallLeafHeapBucketT<SmallAllocationBlockAttributes> SmallLeafHeapBucket;

typedef SmallLeafHeapBucketT<MediumAllocationBlockAttributes> MediumLeafHeapBucket;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

template <class TBlockAttributes>

SmallNormalHeapBlockT<TBlockAttributes> \*

SmallNormalHeapBlockT<TBlockAttributes>::New(HeapBucketT<SmallNormalHeapBlockT<TBlockAttributes>> \* bucket)

{

CompileAssert(TBlockAttributes::MaxObjectSize <= USHRT\_MAX);

Assert(bucket->sizeCat <= TBlockAttributes::MaxObjectSize);

Assert((TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / bucket->sizeCat <= USHRT\_MAX);

ushort objectSize = (ushort)bucket->sizeCat;

ushort objectCount = (ushort)(TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / objectSize;

HeapBlockType blockType = (TBlockAttributes::IsSmallBlock ? SmallNormalBlockType : MediumNormalBlockType);

return NoMemProtectHeapNewNoThrowPlusPrefixZ(GetAllocPlusSize(objectCount), SmallNormalHeapBlockT<TBlockAttributes>, bucket, objectSize, objectCount, blockType);

}

#ifdef RECYCLER\_WRITE\_BARRIER

template <class TBlockAttributes>

SmallNormalWithBarrierHeapBlockT<TBlockAttributes> \*

SmallNormalWithBarrierHeapBlockT<TBlockAttributes>::New(HeapBucketT<SmallNormalWithBarrierHeapBlockT<TBlockAttributes>> \* bucket)

{

CompileAssert(TBlockAttributes::MaxObjectSize <= USHRT\_MAX);

Assert(bucket->sizeCat <= TBlockAttributes::MaxObjectSize);

Assert((TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / bucket->sizeCat <= USHRT\_MAX);

ushort objectSize = (ushort)bucket->sizeCat;

ushort objectCount = (ushort)(TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / objectSize;

HeapBlockType blockType = (TBlockAttributes::IsSmallBlock ? SmallNormalBlockWithBarrierType : MediumNormalBlockWithBarrierType);

return NoMemProtectHeapNewNoThrowPlusPrefixZ(GetAllocPlusSize(objectCount), SmallNormalWithBarrierHeapBlockT<TBlockAttributes>, bucket, objectSize, objectCount, blockType);

}

#endif

template <class TBlockAttributes>

void

SmallNormalHeapBlockT<TBlockAttributes>::Delete(SmallNormalHeapBlockT<TBlockAttributes> \* heapBlock)

{

Assert(heapBlock->IsNormalBlock());

NoMemProtectHeapDeletePlusPrefix(GetAllocPlusSize(heapBlock->objectCount), heapBlock);

}

#ifdef RECYCLER\_WRITE\_BARRIER

template <class TBlockAttributes>

void

SmallNormalWithBarrierHeapBlockT<TBlockAttributes>::Delete(SmallNormalWithBarrierHeapBlockT<TBlockAttributes> \* heapBlock)

{

Assert(heapBlock->IsNormalWriteBarrierBlock());

NoMemProtectHeapDeletePlusPrefix(GetAllocPlusSize(heapBlock->objectCount), heapBlock);

}

#endif

template <class TBlockAttributes>

SmallNormalHeapBlockT<TBlockAttributes>::SmallNormalHeapBlockT(HeapBucket \* bucket, ushort objectSize, ushort objectCount, HeapBlockType heapBlockType)

: SmallHeapBlockT<TBlockAttributes>(bucket, objectSize, objectCount, heapBlockType)

{

}

template <>

SmallNormalHeapBlockT<SmallAllocationBlockAttributes>::SmallNormalHeapBlockT(HeapBucketT<SmallNormalHeapBlockT<SmallAllocationBlockAttributes>> \* bucket, ushort objectSize, ushort objectCount)

: SmallHeapBlockT<SmallAllocationBlockAttributes>(bucket, objectSize, objectCount, SmallNormalBlockType)

{

}

template <>

SmallNormalHeapBlockT<MediumAllocationBlockAttributes>::SmallNormalHeapBlockT(HeapBucketT<SmallNormalHeapBlockT<MediumAllocationBlockAttributes>> \* bucket, ushort objectSize, ushort objectCount)

: SmallHeapBlockT<MediumAllocationBlockAttributes>(bucket, objectSize, objectCount, MediumNormalBlockType)

{

}

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

template <class TBlockAttributes>

void

SmallNormalHeapBlockT<TBlockAttributes>::ScanInitialImplicitRoots(Recycler \* recycler)

{

Assert(IsAnyNormalBlock());

uint const localObjectCount = this->objectCount;

uint const localObjectSize = this->GetObjectSize();

// We can use the markCount to optimize the scan because we update it in ResetMark

if (this->markCount == localObjectCount

#ifdef RECYCLER\_DUMP\_OBJECT\_GRAPH

&& recycler->objectGraphDumper == nullptr

#endif

)

{

// TODO: only interior?

recycler->ScanObjectInlineInterior((void \*\*)this->GetAddress(), localObjectSize \* localObjectCount);

}

else if (this->markCount != 0)

{

uint const localObjectBitDelta = this->GetObjectBitDelta();

for (uint i = 0; i < localObjectCount; i++)

{

if (this->GetMarkedBitVector()->Test(i \* localObjectBitDelta))

{

// TODO: only interior?

void \*\* address = (void \*\*)(this->GetAddress() + i \* localObjectSize);

DUMP\_IMPLICIT\_ROOT(recycler, address);

recycler->ScanObjectInlineInterior(address, localObjectSize);

}

}

}

}

template <class TBlockAttributes>

void

SmallNormalHeapBlockT<TBlockAttributes>::ScanNewImplicitRoots(Recycler \* recycler)

{

Assert(IsAnyNormalBlock());

\_\_super::ScanNewImplicitRootsBase([recycler](void \* objectAddress, size\_t objectSize)

{

// TODO: only interior?

recycler->ScanObjectInlineInterior((void \*\*)objectAddress, objectSize);

});

}

// static

template <class TBlockAttributes>

bool

SmallNormalHeapBlockT<TBlockAttributes>::RescanObject(SmallNormalHeapBlockT<TBlockAttributes>\* block,

\_\_in\_ecount(localObjectSize) char \* objectAddress, uint localObjectSize,

uint objectIndex, Recycler \* recycler)

{

// REVIEW: This would be a good assert to have but we don't have the heap block here

// Assert(block->GetAddressIndex(objectAddress) != SmallHeapBlockT<TBlockAttributes>::InvalidAddressBit);

if (!recycler->AddMark(objectAddress, localObjectSize))

{

// Failed to add to the mark stack due to OOM.

return false;

}

RECYCLER\_STATS\_INC(recycler, markData.rescanObjectCount);

RECYCLER\_STATS\_ADD(recycler, markData.rescanObjectByteCount, localObjectSize);

return true;

}

// static

template <class TBlockAttributes>

bool

SmallNormalHeapBlockT<TBlockAttributes>::CanRescanFullBlock()

{

return true;

}

template <class TBlockAttributes>

uint

SmallNormalHeapBlockT<TBlockAttributes>::CalculateMarkCountForPage(SmallHeapBlockBitVector \* markBits, uint bucketIndex, uint pageStartBitIndex)

{

SmallHeapBlockBitVector temp;

SmallHeapBlockBitVector const\* invalid = HeapInfo::GetInvalidBitVectorForBucket<TBlockAttributes>(bucketIndex);

// Remove any invalid bits from the calculation

temp.Copy(markBits);

temp.Minus(invalid);

Assert(pageStartBitIndex % HeapBlockMap32::PageMarkBitCount == 0);

uint rescanMarkCount = temp.GetRange<HeapBlockMap32::PageMarkBitCount>(pageStartBitIndex)->Count();

// If the first object on the page is not at the start of the page, then the object containing

// the first few bytes of the page is not included in this mark count

// The caller will have to account for this

return rescanMarkCount;

}

#endif

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

template <class TBlockAttributes>

void

SmallNormalHeapBlockT<TBlockAttributes>::FinishPartialCollect()

{

// We don't allocate from a partially swept block

Assert(this->IsFreeBitsValid());

RECYCLER\_SLOW\_CHECK(CheckFreeBitVector(true));

}

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <class TBlockAttributes>

bool

SmallNormalHeapBlockT<TBlockAttributes>::GetFreeObjectListOnAllocator(FreeObject \*\* freeObjectList)

{

return GetFreeObjectListOnAllocatorImpl<SmallNormalHeapBlockT<TBlockAttributes>>(freeObjectList);

}

#endif

template class SmallNormalHeapBlockT<SmallAllocationBlockAttributes>;

template class SmallNormalHeapBlockT<MediumAllocationBlockAttributes>;

#ifdef RECYCLER\_WRITE\_BARRIER

template class SmallNormalWithBarrierHeapBlockT<SmallAllocationBlockAttributes>;

template class SmallNormalWithBarrierHeapBlockT<MediumAllocationBlockAttributes>;

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

template <class TBlockAttributes>

class SmallNormalHeapBlockT : public SmallHeapBlockT<TBlockAttributes>

{

friend class HeapBucketT<SmallNormalHeapBlockT>;

public:

typedef TBlockAttributes HeapBlockAttributes;

static const ObjectInfoBits RequiredAttributes = NoBit;

static const bool IsLeafOnly = false;

static SmallNormalHeapBlockT \* New(HeapBucketT<SmallNormalHeapBlockT> \* bucket);

static void Delete(SmallNormalHeapBlockT \* block);

SmallNormalHeapBlockT \* GetNextBlock() const { return \_\_super::GetNextBlock()->AsNormalBlock<TBlockAttributes>(); }

void SetNextBlock(SmallNormalHeapBlockT \* next) { \_\_super::SetNextBlock(next); }

void ScanInitialImplicitRoots(Recycler \* recycler);

void ScanNewImplicitRoots(Recycler \* recycler);

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

static uint CalculateMarkCountForPage(SmallHeapBlockBitVector\* markBits, uint bucketIndex, uint pageStartBitIndex);

static bool CanRescanFullBlock();

static bool RescanObject(SmallNormalHeapBlockT<TBlockAttributes> \* block, \_\_in\_ecount(localObjectSize) char \* objectAddress, uint localObjectSize, uint objectIndex, Recycler \* recycler);

#endif

#if defined(PARTIAL\_GC\_ENABLED) && defined(CONCURRENT\_GC\_ENABLED)

void FinishPartialCollect();

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

virtual bool GetFreeObjectListOnAllocator(FreeObject \*\* freeObjectList) override;

#endif

virtual bool FindHeapObject(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject) override

{

return FindHeapObjectImpl<SmallNormalHeapBlockT>(objectAddress, recycler, flags, heapObject);

}

protected:

SmallNormalHeapBlockT(HeapBucket \* bucket, ushort objectSize, ushort objectCount, HeapBlockType heapBlockType);

private:

SmallNormalHeapBlockT(HeapBucketT<SmallNormalHeapBlockT> \* bucket, ushort objectSize, ushort objectCount);

};

#ifdef RECYCLER\_WRITE\_BARRIER

template <class TBlockAttributes>

class SmallNormalWithBarrierHeapBlockT : public SmallNormalHeapBlockT<TBlockAttributes>

{

friend class HeapBucketT<SmallNormalWithBarrierHeapBlockT>;

public:

typedef TBlockAttributes HeapBlockAttributes;

static const ObjectInfoBits RequiredAttributes = WithBarrierBit;

static const bool IsLeafOnly = false;

static SmallNormalWithBarrierHeapBlockT \* New(HeapBucketT<SmallNormalWithBarrierHeapBlockT> \* bucket);

static void Delete(SmallNormalWithBarrierHeapBlockT \* heapBlock);

SmallNormalWithBarrierHeapBlockT \* GetNextBlock() const { return ((SmallHeapBlock\*) this)->GetNextBlock()->AsNormalWriteBarrierBlock<TBlockAttributes>(); }

virtual bool FindHeapObject(void\* objectAddress, Recycler \* recycler, FindHeapObjectFlags flags, RecyclerHeapObjectInfo& heapObject) override sealed

{

return FindHeapObjectImpl<SmallNormalWithBarrierHeapBlockT>(objectAddress, recycler, flags, heapObject);

}

protected:

SmallNormalWithBarrierHeapBlockT(HeapBucket \* bucket, ushort objectSize, ushort objectCount, HeapBlockType heapBlockType) :

SmallNormalHeapBlockT(bucket, objectSize, objectCount, heapBlockType)

{}

};

#endif

extern template class SmallNormalHeapBlockT<SmallAllocationBlockAttributes>;

extern template class SmallNormalHeapBlockT<MediumAllocationBlockAttributes>;

typedef SmallNormalHeapBlockT<SmallAllocationBlockAttributes> SmallNormalHeapBlock;

typedef SmallNormalHeapBlockT<MediumAllocationBlockAttributes> MediumNormalHeapBlock;

#ifdef RECYCLER\_WRITE\_BARRIER

extern template class SmallNormalWithBarrierHeapBlockT<SmallAllocationBlockAttributes>;

extern template class SmallNormalWithBarrierHeapBlockT<MediumAllocationBlockAttributes>;

typedef SmallNormalWithBarrierHeapBlockT<SmallAllocationBlockAttributes> SmallNormalWithBarrierHeapBlock;

typedef SmallNormalWithBarrierHeapBlockT<MediumAllocationBlockAttributes> MediumNormalWithBarrierHeapBlock;

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

template <typename TBlockType>

SmallNormalHeapBucketBase<TBlockType>::SmallNormalHeapBucketBase() :

partialHeapBlockList(nullptr),

partialSweptHeapBlockList(nullptr)

{

}

#ifdef DUMP\_FRAGMENTATION\_STATS

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::AggregateBucketStats(HeapBucketStats& stats)

{

\_\_super::AggregateBucketStats(stats);

HeapBlockList::ForEach(partialHeapBlockList, [&stats](SmallHeapBlock\* heapBlock) {

heapBlock->AggregateBlockStats(stats);

});

HeapBlockList::ForEach(partialSweptHeapBlockList, [&stats](SmallHeapBlock\* heapBlock) {

heapBlock->AggregateBlockStats(stats);

});

}

#endif

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::ScanInitialImplicitRoots(Recycler \* recycler)

{

HeapBlockList::ForEach(fullBlockList, [recycler](TBlockType \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

HeapBlockList::ForEach(heapBlockList, [recycler](TBlockType \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#ifdef PARTIAL\_GC\_ENABLED

Assert(recycler->inPartialCollectMode || partialHeapBlockList == nullptr);

if (recycler->inPartialCollectMode)

{

HeapBlockList::ForEach(partialHeapBlockList, [recycler](TBlockType \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#ifdef CONCURRENT\_GC\_ENABLED

HeapBlockList::ForEach(partialSweptHeapBlockList, [recycler](TBlockType \* heapBlock)

{

heapBlock->ScanInitialImplicitRoots(recycler);

});

#endif

}

#endif

}

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::ScanNewImplicitRoots(Recycler \* recycler)

{

\_\_super::ScanNewImplicitRoots(recycler);

#ifdef PARTIAL\_GC\_ENABLED

Assert(recycler->inPartialCollectMode || partialHeapBlockList == nullptr);

// Don't need to scan the partial heap block list for new implicit root as we don't allocate from them

#endif

}

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

template <typename TBlockType>

bool

SmallNormalHeapBucketBase<TBlockType>::RescanObjectsOnPage(TBlockType \* block, char\* pageAddress, char \* blockStartAddress, BVStatic<TBlockAttributes::BitVectorCount> \* heapBlockMarkBits, const uint localObjectSize, uint bucketIndex, \_\_out\_opt bool\* anyObjectRescanned, Recycler \* recycler)

{

RECYCLER\_STATS\_ADD(recycler, markData.rescanPageCount, TBlockAttributes::PageCount);

// By the time we get here, we should have ensured that there's a mark on any page somewhere.

// REVIEW: Worth check on just the page's mark bits?

Assert(!heapBlockMarkBits->IsAllClear());

if (anyObjectRescanned != nullptr)

{

\*anyObjectRescanned = false;

}

Assert((char\*)pageAddress - blockStartAddress < TBlockAttributes::PageCount \* AutoSystemInfo::PageSize);

const uint pageByteOffset = static\_cast<uint>((char\*)pageAddress - blockStartAddress);

uint firstObjectOnPageIndex = pageByteOffset / localObjectSize;

// This is not necessarily the address on the first object that starts on the page

// If the last object on the previous page spans two pages, this is the address of that object

// We do it this way so that we can figure out if we need to rescan the first few bytes of the page

// if the actual first object on this page is not located at the start of the page

char\* const startObjectAddress = blockStartAddress + (firstObjectOnPageIndex \* localObjectSize);

const uint startBitIndex = TBlockType::GetAddressBitIndex(startObjectAddress);

const uint pageStartBitIndex = pageByteOffset >> HeapConstants::ObjectAllocationShift;

Assert(pageByteOffset / AutoSystemInfo::PageSize < USHRT\_MAX);

const ushort pageNumber = static\_cast<const ushort>(pageByteOffset / AutoSystemInfo::PageSize);

const TBlockType::BlockInfo& blockInfoForPage = HeapInfo::GetBlockInfo<TBlockAttributes>(localObjectSize)[pageNumber];

bool lastObjectOnPreviousPageMarked = false;

// Calculate the mark count here since we no longer keep track during marking

uint rescanMarkCount = TBlockType::CalculateMarkCountForPage(heapBlockMarkBits, bucketIndex, pageStartBitIndex);

const uint pageObjectCount = blockInfoForPage.pageObjectCount;

const uint localObjectCount = (TBlockAttributes::PageCount \* AutoSystemInfo::PageSize) / localObjectSize;

// If all objects are marked, rescan whole block at once

if (TBlockType::CanRescanFullBlock() && rescanMarkCount == pageObjectCount)

{

// REVIEW: Can we optimize this more?

if (!recycler->AddMark(pageAddress, AutoSystemInfo::PageSize))

{

// Failed to add to the mark stack due to OOM.

return false;

}

RECYCLER\_STATS\_ADD(recycler, markData.rescanObjectCount, pageObjectCount);

RECYCLER\_STATS\_ADD(recycler, markData.rescanObjectByteCount, localObjectSize \* pageObjectCount);

if (anyObjectRescanned != nullptr)

{

\*anyObjectRescanned = true;

}

return true;

}

if (startObjectAddress != pageAddress)

{

// If the last object on the previous page that spans into the current page is marked,

// we need to count that in the markCount for rescan

Assert(startObjectAddress >= blockStartAddress && startObjectAddress < pageAddress);

lastObjectOnPreviousPageMarked = (heapBlockMarkBits->Test(startBitIndex) == TRUE);

if (lastObjectOnPreviousPageMarked)

{

rescanMarkCount++;

}

}

const uint objectBitDelta = SmallHeapBlockT<TBlockAttributes>::GetObjectBitDeltaForBucketIndex(bucketIndex);

uint rescanCount = 0;

uint objectIndex = firstObjectOnPageIndex;

for (uint bitIndex = startBitIndex; rescanCount < rescanMarkCount; objectIndex++, bitIndex += objectBitDelta)

{

Assert(objectIndex < localObjectCount);

Assert(!HeapInfo::GetInvalidBitVectorForBucket<TBlockAttributes>(bucketIndex)->Test(bitIndex));

if (heapBlockMarkBits->Test(bitIndex))

{

char \* objectAddress = blockStartAddress + objectIndex \* localObjectSize;

if (!TBlockType::RescanObject(block, objectAddress, localObjectSize, objectIndex, recycler))

{

// Failed to add to the mark stack due to OOM.

return false;

}

rescanCount++;

}

}

// Mark bits should not have changed during the Rescan

if (startObjectAddress != pageAddress && lastObjectOnPreviousPageMarked)

{

Assert(rescanMarkCount == TBlockType::CalculateMarkCountForPage(heapBlockMarkBits, bucketIndex, pageStartBitIndex) + 1);

}

else

{

Assert(rescanMarkCount == TBlockType::CalculateMarkCountForPage(heapBlockMarkBits, bucketIndex, pageStartBitIndex));

}

#if DBG

// We stopped when we hit the rescanMarkCount.

// Make sure no other objects were marked, otherwise our rescanMarkCount was wrong.

for (uint i = objectIndex + 1; i < blockInfoForPage.lastObjectIndexOnPage; i++)

{

Assert(!heapBlockMarkBits->Test(i \* objectBitDelta));

}

#endif

// Let the caller know if we rescanned anything on this page

if (anyObjectRescanned != nullptr)

{

(\*anyObjectRescanned) = (rescanCount > 0);

}

return true;

}

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::SweepPendingObjects(RecyclerSweep& recyclerSweep)

{

RECYCLER\_SLOW\_CHECK(VerifyHeapBlockCount(recyclerSweep.IsBackground()));

CompileAssert(!IsLeafBucket);

TBlockType \*& pendingSweepList = recyclerSweep.GetPendingSweepBlockList(this);

TBlockType \* const list = pendingSweepList;

Recycler \* const recycler = recyclerSweep.GetRecycler();

bool const partialSweep = recycler->inPartialCollectMode;

if (list)

{

pendingSweepList = nullptr;

if (partialSweep)

{

// We did a partial sweep.

// Blocks in the pendingSweepList are the ones we decided not to reuse.

HeapBlockList::ForEachEditing(list, [this, recycler](TBlockType \* heapBlock)

{

// We are not going to reuse this block.

// SweepMode\_ConcurrentPartial will not actually collect anything, it will just update some state.

// The sweepable objects will be collected in a future Sweep.

// Note, page heap blocks are never swept concurrently

heapBlock->SweepObjects<false, SweepMode\_ConcurrentPartial>(recycler);

// page heap mode should never reach here, so don't check pageheap enabled or not

if (heapBlock->HasFreeObject<false>())

{

// We have pre-existing free objects, so put this in the partialSweptHeapBlockList

heapBlock->SetNextBlock(this->partialSweptHeapBlockList);

this->partialSweptHeapBlockList = heapBlock;

}

else

{

// No free objects, so put in the fullBlockList

heapBlock->SetNextBlock(this->fullBlockList);

this->fullBlockList = heapBlock;

}

});

}

else

{

// We decided not to do a partial sweep.

// Blocks in the pendingSweepList need to have a regular sweep.

TBlockType \* tail = SweepPendingObjects<SweepMode\_Concurrent>(recycler, list);

tail->SetNextBlock(this->heapBlockList);

this->heapBlockList = list;

this->StartAllocationAfterSweep();

}

RECYCLER\_SLOW\_CHECK(VerifyHeapBlockCount(recyclerSweep.IsBackground()));

}

Assert(!this->IsAllocationStopped());

}

template <typename TBlockType>

template <SweepMode mode>

TBlockType \*

SmallNormalHeapBucketBase<TBlockType>::SweepPendingObjects(Recycler \* recycler, TBlockType \* list)

{

TBlockType \* tail;

HeapBlockList::ForEach(list, [recycler, &tail](TBlockType \* heapBlock)

{

// Note, page heap blocks are never swept concurrently

heapBlock->SweepObjects<false, mode>(recycler);

tail = heapBlock;

});

return tail;

}

#endif

#ifdef PARTIAL\_GC\_ENABLED

template <typename TBlockType>

SmallNormalHeapBucketBase<TBlockType>::~SmallNormalHeapBucketBase()

{

DeleteHeapBlockList(this->partialHeapBlockList);

#ifdef CONCURRENT\_GC\_ENABLED

DeleteHeapBlockList(this->partialSweptHeapBlockList);

#endif

}

template <typename TBlockType>

template <class Fn>

void

SmallNormalHeapBucketBase<TBlockType>::SweepPartialReusePages(RecyclerSweep& recyclerSweep, TBlockType \* heapBlockList,

TBlockType \*& reuseBlocklist, TBlockType \*&unusedBlockList, Fn callback)

{

HeapBlockList::ForEachEditing(heapBlockList,

[&recyclerSweep, &reuseBlocklist, &unusedBlockList, callback](TBlockType \* heapBlock)

{

uint expectFreeByteCount;

if (heapBlock->DoPartialReusePage(recyclerSweep, expectFreeByteCount))

{

callback(heapBlock, true);

// Reuse the page

heapBlock->SetNextBlock(reuseBlocklist);

reuseBlocklist = heapBlock;

RECYCLER\_STATS\_ADD(recyclerSweep.GetRecycler(), smallNonLeafHeapBlockPartialReuseBytes[heapBlock->GetHeapBlockType()], expectFreeByteCount);

RECYCLER\_STATS\_INC(recyclerSweep.GetRecycler(), smallNonLeafHeapBlockPartialReuseCount[heapBlock->GetHeapBlockType()]);

}

else

{

// Don't not reuse the page if it don't have much free memory.

callback(heapBlock, false);

heapBlock->SetNextBlock(unusedBlockList);

unusedBlockList = heapBlock;

recyclerSweep.AddUnusedFreeByteCount(expectFreeByteCount);

RECYCLER\_STATS\_ADD(recyclerSweep.GetRecycler(), smallNonLeafHeapBlockPartialUnusedBytes[heapBlock->GetHeapBlockType()], expectFreeByteCount);

RECYCLER\_STATS\_INC(recyclerSweep.GetRecycler(), smallNonLeafHeapBlockPartialUnusedCount[heapBlock->GetHeapBlockType()]);

}

});

}

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::SweepPartialReusePages(RecyclerSweep& recyclerSweep)

{

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount(recyclerSweep.IsBackground()));

Assert(this->GetRecycler()->inPartialCollectMode);

TBlockType \* currentHeapBlockList = this->heapBlockList;

this->heapBlockList = nullptr;

SmallNormalHeapBucketBase<TBlockType>::SweepPartialReusePages(recyclerSweep, currentHeapBlockList, this->heapBlockList,

this->partialHeapBlockList,

[](TBlockType \* heapBlock, bool isReused) {});

#ifdef CONCURRENT\_GC\_ENABLED

// only collect data for pending sweep list but don't sweep yet

// until we have adjusted the heuristics, and SweepPartialReusePages will

// sweep the page that we are going to reuse in thread.

TBlockType \*& pendingSweepList = recyclerSweep.GetPendingSweepBlockList(this);

currentHeapBlockList = pendingSweepList;

pendingSweepList = nullptr;

Recycler \* recycler = recyclerSweep.GetRecycler();

SmallNormalHeapBucketBase<TBlockType>::SweepPartialReusePages(recyclerSweep, currentHeapBlockList, this->heapBlockList,

pendingSweepList,

[recycler](TBlockType \* heapBlock, bool isReused)

{

if (isReused)

{

// Finalizable blocks are always swept in thread, so shouldn't be here

Assert(!heapBlock->IsAnyFinalizableBlock());

// Page heap blocks are never swept concurrently

heapBlock->SweepObjects<false, SweepMode\_InThread>(recycler);

// This block has been counted as concurrently swept, and now we changed our mind

// and sweep it in thread. Remove the count

RECYCLER\_STATS\_DEC(recycler, heapBlockConcurrentSweptCount[heapBlock->GetHeapBlockType()]);

}

}

);

#endif

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount(recyclerSweep.IsBackground()));

this->StartAllocationAfterSweep();

// PARTIALGC-TODO: revisit partial heap blocks to see if they can be put back into use

// since the heuristics limit may be been changed.

}

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::FinishPartialCollect(RecyclerSweep \* recyclerSweep)

{

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount(recyclerSweep != nullptr && recyclerSweep->IsBackground()));

Assert(this->GetRecycler()->inPartialCollectMode);

Assert(recyclerSweep == nullptr || this->IsAllocationStopped());

#ifdef CONCURRENT\_GC\_ENABLED

// Process the partial Swept block and move it to the partial heap block list

TBlockType \* partialSweptList = this->partialSweptHeapBlockList;

if (partialSweptList)

{

this->partialSweptHeapBlockList = nullptr;

TBlockType \* tail = nullptr;

HeapBlockList::ForEach(partialSweptList, [this, &tail](TBlockType \* heapBlock)

{

heapBlock->FinishPartialCollect();

Assert(heapBlock->HasFreeObject());

tail = heapBlock;

});

Assert(tail != nullptr);

tail->SetNextBlock(this->partialHeapBlockList);

this->partialHeapBlockList = partialSweptList;

}

#endif

TBlockType \* currentPartialHeapBlockList = this->partialHeapBlockList;

if (recyclerSweep == nullptr)

{

if (currentPartialHeapBlockList != nullptr)

{

this->partialHeapBlockList = nullptr;

this->AppendAllocableHeapBlockList(currentPartialHeapBlockList);

}

}

else

{

if (currentPartialHeapBlockList != nullptr)

{

this->partialHeapBlockList = nullptr;

TBlockType \* list = this->heapBlockList;

if (list == nullptr)

{

this->heapBlockList = currentPartialHeapBlockList;

}

else

{

// CONCURRENT-TODO: Optimize this?

TBlockType \* tail = HeapBlockList::Tail(this->heapBlockList);

tail->SetNextBlock(currentPartialHeapBlockList);

}

}

if (recyclerSweep->GetPendingSweepBlockList(this) == nullptr)

{

// nothing else to sweep now, we can start allocating now.

this->StartAllocationAfterSweep();

}

}

RECYCLER\_SLOW\_CHECK(this->VerifyHeapBlockCount(recyclerSweep != nullptr && recyclerSweep->IsBackground()));

}

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size))

{

\_\_super::EnumerateObjects(infoBits, CallBackFunction);

HeapBucket::EnumerateObjects(partialHeapBlockList, infoBits, CallBackFunction);

#ifdef CONCURRENT\_GC\_ENABLED

HeapBucket::EnumerateObjects(partialSweptHeapBlockList, infoBits, CallBackFunction);

#endif

}

//------------------------------------------------------------------------------

// Debug and verify functions

//------------------------------------------------------------------------------

#if DBG

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::ResetMarks(ResetMarkFlags flags)

{

Assert(this->partialHeapBlockList == nullptr);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(this->partialSweptHeapBlockList == nullptr);

#endif

\_\_super::ResetMarks(flags);

}

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::SweepVerifyPartialBlocks(Recycler \* recycler, TBlockType \* heapBlockList)

{

// PARTIALGC-TODO: Add assert to ensure nothing in the partialHeapBlockList is free-able

HeapBlockList::ForEach(heapBlockList, [recycler](TBlockType \* heapBlock)

{

heapBlock->SweepVerifyPartialBlock(recycler);

});

}

#endif // DBG

template <typename TBlockType>

template<bool pageheap>

void

SmallNormalHeapBucketBase<TBlockType>::Sweep(RecyclerSweep& recyclerSweep)

{

#if DBG

Recycler \* recycler = recyclerSweep.GetRecycler();

// Don't need sweep the partialHeapBlockList, the partially collected heap block list.

// There should be nothing there that is free-able since the last time we swept

Assert(recyclerSweep.InPartialCollect() || partialHeapBlockList == nullptr);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(recyclerSweep.InPartialCollect() || partialSweptHeapBlockList == nullptr);

#endif

this->SweepVerifyPartialBlocks(recycler, this->partialHeapBlockList);

#endif

BaseT::SweepBucket<pageheap>(recyclerSweep, [](RecyclerSweep& recyclerSweep){});

}

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

template <typename TBlockType>

size\_t

SmallNormalHeapBucketBase<TBlockType>::GetNonEmptyHeapBlockCount(bool checkCount) const

{

size\_t currentHeapBlockCount = \_\_super::GetNonEmptyHeapBlockCount(false);

currentHeapBlockCount += HeapBlockList::Count(partialHeapBlockList);

#ifdef CONCURRENT\_GC\_ENABLED

currentHeapBlockCount += HeapBlockList::Count(partialSweptHeapBlockList);

#endif

RECYCLER\_SLOW\_CHECK(Assert(!checkCount || heapBlockCount == currentHeapBlockCount));

return currentHeapBlockCount;

}

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

template <typename TBlockType>

size\_t

SmallNormalHeapBucketBase<TBlockType>::Check(bool checkCount)

{

size\_t smallHeapBlockCount = \_\_super::Check(false);

Assert(partialHeapBlockList == nullptr || this->GetRecycler()->inPartialCollectMode);

smallHeapBlockCount += HeapInfo::Check(false, false, this->partialHeapBlockList);

#ifdef CONCURRENT\_GC\_ENABLED

Assert(partialSweptHeapBlockList == nullptr || this->GetRecycler()->inPartialCollectMode);

smallHeapBlockCount += HeapInfo::Check(false, false, this->partialSweptHeapBlockList);

#endif

Assert(!checkCount || this->heapBlockCount == smallHeapBlockCount);

return smallHeapBlockCount;

}

#endif // RECYCLER\_SLOW\_CHECK\_ENABLED

#ifdef RECYCLER\_MEMORY\_VERIFY

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::Verify()

{

\_\_super::Verify();

Assert(this->partialHeapBlockList == nullptr || this->GetRecycler()->inPartialCollectMode);

HeapBlockList::ForEach(this->partialHeapBlockList, [](TBlockType \* heapBlock)

{

Assert(heapBlock->HasFreeObject());

heapBlock->Verify();

});

#ifdef CONCURRENT\_GC\_ENABLED

Assert(this->partialSweptHeapBlockList == nullptr || this->GetRecycler()->inPartialCollectMode);

HeapBlockList::ForEach(this->partialSweptHeapBlockList, [](TBlockType \* heapBlock)

{

heapBlock->Verify();

});

#endif

}

#endif // RECYCLER\_MEMORY\_VERIFY

#ifdef RECYCLER\_VERIFY\_MARK

template <typename TBlockType>

void

SmallNormalHeapBucketBase<TBlockType>::VerifyMark()

{

\_\_super::VerifyMark();

HeapBlockList::ForEach(this->partialHeapBlockList, [](TBlockType \* heapBlock)

{

heapBlock->VerifyMark();

});

#ifdef CONCURRENT\_GC\_ENABLED

HeapBlockList::ForEach(this->partialSweptHeapBlockList, [](TBlockType \* heapBlock)

{

heapBlock->VerifyMark();

});

#endif

}

#endif // RECYCLER\_VERIFY\_MARK

#endif // PARTIAL\_GC\_ENABLED

template class SmallNormalHeapBucketBase<SmallNormalHeapBlock>;

template class SmallNormalHeapBucketBase<MediumNormalHeapBlock>;

#ifdef RECYCLER\_WRITE\_BARRIER

template class SmallNormalHeapBucketBase<SmallNormalWithBarrierHeapBlock>;

template class SmallNormalHeapBucketBase<MediumNormalWithBarrierHeapBlock>;

#endif

template class SmallNormalHeapBucketBase<SmallFinalizableHeapBlock>;

template class SmallNormalHeapBucketBase<MediumFinalizableHeapBlock>;

#ifdef RECYCLER\_WRITE\_BARRIER

template class SmallNormalHeapBucketBase<SmallFinalizableWithBarrierHeapBlock>;

template class SmallNormalHeapBucketBase<MediumFinalizableWithBarrierHeapBlock>;

#endif

template void SmallNormalHeapBucketBase<SmallNormalHeapBlock>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallNormalHeapBucketBase<SmallNormalHeapBlock>::Sweep<false>(RecyclerSweep& recyclerSweep);

template void SmallNormalHeapBucketBase<MediumNormalHeapBlock>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallNormalHeapBucketBase<MediumNormalHeapBlock>::Sweep<false>(RecyclerSweep& recyclerSweep);

#ifdef RECYCLER\_WRITE\_BARRIER

template void SmallNormalHeapBucketBase<SmallNormalWithBarrierHeapBlock>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallNormalHeapBucketBase<SmallNormalWithBarrierHeapBlock>::Sweep<false>(RecyclerSweep& recyclerSweep);

template void SmallNormalHeapBucketBase<MediumNormalWithBarrierHeapBlock>::Sweep<true>(RecyclerSweep& recyclerSweep);

template void SmallNormalHeapBucketBase<MediumNormalWithBarrierHeapBlock>::Sweep<false>(RecyclerSweep& recyclerSweep);

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

template <typename TBlockType>

class SmallNormalHeapBucketBase : public HeapBucketT<TBlockType>

{

typedef HeapBucketT<TBlockType> BaseT;

public:

typedef typename TBlockType::HeapBlockAttributes TBlockAttributes;

SmallNormalHeapBucketBase();

CompileAssert(!IsLeafBucket);

#ifdef ENABLE\_DEBUG\_CONFIG\_OPTIONS

friend class ScriptMemoryDumper;

#endif

#ifdef DUMP\_FRAGMENTATION\_STATS

void AggregateBucketStats(HeapBucketStats& stats);

#endif

protected:

template <class TBlockAttributes>

friend class HeapBucketGroup;

friend class HeapInfo;

friend class HeapBlockMap32;

void ScanInitialImplicitRoots(Recycler \* recycler);

void ScanNewImplicitRoots(Recycler \* recycler);

#if defined(PARTIAL\_GC\_ENABLED) || defined(CONCURRENT\_GC\_ENABLED)

static bool RescanObjectsOnPage(TBlockType \* block, char \* address, char \* blockStartAddress, BVStatic<TBlockAttributes::BitVectorCount>\* markBits, const uint localObjectSize, uint bucketIndex, \_\_out\_opt bool\* anyObjectRescanned, Recycler\* recycler);

void SweepPendingObjects(RecyclerSweep& recyclerSweep);

template <SweepMode mode>

static TBlockType \* SweepPendingObjects(Recycler \* recycler, TBlockType \* list);

#endif

#ifdef PARTIAL\_GC\_ENABLED

~SmallNormalHeapBucketBase();

template<bool pageheap>

void Sweep(RecyclerSweep& recyclerSweep);

template <class Fn>

static void SweepPartialReusePages(RecyclerSweep& recyclerSweep, TBlockType \* heapBlockList,

TBlockType \*& reuseBlocklist, TBlockType \*&unusedBlockList, Fn callBack);

void SweepPartialReusePages(RecyclerSweep& recyclerSweep);

void FinishPartialCollect(RecyclerSweep \* recyclerSweep);

void EnumerateObjects(ObjectInfoBits infoBits, void (\*CallBackFunction)(void \* address, size\_t size));

#if DBG

void ResetMarks(ResetMarkFlags flags);

static void SweepVerifyPartialBlocks(Recycler \* recycler, TBlockType \* heapBlockList);

#endif

#if DBG || defined(RECYCLER\_SLOW\_CHECK\_ENABLED)

size\_t GetNonEmptyHeapBlockCount(bool checkCount) const;

#endif

#ifdef RECYCLER\_SLOW\_CHECK\_ENABLED

size\_t Check(bool checkCount = true);

friend class HeapBucketT<TBlockType>;

#endif

#ifdef RECYCLER\_MEMORY\_VERIFY

void Verify();

#endif

#ifdef RECYCLER\_VERIFY\_MARK

void VerifyMark();

#endif

protected:

TBlockType \* partialHeapBlockList; // list of blocks that is partially collected

// This list exists to keep track of heap blocks that

// are not full but don't have a large amount of free space

// where allocating from it causing a write watch to be triggered

// is not worth the effort

#ifdef CONCURRENT\_GC\_ENABLED

TBlockType \* partialSweptHeapBlockList; // list of blocks that is partially swept

#endif

#endif

};

template <typename TBlockAttributes>

class SmallNormalHeapBucketT : public SmallNormalHeapBucketBase<SmallNormalHeapBlockT<TBlockAttributes>> {};

typedef SmallNormalHeapBucketT<SmallAllocationBlockAttributes> SmallNormalHeapBucket;

typedef SmallNormalHeapBucketT<MediumAllocationBlockAttributes> MediumNormalHeapBucket;

#ifdef RECYCLER\_WRITE\_BARRIER

template <typename TBlockAttributes>

class SmallNormalWithBarrierHeapBucketT : public SmallNormalHeapBucketBase<SmallNormalWithBarrierHeapBlockT<TBlockAttributes>>

{

public:

void Initialize(HeapInfo \* heapInfo, uint sizeCat)

{

CompileAssert(SmallNormalWithBarrierHeapBucketT::IsLeafBucket == false);

\_\_super::Initialize(heapInfo, sizeCat);

}

};

typedef SmallNormalWithBarrierHeapBucketT<SmallAllocationBlockAttributes> SmallNormalWithBarrierHeapBucket;

typedef SmallNormalWithBarrierHeapBucketT<MediumAllocationBlockAttributes> MediumNormalWithBarrierHeapBucket;

#endif

extern template class SmallNormalHeapBucketBase<SmallNormalHeapBlock>;

extern template class SmallNormalHeapBucketBase<MediumNormalHeapBlock>;

#ifdef RECYCLER\_WRITE\_BARRIER

extern template class SmallNormalHeapBucketBase<SmallNormalWithBarrierHeapBlock>;

extern template class SmallNormalHeapBucketBase<MediumNormalWithBarrierHeapBlock>;

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

#if DBG

#include "common\Int32Math.h"

#include "DataStructures\list.h"

#include "Memory\StressTest.h"

#include <malloc.h>

typedef JsUtil::BaseDictionary<TestObject\*, bool, RecyclerNonLeafAllocator> ObjectTracker\_t;

typedef JsUtil::List<TestObject\*, Recycler> ObjectList\_t;

template<size\_t align> bool IsAligned(void \*p)

{

return (reinterpret\_cast<size\_t>(p) & (align - 1)) == 0;

}

TestObject::TestObject(size\_t \_size, int \_pointerCount) : size(\_size), pointerCount(\_pointerCount)

{

cookie = CalculateCookie();

memset(GetDataPointer(), 0, pointerCount \* sizeof(TestObject\*));

}

size\_t TestObject::CalculateCookie()

{

return reinterpret\_cast<size\_t>(this) ^ (static\_cast<size\_t>(pointerCount) << 12) ^ (size << 24) + 1;

}

void TestObject::CheckCookie()

{

Assert((reinterpret\_cast<size\_t>(this)& (OBJALIGN - 1)) == 0);

Assert(cookie == CalculateCookie());

}

TestObject \*TestObject::Get(int index)

{

Assert(index < pointerCount);

TestObject \*addr = GetDataPointer()[index];

Assert((reinterpret\_cast<size\_t>(addr) & (OBJALIGN - 1)) == 0);

return addr;

}

void TestObject::Set(int index, TestObject \*val)

{

Assert(index < pointerCount);

GetDataPointer()[index] = val;

}

void TestObject::SetRandom(TestObject \*val)

{

if (pointerCount != 0)

{

Set(rand() % pointerCount, val);

}

}

void TestObject::Add(TestObject \*val)

{

TestObject \*\*data = GetDataPointer();

for (int i = 0; i < pointerCount; ++i)

{

if (data[i] == nullptr/\* || !IsAligned<64>(data[i])\*/)

{

data[i] = val;

break;

}

}

}

void TestObject::ClearOne()

{

CheckCookie();

TestObject \*\*data = GetDataPointer();

for (int i = 0; i < pointerCount; ++i)

{

if (data[i] != nullptr/\* && IsAligned<64>(data[i])\*/)

{

// CreateFalseReferenceRandom(data[i]);

data[i] = nullptr;

break;

}

}

}

void TestObject::Visit(Recycler \*recycler, TestObject \*root)

{

Visit(recycler, root, [](TestObject\*) { });

}

template<class Fn> void TestObject::Visit(Recycler \*recycler, TestObject \*root, Fn fn)

{

// TODO: move these allocations to HeapAllocator.

ObjectTracker\_t \*objectTracker = RecyclerNew(recycler, ObjectTracker\_t, recycler);

ObjectList\_t \*objectList = RecyclerNew(recycler, ObjectList\_t, recycler);

// Prime the list with the first object

objectList->Add(root);

objectTracker->Add(root, true);

int numObjects = 0;

while (objectList->Count() > 0)

{

TestObject \*curr = objectList->Item(0);

objectList->RemoveAt(0);

curr->CheckCookie();

for (int i = 0; i < curr->pointerCount; ++i)

{

TestObject \*obj = curr->Get(i);

if (obj != nullptr /\*&& IsAligned<64>(obj)\*/ && !objectTracker->ContainsKey(obj))

{

objectTracker->Add(obj, true);

objectList->Add(obj);

}

}

++numObjects;

}

objectTracker->Map([&](TestObject \* val, bool) {

fn(val);

});

}

TestObject\* TestObject::Create(Recycler \*recycler, int pointerCount, size\_t extraBytes, CreateOptions options)

{

size\_t size = sizeof(TestObject)+pointerCount \* sizeof(TestObject\*) + extraBytes;

if (options == NormalObj)

{

return RecyclerNewPlus(recycler, size, TestObject, size, pointerCount);

}

else if (options == LeafObj)

{

Assert(pointerCount == 0);

return RecyclerNewPlusLeaf(recycler, size, TestObject, size, pointerCount);

}

else

{

Assert(false);

return nullptr;

}

}

void TestObject::CreateFalseReferenceRandom(TestObject \*val)

{

char \*addr = reinterpret\_cast<char\*>(val);

addr += 32;

SetRandom(reinterpret\_cast<TestObject\*>(addr));

}

StressTester::StressTester(Recycler \*\_recycler) : recycler(\_recycler)

{

uint seed = (uint)time(NULL);

printf("Random seed: %u\n", seed);

srand(seed);

}

size\_t StressTester::GetRandomSize()

{

int i = rand() % 5;

switch (i)

{

case 0: return 0;

case 1: return rand() % 16;

case 2: return rand() % 4096;

case 3: return rand() % 16384;

case 4: return rand();

default:

Assert(false);

return 0;

}

}

TestObject\* StressTester::CreateLinkedList()

{

TestObject \*root = TestObject::Create(recycler, 1, GetRandomSize());

TestObject \*curr = root;

int length = rand() % MaxLinkedListLength;

for (int i = 0; i < length; ++i)

{

CreateOptions options = (i == length - 1) ? LeafObj : NormalObj;

TestObject \*next = TestObject::Create(recycler, options == LeafObj ? 0 : 1, GetRandomSize());

curr->Add(next);

curr = next;

}

return root;

}

void StressTester::CreateTreeHelper(TestObject \*root, int depth) {

for (int i = 0; i < root->pointerCount; ++i, ++treeTotal)

{

if (depth == 0 || treeTotal > MaxNodesInTree)

{

root->Add(TestObject::Create(recycler, 0, rand(), LeafObj));

}

else

{

TestObject \*newObj = TestObject::Create(recycler, 4, GetRandomSize());

CreateTreeHelper(newObj, depth - 1);

root->Add(newObj);

}

}

};

TestObject\* StressTester::CreateTree()

{

TestObject \*root = TestObject::Create(recycler, 4, GetRandomSize());

treeTotal = 0;

CreateTreeHelper(root, rand() % MaxTreeDepth);

return root;

}

TestObject \*StressTester::CreateRandom()

{

int numObjects = rand() % 5000 + 1;

void \*memory = \_alloca(numObjects \* sizeof(TestObject\*)+OBJALIGN);

TestObject \*\*objs = reinterpret\_cast<TestObject\*\*>(AlignPtr(memory, OBJALIGN));

// Create the objects

for (int i = 0; i < numObjects; ++i)

{

objs[i] = TestObject::Create(recycler, 10, rand());

}

// Create links between objects

for (int i = 0; i < numObjects; ++i)

{

for (int j = 0; j < 5; ++j)

{

objs[i]->SetRandom(objs[rand() % numObjects]);

}

}

return objs[0];

}

void StressTester::Run()

{

const int stackExtraBytes = 1000;

const int stackPointers = 50;

const size\_t sizeRequired = sizeof(TestObject)+stackExtraBytes + stackPointers \* sizeof(TestObject\*) + OBJALIGN;

char memory[sizeRequired];

void \*addr = AlignPtr(memory, OBJALIGN);

TestObject \*stack = new (addr) TestObject(stackExtraBytes, stackPointers);

auto ObjectVisitor = [&](TestObject \*object) {

// Clear out one of the pointers.

if (rand() % 5 == 0)

{

object->ClearOne();

}

// Maybe store a pointer on the stack.

if (rand() % 25 == 0)

{

stack->SetRandom(object);

}

// Maybe add a stack reference to the current object

if (rand() % 25 == 0)

{

object->SetRandom(stack->Get(rand() % stack->pointerCount));

}

};

while (1)

{

TestObject \*root = CreateLinkedList();

TestObject::Visit(recycler, root);

root = CreateTree();

TestObject::Visit(recycler, root, ObjectVisitor);

TestObject::Visit(recycler, root);

root = CreateRandom();

TestObject::Visit(recycler, root, ObjectVisitor);

TestObject::Visit(recycler, root);

TestObject::Visit(recycler, stack, [&](TestObject \*object) {

if (rand() % 10 == 0)

{

object->ClearOne();

}

});

if (rand() % 3 == 0)

{

for (int i = 0; i < stack->pointerCount; ++i)

{

stack->Set(i, nullptr);

}

}

}

}

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#if DBG

// Recycler requires 16/32 byte alignment

const int OBJALIGN = sizeof(void\*) \* 4;

enum CreateOptions

{

NormalObj,

LeafObj

};

template<class T> T AlignPtr(T ptr, size\_t align)

{

return reinterpret\_cast<T>(::Math::Align(reinterpret\_cast<size\_t>(ptr), align));

}

struct TestObject

{

// Full object size

size\_t size;

// Number of pointers to other objects this object potentially has

int pointerCount;

// Hash of part of the object's contents, used for corruption detection

size\_t cookie;

TestObject(size\_t \_size, int \_pointerCount);

size\_t CalculateCookie();

void CheckCookie();

// Sets an object pointer at index indicated

void Set(int index, TestObject \*val);

void SetRandom(TestObject \*val);

void CreateFalseReferenceRandom(TestObject \*val);

// Does a best-effort attempt to add an object pointer to an unused slot

void Add(TestObject \*val);

// Clears the first non-null pointer in the list

void ClearOne();

// Retrieves a pointer

TestObject\* Get(int index);

TestObject\*\* GetDataPointer() { return reinterpret\_cast<TestObject\*\*>(AlignPtr((char\*)this + sizeof(TestObject), OBJALIGN)); }

static void Visit(Recycler \*recycler, TestObject \*root);

template<class Fn> static void Visit(Recycler \*recycler, TestObject \*root, Fn fn);

static TestObject\* Create(Recycler \*recycler, int pointerCount, size\_t extraBytes, CreateOptions options = NormalObj);

};

class StressTester

{

Recycler \*recycler;

static const int MaxLinkedListLength = 100;

static const int MaxTreeDepth = 8;

static const int MaxNodesInTree = 1000;

size\_t GetRandomSize();

TestObject \*CreateRandom();

TestObject\* CreateLinkedList();

TestObject\* CreateTree();

int treeTotal;

void CreateTreeHelper(TestObject \*root, int depth);

public:

StressTester(Recycler \*\_recycler);

void Run();

};

#endif

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

/\*

\* class VirtualAllocWrapper

\*/

LPVOID VirtualAllocWrapper::Alloc(LPVOID lpAddress, size\_t dwSize, DWORD allocationType, DWORD protectFlags, bool isCustomHeapAllocation)

{

Assert(this == nullptr);

LPVOID address = nullptr;

#if defined(\_CONTROL\_FLOW\_GUARD)

DWORD oldProtectFlags;

if (AutoSystemInfo::Data.IsCFGEnabled() && isCustomHeapAllocation)

{

//We do the allocation in two steps - CFG Bitmap in kernel will be created only on allocation with EXECUTE flag.

//We again call VirtualProtect to set to the requested protectFlags.

address = VirtualAlloc(lpAddress, dwSize, allocationType, PAGE\_EXECUTE\_READWRITE | PAGE\_TARGETS\_INVALID);

VirtualProtect(address, dwSize, protectFlags, &oldProtectFlags);

}

else

#endif

{

address = VirtualAlloc(lpAddress, dwSize, allocationType, protectFlags);

}

return address;

}

BOOL VirtualAllocWrapper::Free(LPVOID lpAddress, size\_t dwSize, DWORD dwFreeType)

{

Assert(this == nullptr);

AnalysisAssert(dwFreeType == MEM\_RELEASE || dwFreeType == MEM\_DECOMMIT);

size\_t bytes = (dwFreeType == MEM\_RELEASE)? 0 : dwSize;

return VirtualFree(lpAddress, bytes, dwFreeType);

}

bool

VirtualAllocWrapper::IsPreReservedRegionPresent()

{

return false;

}

LPVOID

VirtualAllocWrapper::GetPreReservedStartAddress()

{

Assert(false);

return nullptr;

}

LPVOID

VirtualAllocWrapper::GetPreReservedEndAddress()

{

Assert(false);

return nullptr;

}

bool

VirtualAllocWrapper::IsInRange(void \* address)

{

Assert(this == nullptr);

return false;

}

/\*

\* class PreReservedVirtualAllocWrapper

\*/

PreReservedVirtualAllocWrapper::PreReservedVirtualAllocWrapper() :

preReservedStartAddress(nullptr),

cs(4000)

{

freeSegments.SetAll();

}

BOOL

PreReservedVirtualAllocWrapper::Shutdown()

{

Assert(this);

BOOL success = FALSE;

if (IsPreReservedRegionPresent())

{

success = VirtualFree(preReservedStartAddress, 0, MEM\_RELEASE);

PreReservedHeapTrace(L"MEM\_RELEASE the PreReservedSegment. Start Address: 0x%p, Size: 0x%x \* 0x%x bytes", preReservedStartAddress, PreReservedAllocationSegmentCount,

AutoSystemInfo::Data.GetAllocationGranularityPageSize());

if (!success)

{

Assert(false);

}

}

return success;

}

bool

PreReservedVirtualAllocWrapper::IsPreReservedRegionPresent()

{

Assert(this);

return preReservedStartAddress != nullptr;

}

bool

PreReservedVirtualAllocWrapper::IsInRange(void \* address)

{

if (this == nullptr)

{

return false;

}

#if DBG

//Check if the region is in MEM\_COMMIT state.

MEMORY\_BASIC\_INFORMATION memBasicInfo;

size\_t bytes = VirtualQuery(address, &memBasicInfo, sizeof(memBasicInfo));

if (bytes == 0 || memBasicInfo.State != MEM\_COMMIT)

{

AssertMsg(false, "Memory not committed? Checking for uncommitted address region?");

}

#endif

return IsPreReservedRegionPresent() && address >= GetPreReservedStartAddress() && address < GetPreReservedEndAddress();

}

LPVOID

PreReservedVirtualAllocWrapper::GetPreReservedStartAddress()

{

Assert(this);

return preReservedStartAddress;

}

LPVOID

PreReservedVirtualAllocWrapper::GetPreReservedEndAddress()

{

Assert(this);

return (char\*)preReservedStartAddress + (PreReservedAllocationSegmentCount \* AutoSystemInfo::Data.GetAllocationGranularityPageCount() \* AutoSystemInfo::PageSize);

}

/\*

\* LPVOID PreReservedVirtualAllocWrapper::Alloc

\* - Reserves only one big memory region.

\* - Returns an Allocated memory region within this preReserved region with the specified protectFlags.

\* - Tracks the committed pages

\*/

LPVOID PreReservedVirtualAllocWrapper::Alloc(LPVOID lpAddress, size\_t dwSize, DWORD allocationType, DWORD protectFlags, bool isCustomHeapAllocation)

{

Assert(this);

AssertMsg(isCustomHeapAllocation, "PreReservation used for allocations other than CustomHeap?");

AssertMsg(AutoSystemInfo::Data.IsCFGEnabled() || PHASE\_FORCE1(Js::PreReservedHeapAllocPhase), "PreReservation without CFG ?");

Assert((allocationType & MEM\_COMMIT) != 0);

Assert(dwSize != 0);

{

AutoCriticalSection autocs(&this->cs);

if (preReservedStartAddress == NULL)

{

//PreReserve a (bigger) segment

size\_t bytes = PreReservedAllocationSegmentCount \* AutoSystemInfo::Data.GetAllocationGranularityPageSize();

#if defined(\_CONTROL\_FLOW\_GUARD)

if (AutoSystemInfo::Data.IsCFGEnabled())

{

preReservedStartAddress = VirtualAlloc(NULL, bytes, MEM\_RESERVE, PAGE\_READWRITE);

PreReservedHeapTrace(L"Reserving PreReservedSegment For the first time(CFG Enabled). Address: 0x%p\n", preReservedStartAddress);

}

else

#endif

if (PHASE\_FORCE1(Js::PreReservedHeapAllocPhase))

{

//This code is used where CFG is not available, but still PreReserve optimization for CFG can be tested

preReservedStartAddress = VirtualAlloc(NULL, bytes, MEM\_RESERVE, protectFlags);

PreReservedHeapTrace(L"Reserving PreReservedSegment For the first time(CFG Non-Enabled). Address: 0x%p\n", preReservedStartAddress);

}

}

//Return nullptr, if no space to Reserve

if (preReservedStartAddress == NULL)

{

PreReservedHeapTrace(L"No space to pre-reserve memory with %d pages. Returning NULL\n", PreReservedAllocationSegmentCount \* AutoSystemInfo::Data.GetAllocationGranularityPageCount());

return nullptr;

}

char \* addressToCommit = nullptr;

uint freeSegmentsBVIndex = BVInvalidIndex;

size\_t requestedNumOfSegments = dwSize / (AutoSystemInfo::Data.GetAllocationGranularityPageSize());

Assert(requestedNumOfSegments <= MAXUINT32);

if (lpAddress == nullptr)

{

Assert(requestedNumOfSegments != 0);

AssertMsg(dwSize % AutoSystemInfo::Data.GetAllocationGranularityPageSize() == 0, "dwSize should be aligned with Allocation Granularity");

do

{

freeSegmentsBVIndex = freeSegments.GetNextBit(freeSegmentsBVIndex + 1);

//Return nullptr, if we don't have free/decommit pages to allocate

if ((freeSegments.Length() - freeSegmentsBVIndex < requestedNumOfSegments) ||

freeSegmentsBVIndex == BVInvalidIndex)

{

PreReservedHeapTrace(L"No more space to commit in PreReserved Memory region.\n");

return nullptr;

}

} while (!freeSegments.TestRange(freeSegmentsBVIndex, static\_cast<uint>(requestedNumOfSegments)));

uint offset = freeSegmentsBVIndex \* AutoSystemInfo::Data.GetAllocationGranularityPageSize();

addressToCommit = (char\*) preReservedStartAddress + offset;

//Check if the region is not already in MEM\_COMMIT state.

MEMORY\_BASIC\_INFORMATION memBasicInfo;

size\_t bytes = VirtualQuery(addressToCommit, &memBasicInfo, sizeof(memBasicInfo));

if (bytes == 0

|| memBasicInfo.RegionSize < requestedNumOfSegments \* AutoSystemInfo::Data.GetAllocationGranularityPageSize()

|| memBasicInfo.State == MEM\_COMMIT

)

{

CustomHeap\_BadPageState\_fatal\_error((ULONG\_PTR)this);

return nullptr;

}

}

else

{

//Check If the lpAddress is within the range of the preReserved Memory Region

Assert(((char\*) lpAddress) >= (char\*) preReservedStartAddress || ((char\*) lpAddress + dwSize) < GetPreReservedEndAddress());

addressToCommit = (char\*) lpAddress;

freeSegmentsBVIndex = (uint) ((addressToCommit - (char\*) preReservedStartAddress) / AutoSystemInfo::Data.GetAllocationGranularityPageSize());

#if DBG

uint numOfSegments = (uint)ceil((double)dwSize / (double)AutoSystemInfo::Data.GetAllocationGranularityPageSize());

Assert(numOfSegments != 0);

Assert(freeSegmentsBVIndex + numOfSegments - 1 < freeSegments.Length());

Assert(!freeSegments.TestRange(freeSegmentsBVIndex, numOfSegments));

#endif

}

AssertMsg(freeSegmentsBVIndex < PreReservedAllocationSegmentCount, "Invalid BitVector index calculation?");

AssertMsg(dwSize % AutoSystemInfo::PageSize == 0, "COMMIT is managed at AutoSystemInfo::PageSize granularity");

char \* commitedAddress = nullptr;

#if defined(\_CONTROL\_FLOW\_GUARD)

if (AutoSystemInfo::Data.IsCFGEnabled())

{

DWORD oldProtect;

commitedAddress = (char \*) VirtualAlloc(addressToCommit, dwSize, MEM\_COMMIT, PAGE\_EXECUTE\_READWRITE | PAGE\_TARGETS\_INVALID);

AssertMsg(commitedAddress != nullptr, "If no space to allocate, then how did we fetch this address from the tracking bit vector?");

VirtualProtect(commitedAddress, dwSize, protectFlags, &oldProtect);

AssertMsg(oldProtect == (PAGE\_EXECUTE\_READWRITE), "CFG Bitmap gets allocated and bits will be set to invalid only upon passing these flags.");

}

else

#endif

{

commitedAddress = (char \*) VirtualAlloc(addressToCommit, dwSize, MEM\_COMMIT, protectFlags);

}

//Keep track of the committed pages within the preReserved Memory Region

if (lpAddress == nullptr && commitedAddress != nullptr)

{

Assert(commitedAddress == addressToCommit);

Assert(requestedNumOfSegments != 0);

freeSegments.ClearRange(freeSegmentsBVIndex, static\_cast<uint>(requestedNumOfSegments));

}

PreReservedHeapTrace(L"MEM\_COMMIT: StartAddress: 0x%p of size: 0x%x \* 0x%x bytes \n", commitedAddress, requestedNumOfSegments, AutoSystemInfo::Data.GetAllocationGranularityPageSize());

return commitedAddress;

}

}

/\*

\* PreReservedVirtualAllocWrapper::Free

\* - Doesn't actually release the pages to the CPU.

\* - It Decommits the page (memory region within the preReserved Region)

\* - Update the tracking of the committed pages.

\*/

BOOL

PreReservedVirtualAllocWrapper::Free(LPVOID lpAddress, size\_t dwSize, DWORD dwFreeType)

{

Assert(this);

{

AutoCriticalSection autocs(&this->cs);

if (dwSize == 0)

{

Assert(false);

return FALSE;

}

if (preReservedStartAddress == nullptr)

{

Assert(false);

return FALSE;

}

Assert(dwSize % AutoSystemInfo::PageSize == 0);

#pragma warning(suppress: 6250)

BOOL success = VirtualFree(lpAddress, dwSize, MEM\_DECOMMIT);

size\_t requestedNumOfSegments = dwSize / AutoSystemInfo::Data.GetAllocationGranularityPageSize();

Assert(requestedNumOfSegments <= MAXUINT32);

if (success)

{

PreReservedHeapTrace(L"MEM\_DECOMMIT: Address: 0x%p of size: 0x%x bytes\n", lpAddress, dwSize);

}

if (success && (dwFreeType & MEM\_RELEASE) != 0)

{

Assert((uintptr\_t) lpAddress >= (uintptr\_t) preReservedStartAddress);

AssertMsg(((uintptr\_t)lpAddress & (AutoSystemInfo::Data.GetAllocationGranularityPageCount() - 1)) == 0, "Not aligned with Allocation Granularity?");

AssertMsg(dwSize % AutoSystemInfo::Data.GetAllocationGranularityPageSize() == 0, "Release size should match the allocation granularity size");

Assert(requestedNumOfSegments != 0);

BVIndex freeSegmentsBVIndex = (BVIndex) (((uintptr\_t) lpAddress - (uintptr\_t) preReservedStartAddress) / AutoSystemInfo::Data.GetAllocationGranularityPageSize());

AssertMsg(freeSegmentsBVIndex < PreReservedAllocationSegmentCount, "Invalid Index ?");

freeSegments.SetRange(freeSegmentsBVIndex, static\_cast<uint>(requestedNumOfSegments));

PreReservedHeapTrace(L"MEM\_RELEASE: Address: 0x%p of size: 0x%x \* 0x%x bytes\n", lpAddress, requestedNumOfSegments, AutoSystemInfo::Data.GetAllocationGranularityPageSize());

}

return success;

}

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

namespace Memory

{

#define PreReservedHeapTrace(...) { \

OUTPUT\_TRACE(Js::PreReservedHeapAllocPhase, \_\_VA\_ARGS\_\_); \

}

/\*

\* VirtualAllocWrapper is just a delegator class to call VirtualAlloc and VirtualFree.

\*/

class VirtualAllocWrapper

{

public:

LPVOID Alloc(LPVOID lpAddress, size\_t dwSize, DWORD allocationType, DWORD protectFlags, bool isCustomHeapAllocation = false);

BOOL Free(LPVOID lpAddress, size\_t dwSize, DWORD dwFreeType);

bool IsPreReservedRegionPresent();

bool IsInRange(void \* address);

LPVOID GetPreReservedStartAddress();

LPVOID GetPreReservedEndAddress();

};

/\*

\* PreReservedVirtualAllocWrapper class takes care of Reserving a large memory region initially

\* and then committing mem regions for the size requested.

\* Committed pages are being tracked with a bitVector.

\* Committing memory outside of the preReserved Memory region is not handled by this allocator

\*/

class PreReservedVirtualAllocWrapper

{

public:

#if \_M\_IX86\_OR\_ARM32

static const uint PreReservedAllocationSegmentCount = 256; // (256 \* 64K) == 16 MB, if 64k is the AllocationGranularity

#else // \_M\_X64\_OR\_ARM64

static const uint PreReservedAllocationSegmentCount = 4096; //(4096 \* 64K) == 256MB, if 64k is the AllocationGranularity

#endif

public:

PreReservedVirtualAllocWrapper();

BOOL Shutdown();

LPVOID Alloc(LPVOID lpAddress, size\_t dwSize, DWORD allocationType, DWORD protectFlags, bool isCustomHeapAllocation = false);

BOOL Free(LPVOID lpAddress, size\_t dwSize, DWORD dwFreeType);

bool IsPreReservedRegionPresent();

bool IsInRange(void \* address);

LPVOID GetPreReservedStartAddress();

LPVOID GetPreReservedEndAddress();

private:

BVStatic<PreReservedAllocationSegmentCount> freeSegments;

LPVOID preReservedStartAddress;

CriticalSection cs;

};

template<typename TVirtualAlloc = VirtualAllocWrapper>

class PageAllocatorBase;

template<typename TVirtualAlloc = VirtualAllocWrapper>

class PageSegmentBase;

template<typename TVirtualAlloc = VirtualAllocWrapper>

class SegmentBase;

typedef PageAllocatorBase<> PageAllocator;

typedef PageSegmentBase<> PageSegment;

typedef SegmentBase<> Segment;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

#ifdef \_M\_X64

#include "Memory\amd64\XDataAllocator.h"

#elif defined(\_M\_ARM)

#include "Memory\arm\XDataAllocator.h"

#elif defined(\_M\_ARM64)

#include "Memory\arm64\XDataAllocator.h"

#endif

<?xml version="1.0" encoding="utf-8"?>

<Project DefaultTargets="Build" ToolsVersion="12.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

<Import Condition="'$(ChakraBuildPathImported)'!='true'" Project="$(SolutionDir)Chakra.Build.Paths.props"/>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.ProjectConfiguration.props" />

<PropertyGroup Label="Globals">

<TargetName>Chakra.Common.Util</TargetName>

<ProjectGuid>{6979EC58-7A28-465C-A694-F3323A1F5401}</ProjectGuid>

<RootNamespace>JS</RootNamespace>

<Keyword>Win32Proj</Keyword>

</PropertyGroup>

<PropertyGroup Label="Configuration">

<ConfigurationType>StaticLibrary</ConfigurationType>

</PropertyGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />

<Import Project="$(BuildConfigPropsPath)Chakra.Build.props" />

<PropertyGroup>

<\_ProjectFileVersion>10.0.30319.1</\_ProjectFileVersion>

</PropertyGroup>

<ItemDefinitionGroup>

<ClCompile>

<AdditionalIncludeDirectories>

$(MSBuildThisFileDirectory)..;

%(AdditionalIncludeDirectories)

</AdditionalIncludeDirectories>

<PrecompiledHeader>Use</PrecompiledHeader>

</ClCompile>

</ItemDefinitionGroup>

<ItemGroup>

<ClCompile Include="$(MSBuildThisFileDirectory)Pinned.cpp">

<WholeProgramOptimization>false</WholeProgramOptimization>

<PrecompiledHeader>NotUsing</PrecompiledHeader>

</ClCompile>

</ItemGroup>

<ItemGroup>

<ClInclude Include="Pinned.h" />

</ItemGroup>

<Import Project="$(BuildConfigPropsPath)Chakra.Build.targets" Condition="exists('$(BuildConfigPropsPath)Chakra.Build.targets')"/>

<Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />

</Project>

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "pinned.h"

#define UNREFERENCED\_PARAMETER(x) (x)

void EnterPinnedScope(volatile void\*\* var)

{

UNREFERENCED\_PARAMETER(var);

return;

}

void LeavePinnedScope()

{

return;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#pragma once

// Only need to be used if the pointer is long live within the function and doesn't escape

// NOTE: Currently, there is no way to tell the compiler a local variable need to be on the stack

// and last for a certain life time. But if we take the address of the local and pass it to a non-LTCG

// function the assignment won't be dead stored and the local will not be stack packed away.

// IMPORTANT!!! LeavePinnedScope MUST be called at the end of the desired lifetime of a local

// that you allocate using LEAVE\_PINNED\_SCOPE().

void EnterPinnedScope(volatile void\*\* var);

void LeavePinnedScope();

// These MACROs enforce scoping so LEAVE must be called for each instance of ENTER

#define ENTER\_PINNED\_SCOPE(T, var) \

T \* var; \

EnterPinnedScope((volatile void\*\*)& ## var); \

{

#define LEAVE\_PINNED\_SCOPE() \

LeavePinnedScope(); \

}

;-------------------------------------------------------------------------------------------------------

; Copyright (C) Microsoft. All rights reserved.

; Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

;-------------------------------------------------------------------------------------------------------

;Var arm\_GET\_CURRENT\_FRAME()

;

; This method returns the current value of the frame pointer.

;

OPT 2 ; disable listing

#include "ksarm.h"

OPT 1 ; re-enable listing

TTL Lib\Common\arm\arm\_GET\_CURRENT\_FRAME.asm

EXPORT arm\_GET\_CURRENT\_FRAME

TEXTAREA

LEAF\_ENTRY arm\_GET\_CURRENT\_FRAME

mov r0,r11

bx lr

LEAF\_END arm\_GET\_CURRENT\_FRAME

END

;-------------------------------------------------------------------------------------------------------

; Copyright (C) Microsoft. All rights reserved.

; Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

;-------------------------------------------------------------------------------------------------------

;Var arm64\_GET\_CURRENT\_FRAME()

;

; This method returns the current value of the frame pointer.

;

OPT 2 ; disable listing

#include "ksarm64.h"

OPT 1 ; re-enable listing

TTL Lib\Common\arm64\arm64\_GET\_CURRENT\_FRAME.asm

EXPORT arm64\_GET\_CURRENT\_FRAME

TEXTAREA

LEAF\_ENTRY arm64\_GET\_CURRENT\_FRAME

mov x0,x29

br lr

LEAF\_END arm64\_GET\_CURRENT\_FRAME

END

;-------------------------------------------------------------------------------------------------------

; Copyright (C) Microsoft. All rights reserved.

; Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

;-------------------------------------------------------------------------------------------------------

include ksamd64.inc

\_TEXT SEGMENT

;void amd64\_SAVE\_REGISTERS(registers)

;

; This method pushes the 16 general purpose registers into the passed in array.

; By convention, the stack pointer will always be stored at registers[0]

;

; void\* registers[16];

; amd64\_SAVE\_REGISTERS(registers);

;

amd64\_SAVE\_REGISTERS PROC

mov [rcx+00h], rsp

mov [rcx+08h], rax

mov [rcx+10h], rbx

mov [rcx+18h], rcx

mov [rcx+20h], rdx

mov [rcx+28h], rbp

mov [rcx+30h], rsi

mov [rcx+38h], rdi

mov [rcx+40h], r8

mov [rcx+48h], r9

mov [rcx+50h], r10

mov [rcx+58h], r11

mov [rcx+60h], r12

mov [rcx+68h], r13

mov [rcx+70h], r14

mov [rcx+78h], r15

ret

amd64\_SAVE\_REGISTERS ENDP

\_TEXT ENDS

end

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#include "CommonMemoryPch.h"

// This one works only for x64

#if !defined(\_M\_X64)

CompileAssert(false)

#endif

#include "XDataAllocator.h"

#include "core\DelayLoadLibrary.h"

XDataAllocator::XDataAllocator(BYTE\* address, uint size) :

freeList(nullptr),

start(address),

current(address),

size(size),

pdataEntries(nullptr),

functionTableHandles(nullptr)

{

#ifdef RECYCLER\_MEMORY\_VERIFY

memset(this->start, Recycler::VerifyMemFill, this->size);

#endif

Assert(size > 0);

Assert(address != nullptr);

}

bool XDataAllocator::Initialize(void\* segmentStart, void\* segmentEnd)

{

Assert(segmentEnd > segmentStart);

Assert(this->pdataEntries == nullptr);

Assert(this->functionTableHandles == nullptr);

bool success = true;

this->pdataEntries = HeapNewNoThrowArrayZ(RUNTIME\_FUNCTION, GetTotalPdataCount());

success = this->pdataEntries != nullptr;

if(success && AutoSystemInfo::Data.IsWin8OrLater())

{

this->functionTableHandles = HeapNewNoThrowArrayZ(FunctionTableHandle, GetTotalPdataCount());

success = this->functionTableHandles != nullptr;

}

return success;

}

XDataAllocator::~XDataAllocator()

{

if(this->pdataEntries)

{

if(!AutoSystemInfo::Data.IsWin8OrLater())

{

ushort count = this->GetCurrentPdataCount();

for(ushort i = 0; i < count; i++)

{

RUNTIME\_FUNCTION\* pdata = GetPdataEntry(i);

if(pdata->UnwindInfoAddress != 0)

{

BOOLEAN success = RtlDeleteFunctionTable(pdata);

Assert(success);

}

}

}

HeapDeleteArray(this->GetTotalPdataCount(), this->pdataEntries);

this->pdataEntries = nullptr;

}

if(this->functionTableHandles)

{

Assert(AutoSystemInfo::Data.IsWin8OrLater());

ushort count = this->GetCurrentPdataCount();

for(ushort i = 0; i < count; i++)

{

FunctionTableHandle handle = GetFunctionTableHandle(i);

if(handle)

{

NtdllLibrary::Instance->DeleteGrowableFunctionTable(handle);

}

}

HeapDeleteArray(this->GetTotalPdataCount(), this->functionTableHandles);

this->functionTableHandles = nullptr;

}

ClearFreeList();

}

void XDataAllocator::Delete()

{

HeapDelete(this);

}

bool XDataAllocator::Alloc(ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation)

{

XDataAllocation\* xdata = static\_cast<XDataAllocation\*>(allocation);

Assert(start != nullptr);

Assert(current != nullptr);

Assert(current >= start);

Assert(xdataSize <= XDATA\_SIZE);

Assert(AutoSystemInfo::Data.IsWin8OrLater() || pdataCount == 1);

// Allocate a new xdata entry

if((End() - current) >= XDATA\_SIZE)

{

xdata->address = current;

GetNextPdataEntry(&xdata->pdataIndex);

current += XDATA\_SIZE;

} // try allocating from the free list

else if(freeList)

{

auto entry = freeList;

xdata->address = entry->address;

xdata->pdataIndex = entry->pdataIndex;

this->freeList = entry->next;

HeapDelete(entry);

}

else

{

OUTPUT\_TRACE(Js::XDataAllocatorPhase, L"No space for XDATA.\n");

}

if(xdata->address != nullptr)

{

Register(xdata, functionStart, functionSize);

}

return xdata->address != nullptr;

}

void XDataAllocator::Release(const SecondaryAllocation& allocation)

{

const XDataAllocation& xdata = static\_cast<const XDataAllocation&>(allocation);

Assert(allocation.address);

// Add it to free list

auto freed = HeapNewNoThrowStruct(XDataAllocationEntry);

if(freed)

{

freed->address = xdata.address;

freed->pdataIndex = xdata.pdataIndex;

freed->next = this->freeList;

this->freeList = freed;

}

Assert(this->pdataEntries != nullptr);

// Delete the table

if (AutoSystemInfo::Data.IsWin8OrLater())

{

FunctionTableHandle handle = GetFunctionTableHandle(xdata.pdataIndex);

Assert(handle);

NtdllLibrary::Instance->DeleteGrowableFunctionTable(handle);

functionTableHandles[xdata.pdataIndex] = nullptr;

}

else

{

RUNTIME\_FUNCTION\* pdata = GetPdataEntry(xdata.pdataIndex);

BOOLEAN success = RtlDeleteFunctionTable(pdata);

memset(pdata, 0, sizeof(RUNTIME\_FUNCTION));

Assert(success);

}

#ifdef RECYCLER\_MEMORY\_VERIFY

memset(allocation.address, Recycler::VerifyMemFill, XDATA\_SIZE);

#endif

}

bool XDataAllocator::CanAllocate()

{

return ((End() - current) >= XDATA\_SIZE) || this->freeList;

}

void XDataAllocator::ReleaseAll()

{

#ifdef RECYCLER\_MEMORY\_VERIFY

memset(this->start, Recycler::VerifyMemFill, this->size);

#endif

this->current = this->start;

ClearFreeList();

}

void XDataAllocator::ClearFreeList()

{

XDataAllocationEntry\* next = this->freeList;

XDataAllocationEntry\* entry;

while(next)

{

entry = next;

next = entry->next;

HeapDelete(entry);

}

this->freeList = NULL;

}

void XDataAllocator::Register(XDataAllocation\* const xdata, ULONG\_PTR functionStart, DWORD functionSize)

{

RUNTIME\_FUNCTION\* pdata = this->GetPdataEntry(xdata->pdataIndex);

ULONG\_PTR baseAddress = functionStart;

pdata->BeginAddress = (DWORD)(functionStart - baseAddress);

pdata->EndAddress = (DWORD)(pdata->BeginAddress + functionSize);

pdata->UnwindInfoAddress = (DWORD)((ULONG\_PTR)xdata->address - baseAddress);

BOOLEAN success = FALSE;

if (AutoSystemInfo::Data.IsWin8OrLater())

{

Assert(this->functionTableHandles[xdata->pdataIndex] == NULL);

DWORD status = NtdllLibrary::Instance->AddGrowableFunctionTable(&this->functionTableHandles[xdata->pdataIndex],

pdata,

/\*MaxEntryCount\*/ 1,

/\*Valid entry count\*/ 1,

/\*RangeBase\*/ functionStart,

/\*RangeEnd\*/ functionStart + functionSize);

success = NT\_SUCCESS(status);

if (success)

{

Assert(this->functionTableHandles[xdata->pdataIndex]);

}

}

else

{

success = RtlAddFunctionTable(pdata, 1, functionStart);

}

Js::Throw::CheckAndThrowOutOfMemory(success);

#if DBG

// Validate that the PDATA registration succeeded

ULONG64 imageBase = 0;

RUNTIME\_FUNCTION \*runtimeFunction = RtlLookupFunctionEntry((DWORD64)functionStart, &imageBase, nullptr);

Assert(runtimeFunction != NULL);

#endif

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#if !defined(\_M\_X64)

CompileAssert(false)

#endif

#pragma once

namespace Memory

{

#define XDATA\_SIZE (72)

struct XDataAllocation : public SecondaryAllocation

{

XDataAllocation() :

pdataIndex(0)

{}

bool IsFreed() const

{

return address == nullptr;

}

void Free()

{

address = nullptr;

}

// ---- Data members ---- //

ushort pdataIndex;

};

//

// Allocates xdata and pdata entries for x64 architecture.

//

// xdata

// ------

// x64 architecture requires the xdata to be within 32-bit address range of the jitted code itself

// Hence, for every page segment we have an instance of the xdata allocator that allocates

// xdata entries in some specified non-executable region at the end of the page segment.

//

// pdata

// -------

// XDataAllocator also manages the pdata entries for a the page segment range. It allocates the table of pdata entries

// on the heap to do that.

//

class XDataAllocator sealed : public SecondaryAllocator

{

// -------- Private members ---------/

private:

struct XDataAllocationEntry : XDataAllocation

{

XDataAllocationEntry\* next;

};

BYTE\* start;

BYTE\* current;

uint size;

XDataAllocationEntry\* freeList;

RUNTIME\_FUNCTION\* pdataEntries;

FunctionTableHandle\* functionTableHandles;

// --------- Public functions ---------/

public:

XDataAllocator(BYTE\* address, uint size);

virtual ~XDataAllocator();

bool Initialize(void\* segmentStart, void\* segmentEnd);

void Delete();

bool Alloc(ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation);

void Release(const SecondaryAllocation& address);

void ReleaseAll();

bool CanAllocate();

// -------- Private helpers ---------/

private:

BYTE\* End() { return start + size; }

ushort GetTotalPdataCount()

{

return (ushort)(this->size / XDATA\_SIZE);

}

RUNTIME\_FUNCTION\* GetNextPdataEntry(ushort\* pdataIndex)

{

Assert(this->pdataEntries);

\*pdataIndex = GetCurrentPdataCount();

Assert(\*pdataIndex < GetTotalPdataCount());

RUNTIME\_FUNCTION\* pdata = &(this->pdataEntries[\*pdataIndex]);

return pdata;

}

ushort GetCurrentPdataCount()

{

return (ushort)((current - start) / XDATA\_SIZE);

}

RUNTIME\_FUNCTION\* GetPdataEntry(ushort pdataIndex)

{

Assert(pdataIndex < GetTotalPdataCount());

Assert(this->pdataEntries);

return &(pdataEntries[pdataIndex]);

}

FunctionTableHandle GetFunctionTableHandle(ushort pdataIndex)

{

Assert(pdataIndex < GetTotalPdataCount());

Assert(functionTableHandles);

return functionTableHandles[pdataIndex];

}

void ClearFreeList();

void Register(XDataAllocation\* const xdata, ULONG\_PTR functionStart, DWORD functionSize);

};

}

;-------------------------------------------------------------------------------------------------------

; Copyright (C) Microsoft. All rights reserved.

; Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

;-------------------------------------------------------------------------------------------------------

;void arm\_SAVE\_REGISTERS(void\*)

;

; This method pushes sp, r1-r12 onto an array of 13 DWORDs at r0.

; By convention, the stack pointer is at offset 0

;

; DWORD registers[13];

; arm\_SAVE\_REGISTERS(registers);

;

OPT 2 ; disable listing

#include "ksarm.h"

OPT 1 ; re-enable listing

TTL Lib\Common\Memory\arm\arm\_SAVE\_REGISTERS.asm

EXPORT arm\_SAVE\_REGISTERS

TEXTAREA

LEAF\_ENTRY arm\_SAVE\_REGISTERS

str sp, [r0], #+4

stmia r0, {r1-r12}

bx lr

LEAF\_END arm\_SAVE\_REGISTERS

END

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// This one works only for ARM

#include "CommonMemoryPch.h"

#if !defined(\_M\_ARM)

CompileAssert(false)

#endif

#include "XDataAllocator.h"

#include "core\DelayLoadLibrary.h"

XDataAllocator::XDataAllocator(BYTE\* address, uint size)

{

Assert(size == 0);

}

void XDataAllocator::Delete()

{

HeapDelete(this);

}

bool XDataAllocator::Initialize(void\* segmentStart, void\* segmentEnd)

{

return true;

}

bool XDataAllocator::Alloc(ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation)

{

XDataAllocation\* xdata = static\_cast<XDataAllocation\*>(allocation);

Assert(pdataCount > 0);

Assert(xdataSize >= 0);

Assert(xdata);

DWORD size = GetAllocSize(pdataCount, xdataSize);

BYTE\* alloc = HeapNewNoThrowArray(BYTE, size);

if (alloc != nullptr)

{

xdata->address = alloc;

xdata->xdataSize = xdataSize;

xdata->pdataCount = pdataCount;

return true; //success

}

return false; //fail;

}

void XDataAllocator::Register(XDataAllocation& allocation, DWORD functionStart, DWORD functionSize)

{

RUNTIME\_FUNCTION\* pdataArray = allocation.GetPdataArray();

for(ushort i = 0; i < allocation.pdataCount; i++)

{

RUNTIME\_FUNCTION\* pdata = pdataArray + i;

Assert(pdata->UnwindData != 0);

Assert(pdata->BeginAddress != 0);

pdata->BeginAddress = pdata->BeginAddress - (DWORD)functionStart;

if(pdata->Flag != 1) // if it is not packed unwind data

{

pdata->UnwindData = pdata->UnwindData - (DWORD)functionStart;

}

}

Assert(allocation.functionTable == nullptr);

// Since we do not expect many thunk functions to be created, we are using 1 table/function

// for now. This can be optimized further if needed.

DWORD status = NtdllLibrary::Instance->AddGrowableFunctionTable(&allocation.functionTable,

pdataArray,

/\*MaxEntryCount\*/ allocation.pdataCount,

/\*Valid entry count\*/ allocation.pdataCount,

/\*RangeBase\*/ functionStart,

/\*RangeEnd\*/ functionStart + functionSize);

Js::Throw::CheckAndThrowOutOfMemory(NT\_SUCCESS(status));

}

void XDataAllocator::Release(const SecondaryAllocation& allocation)

{

const XDataAllocation& xdata = static\_cast<const XDataAllocation&>(allocation);

if(xdata.address != nullptr)

{

if(xdata.functionTable)

{

NtdllLibrary::Instance->DeleteGrowableFunctionTable(xdata.functionTable);

}

HeapDeleteArray(GetAllocSize(xdata.pdataCount, xdata.xdataSize), xdata.address);

}

}

bool XDataAllocator::CanAllocate()

{

return true;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#if !defined(\_M\_ARM)

CompileAssert(false)

#endif

struct XDataAllocation sealed : public SecondaryAllocation

{

// ---- Methods ----- //

XDataAllocation() :

pdataCount(0)

, functionTable(NULL)

, xdataSize(0)

{}

RUNTIME\_FUNCTION\* GetPdataArray() const

{

return reinterpret\_cast<RUNTIME\_FUNCTION\*>(address + xdataSize);

}

bool IsFreed() const

{

return address == nullptr;

}

void Free()

{

address = nullptr;

pdataCount = 0;

functionTable = nullptr;

xdataSize = 0;

}

// ---- Data members ---- //

ushort pdataCount; // ARM requires more than 1 pdata/function

FunctionTableHandle functionTable; // stores the handle to the growable function table

ushort xdataSize;

};

//

// Allocates xdata and pdata entries for ARM architecture on the heap. They are freed when released.

//

//

class XDataAllocator sealed : public SecondaryAllocator

{

// -------- Private members ---------/

private:

ushort pdataCount;

FunctionTableHandle\* functionTableHandles;

// --------- Public functions ---------/

public:

XDataAllocator(BYTE\* address, uint size);

bool Initialize(void\* segmentStart, void\* segmentEnd);

void Delete();

bool Alloc(ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation);

void Register(XDataAllocation& allocation, DWORD functionStart, DWORD functionSize);

void Release(const SecondaryAllocation& address);

bool CanAllocate();

DWORD GetAllocSize(ushort pdataCount, ushort xdataSize)

{

return sizeof(RUNTIME\_FUNCTION) \* pdataCount + xdataSize;

}

};

;-------------------------------------------------------------------------------------------------------

; Copyright (C) Microsoft. All rights reserved.

; Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

;-------------------------------------------------------------------------------------------------------

;void arm64\_SAVE\_REGISTERS(void\*)

;

; This method pushes sp and x1-x12 onto an array of 13 ULONG\_PTR at x0.

; By convention, sp is at offset 0

; TODO: From talking to SaRavind, the commented out register copies below

; likely need to be uncommented out

;

; ULONG\_PTR registers[13];

; arm64\_SAVE\_REGISTERS(registers);

;

OPT 2 ; disable listing

#include "ksarm64.h"

OPT 1 ; re-enable listing

TTL Lib\Common\Memory\arm64\arm64\_SAVE\_REGISTERS.asm

EXPORT arm64\_SAVE\_REGISTERS

TEXTAREA

LEAF\_ENTRY arm64\_SAVE\_REGISTERS

; Can't use sp with stp so mov to a volatile register

; and then store onto passed in array

mov x16, sp

str x16, [x0, #0x00]

str x1, [x0, #0x08]

stp x2, x3, [x0, #0x10]

stp x4, x5, [x0, #0x20]

stp x6, x7, [x0, #0x30]

stp x8, x9, [x0, #0x40]

stp x10, x11, [x0, #0x50]

str x12, [x0, #0x60]

;stp x13, x14, [x0, #0x70]

;stp x15, x17, [x0, #0x80]

;stp x18, x19, [x0, #0x90]

;stp x20, x21, [x0, #0xA0]

;stp x22, x23, [x0, #0xB0]

;stp x24, x25, [x0, #0xC0]

;stp x26, x27, [x0, #0xD0]

;str x28, [x0, #0xE0]

br lr

LEAF\_END arm64\_SAVE\_REGISTERS

END

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// This one works only for ARM64

#include "StdAfx.h"

#if !defined(\_M\_ARM64)

CompileAssert(false)

#endif

XDataAllocator::XDataAllocator(BYTE\* address, uint size)

{

Assert(size == 0);

}

void XDataAllocator::Delete()

{

HeapDelete(this);

}

bool XDataAllocator::Initialize(void\* segmentStart, void\* segmentEnd)

{

return true;

}

bool XDataAllocator::Alloc(ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation)

{

XDataAllocation\* xdata = static\_cast<XDataAllocation\*>(allocation);

Assert(pdataCount > 0);

Assert(xdataSize >= 0);

Assert(xdata);

bool success = true;

DWORD size = GetAllocSize(pdataCount, xdataSize);

BYTE\* alloc = HeapNewNoThrowArray(BYTE, size);

if(alloc != null)

{

xdata->address = alloc;

xdata->xdataSize = xdataSize;

xdata->pdataCount = pdataCount;

}

return success;

}

void XDataAllocator::Register(XDataAllocation& allocation, ULONG\_PTR functionStart, DWORD functionSize)

{

RUNTIME\_FUNCTION\* pdataArray = allocation.GetPdataArray();

for(ushort i = 0; i < allocation.pdataCount; i++)

{

RUNTIME\_FUNCTION\* pdata = pdataArray + i;

Assert(pdata->UnwindData != 0);

Assert(pdata->BeginAddress != 0);

pdata->BeginAddress = (DWORD)((ULONG\_PTR)pdata->BeginAddress - functionStart);

if(pdata->Flag != 1) // if it is not packed unwind data

{

pdata->UnwindData = (DWORD)((ULONG\_PTR)pdata->UnwindData - functionStart);

}

}

Assert(allocation.functionTable == null);

// Since we do not expect many thunk functions to be created, we are using 1 table/function

// for now. This can be optimized further if needed.

DWORD status = NtdllLibrary::Instance->AddGrowableFunctionTable(&allocation.functionTable,

pdataArray,

/\*MaxEntryCount\*/ allocation.pdataCount,

/\*Valid entry count\*/ allocation.pdataCount,

/\*RangeBase\*/ functionStart,

/\*RangeEnd\*/ functionStart + functionSize);

Assert(NT\_SUCCESS(status));

}

void XDataAllocator::Release(const SecondaryAllocation& allocation)

{

const XDataAllocation& xdata = static\_cast<const XDataAllocation&>(allocation);

if(xdata.address != null)

{

if(xdata.functionTable)

{

NtdllLibrary::Instance->DeleteGrowableFunctionTable(xdata.functionTable);

}

HeapDeleteArray(GetAllocSize(xdata.pdataCount, xdata.xdataSize), xdata.address);

}

}

bool XDataAllocator::CanAllocate()

{

return true;

}

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

#if !defined(\_M\_ARM64)

CompileAssert(false)

#endif

struct XDataAllocation sealed : public SecondaryAllocation

{

// ---- Methods ----- //

XDataAllocation() :

pdataCount(0)

, functionTable(NULL)

, xdataSize(0)

{}

RUNTIME\_FUNCTION\* GetPdataArray() const

{

return reinterpret\_cast<RUNTIME\_FUNCTION\*>(address + xdataSize);

}

bool IsFreed() const

{

return address == nullptr;

}

void Free()

{

address = nullptr;

pdataCount = 0;

functionTable = nullptr;

xdataSize = 0;

}

// ---- Data members ---- //

ushort pdataCount; // ARM requires more than 1 pdata/function

FunctionTableHandle functionTable; // stores the handle to the growable function table

ushort xdataSize;

};

//

// Allocates xdata and pdata entries for ARM architecture on the heap. They are freed when released.

//

//

class XDataAllocator sealed : public SecondaryAllocator

{

// -------- Private members ---------/

private:

ushort pdataCount;

FunctionTableHandle\* functionTableHandles;

// --------- Public functions ---------/

public:

XDataAllocator(BYTE\* address, uint size) { }

bool Initialize(\_In\_ void\* segmentStart, \_In\_ void\* segmentEnd) { \_\_debugbreak(); return 0; }

void Delete() { \_\_debugbreak(); }

bool Alloc(ULONG\_PTR functionStart, DWORD functionSize, ushort pdataCount, ushort xdataSize, SecondaryAllocation\* allocation) { \_\_debugbreak(); return 0; }

void Register(XDataAllocation& allocation, ULONG\_PTR functionStart, DWORD functionSize) { \_\_debugbreak(); }

void Release(const SecondaryAllocation& address) { \_\_debugbreak(); }

bool CanAllocate() { \_\_debugbreak(); return 0; }

DWORD GetAllocSize(ushort pdataCount, ushort xdataSize)

{

return sizeof(RUNTIME\_FUNCTION) \* pdataCount + xdataSize;

}

};

::-------------------------------------------------------------------------------------------------------

:: Copyright (C) Microsoft. All rights reserved.

:: Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

::-------------------------------------------------------------------------------------------------------

call %\*

cd %JSCRIPT\_ROOT%

build -cz -dir inetcore\jscript\manifests;inetcore\jscript\lib\author;inetcore\jscript\lib\backend;inetcore\jscript\lib\common;inetcore\jscript\lib\parser;inetcore\jscript\lib\runtime\bytecode;inetcore\jscript\lib\runtime\math;inetcore\jscript\lib\runtime\language;inetcore\jscript\lib\runtime\library;inetcore\jscript\lib\runtime\types;inetcore\jscript\lib\winrt;inetcore\jscript\dll\jscript\test;inetcore\jscript\exe\common;inetcore\jscript\exe\jshost\release

::-------------------------------------------------------------------------------------------------------

:: Copyright (C) Microsoft. All rights reserved.

:: Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

::-------------------------------------------------------------------------------------------------------

@echo off

setlocal

set \_FILE=

set \_HASERROR=0

:ParseArg

IF "%1" == "" (

goto :DoneParse

)

IF "%1" == "build" (

set \_BUILD=1

goto :Parse1

)

IF EXIST "%1" (

set \_FILE=%\_FILE% %1

goto :Parse1

)

echo ERROR: Invalid parameter %1

exit /B 1

:Parse1

shift /1

goto :ParseArg

:DoneParse

set \_RZARG=%\_RazzleArguments%

set \_RZARG=%\_RZARG:amd64=%

set \_RZARG=%\_RZARG:x86=%

set \_RZARG=%\_RZARG:chk=%

set \_RZARG=%\_RZARG:fre=%

IF "%\_BUILD%" == "1" (

echo Building x86chk

start /WAIT /I %WINDIR%\SysWOW64\cmd.exe /C "%~dp0\BuildValidPointersMap.cmd %RazzleToolPath%\razzle %\_RZARG% x86chk"

IF EXIST "%JSCRIPT\_ROOT%\buildchk.err" (

type %JSCRIPT\_ROOT%\buildchk.err

)

echo Building amd64chk

start /WAIT /I %WINDIR%\SysWOW64\cmd.exe /C "%~dp0\BuildValidPointersMap.cmd %RazzleToolPath%\razzle %\_RZARG% amd64chk"

IF EXIST "%JSCRIPT\_ROOT%\buildchk.err" (

type %JSCRIPT\_ROOT%\buildchk.err

)

)

IF "%\_FILE%" == "" (

set "\_FILE=vpm"

)

for /D %%i IN (%\_FILE%) DO (

call :GenerateValidPointersMapHeader %%i

)

exit /B %\_HASERROR%

:GenerateValidPointersMapHeader

set \_BASE\_PATH=%\_NTTREE%

set \_BASE\_PATH=%\_BASE\_PATH:.x86chk=%

set \_BASE\_PATH=%\_BASE\_PATH:.amd64chk=%

set \_BASE\_PATH=%\_BASE\_PATH:.x86fre=%

set \_BASE\_PATH=%\_BASE\_PATH:.amd64fre=%

set \_BASE\_PATH=%\_BASE\_PATH:.armfre=%

set \_BASE\_PATH=%\_BASE\_PATH:.armfre=%

echo Generating %1.32b.h

call :Generate %1 x86 %1.32b.h

echo Generating %1.64b.h

call :Generate %1 amd64 %1.64b.h

exit /B 0

:Generate

sd edit %3 > nul

echo "// do nothing" > dummy.js

%\_BASE\_PATH%.%2chk\jscript\jshost -GenerateValidPointersMapHeader:%3 dummy.js

if "%errorlevel%" NEQ "0" (

Echo %1: Error generating ValidPointersMap header. Ensure %3 writable.

set \_HASERROR=1

) ELSE (

Echo ValidPointersMap header generated. Please rebuild to incorporate the new definition.

)

del dummy.js

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// Generated via jshost -GenerateValidPointersMapHeader

// Target platforms: 32bit - x86 & arm

#if USE\_STATIC\_VPM

const ushort HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::validPointersBuffer[HeapConstants::BucketCount][HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::rowSize] =

{

{

0x0000, 0x0001, 0x0002, 0x0003, 0x0004, 0x0005, 0x0006, 0x0007, 0x0008, 0x0009, 0x000A, 0x000B, 0x000C, 0x000D, 0x000E, 0x000F, 0x0010, 0x0011, 0x0012, 0x0013, 0x0014, 0x0015, 0x0016, 0x0017, 0x0018, 0x0019, 0x001A, 0x001B, 0x001C, 0x001D, 0x001E, 0x001F, 0x0020, 0x0021, 0x0022, 0x0023, 0x0024, 0x0025, 0x0026, 0x0027, 0x0028, 0x0029, 0x002A, 0x002B, 0x002C, 0x002D, 0x002E, 0x002F, 0x0030, 0x0031, 0x0032, 0x0033, 0x0034, 0x0035, 0x0036, 0x0037, 0x0038, 0x0039, 0x003A, 0x003B, 0x003C, 0x003D, 0x003E, 0x003F, 0x0040, 0x0041, 0x0042, 0x0043, 0x0044, 0x0045, 0x0046, 0x0047, 0x0048, 0x0049, 0x004A, 0x004B, 0x004C, 0x004D, 0x004E, 0x004F, 0x0050, 0x0051, 0x0052, 0x0053, 0x0054, 0x0055, 0x0056, 0x0057, 0x0058, 0x0059, 0x005A, 0x005B, 0x005C, 0x005D, 0x005E, 0x005F, 0x0060, 0x0061, 0x0062, 0x0063, 0x0064, 0x0065, 0x0066, 0x0067, 0x0068, 0x0069, 0x006A, 0x006B, 0x006C, 0x006D, 0x006E, 0x006F, 0x0070, 0x0071, 0x0072, 0x0073, 0x0074, 0x0075, 0x0076, 0x0077, 0x0078, 0x0079, 0x007A, 0x007B, 0x007C, 0x007D, 0x007E, 0x007F, 0x0080, 0x0081, 0x0082, 0x0083, 0x0084, 0x0085, 0x0086, 0x0087, 0x0088, 0x0089, 0x008A, 0x008B, 0x008C, 0x008D, 0x008E, 0x008F, 0x0090, 0x0091, 0x0092, 0x0093, 0x0094, 0x0095, 0x0096, 0x0097, 0x0098, 0x0099, 0x009A, 0x009B, 0x009C, 0x009D, 0x009E, 0x009F, 0x00A0, 0x00A1, 0x00A2, 0x00A3, 0x00A4, 0x00A5, 0x00A6, 0x00A7, 0x00A8, 0x00A9, 0x00AA, 0x00AB, 0x00AC, 0x00AD, 0x00AE, 0x00AF, 0x00B0, 0x00B1, 0x00B2, 0x00B3, 0x00B4, 0x00B5, 0x00B6, 0x00B7, 0x00B8, 0x00B9, 0x00BA, 0x00BB, 0x00BC, 0x00BD, 0x00BE, 0x00BF, 0x00C0, 0x00C1, 0x00C2, 0x00C3, 0x00C4, 0x00C5, 0x00C6, 0x00C7, 0x00C8, 0x00C9, 0x00CA, 0x00CB, 0x00CC, 0x00CD, 0x00CE, 0x00CF, 0x00D0, 0x00D1, 0x00D2, 0x00D3, 0x00D4, 0x00D5, 0x00D6, 0x00D7, 0x00D8, 0x00D9, 0x00DA, 0x00DB, 0x00DC, 0x00DD, 0x00DE, 0x00DF, 0x00E0, 0x00E1, 0x00E2, 0x00E3, 0x00E4, 0x00E5, 0x00E6, 0x00E7, 0x00E8, 0x00E9, 0x00EA, 0x00EB, 0x00EC, 0x00ED, 0x00EE, 0x00EF, 0x00F0, 0x00F1, 0x00F2, 0x00F3, 0x00F4, 0x00F5, 0x00F6, 0x00F7, 0x00F8, 0x00F9, 0x00FA, 0x00FB, 0x00FC, 0x00FD, 0x00FE, 0x00FF, 0x0100, 0x0101, 0x0102, 0x0103, 0x0104, 0x0105, 0x0106, 0x0107, 0x0108, 0x0109, 0x010A, 0x010B, 0x010C, 0x010D, 0x010E, 0x010F, 0x0110, 0x0111, 0x0112, 0x0113, 0x0114, 0x0115, 0x0116, 0x0117, 0x0118, 0x0119, 0x011A, 0x011B, 0x011C, 0x011D, 0x011E, 0x011F, 0x0120, 0x0121, 0x0122, 0x0123, 0x0124, 0x0125, 0x0126, 0x0127, 0x0128, 0x0129, 0x012A, 0x012B, 0x012C, 0x012D, 0x012E, 0x012F, 0x0130, 0x0131, 0x0132, 0x0133, 0x0134, 0x0135, 0x0136, 0x0137, 0x0138, 0x0139, 0x013A, 0x013B, 0x013C, 0x013D, 0x013E, 0x013F, 0x0140, 0x0141, 0x0142, 0x0143, 0x0144, 0x0145, 0x0146, 0x0147, 0x0148, 0x0149, 0x014A, 0x014B, 0x014C, 0x014D, 0x014E, 0x014F, 0x0150, 0x0151, 0x0152, 0x0153, 0x0154, 0x0155, 0x0156, 0x0157, 0x0158, 0x0159, 0x015A, 0x015B, 0x015C, 0x015D, 0x015E, 0x015F, 0x0160, 0x0161, 0x0162, 0x0163, 0x0164, 0x0165, 0x0166, 0x0167, 0x0168, 0x0169, 0x016A, 0x016B, 0x016C, 0x016D, 0x016E, 0x016F, 0x0170, 0x0171, 0x0172, 0x0173, 0x0174, 0x0175, 0x0176, 0x0177, 0x0178, 0x0179, 0x017A, 0x017B, 0x017C, 0x017D, 0x017E, 0x017F, 0x0180, 0x0181, 0x0182, 0x0183, 0x0184, 0x0185, 0x0186, 0x0187, 0x0188, 0x0189, 0x018A, 0x018B, 0x018C, 0x018D, 0x018E, 0x018F, 0x0190, 0x0191, 0x0192, 0x0193, 0x0194, 0x0195, 0x0196, 0x0197, 0x0198, 0x0199, 0x019A, 0x019B, 0x019C, 0x019D, 0x019E, 0x019F, 0x01A0, 0x01A1, 0x01A2, 0x01A3, 0x01A4, 0x01A5, 0x01A6, 0x01A7, 0x01A8, 0x01A9, 0x01AA, 0x01AB, 0x01AC, 0x01AD, 0x01AE, 0x01AF, 0x01B0, 0x01B1, 0x01B2, 0x01B3, 0x01B4, 0x01B5, 0x01B6, 0x01B7, 0x01B8, 0x01B9, 0x01BA, 0x01BB, 0x01BC, 0x01BD, 0x01BE, 0x01BF, 0x01C0, 0x01C1, 0x01C2, 0x01C3, 0x01C4, 0x01C5, 0x01C6, 0x01C7, 0x01C8, 0x01C9, 0x01CA, 0x01CB, 0x01CC, 0x01CD, 0x01CE, 0x01CF, 0x01D0, 0x01D1, 0x01D2, 0x01D3, 0x01D4, 0x01D5, 0x01D6, 0x01D7, 0x01D8, 0x01D9, 0x01DA, 0x01DB, 0x01DC, 0x01DD, 0x01DE, 0x01DF, 0x01E0, 0x01E1, 0x01E2, 0x01E3, 0x01E4, 0x01E5, 0x01E6, 0x01E7, 0x01E8, 0x01E9, 0x01EA, 0x01EB, 0x01EC, 0x01ED, 0x01EE, 0x01EF, 0x01F0, 0x01F1, 0x01F2, 0x01F3, 0x01F4, 0x01F5, 0x01F6, 0x01F7, 0x01F8, 0x01F9, 0x01FA, 0x01FB, 0x01FC, 0x01FD, 0x01FE, 0x01FF, 0x0000, 0x0001, 0x0002, 0x0003, 0x0004, 0x0005, 0x0006, 0x0007, 0x0008, 0x0009, 0x000A, 0x000B, 0x000C, 0x000D, 0x000E, 0x000F, 0x0010, 0x0011, 0x0012, 0x0013, 0x0014, 0x0015, 0x0016, 0x0017, 0x0018, 0x0019, 0x001A, 0x001B, 0x001C, 0x001D, 0x001E, 0x001F, 0x0020, 0x0021, 0x0022, 0x0023, 0x0024, 0x0025, 0x0026, 0x0027, 0x0028, 0x0029, 0x002A, 0x002B, 0x002C, 0x002D, 0x002E, 0x002F, 0x0030, 0x0031, 0x0032, 0x0033, 0x0034, 0x0035, 0x0036, 0x0037, 0x0038, 0x0039, 0x003A, 0x003B, 0x003C, 0x003D, 0x003E, 0x003F, 0x0040, 0x0041, 0x0042, 0x0043, 0x0044, 0x0045, 0x0046, 0x0047, 0x0048, 0x0049, 0x004A, 0x004B, 0x004C, 0x004D, 0x004E, 0x004F, 0x0050, 0x0051, 0x0052, 0x0053, 0x0054, 0x0055, 0x0056, 0x0057, 0x0058, 0x0059, 0x005A, 0x005B, 0x005C, 0x005D, 0x005E, 0x005F, 0x0060, 0x0061, 0x0062, 0x0063, 0x0064, 0x0065, 0x0066, 0x0067, 0x0068, 0x0069, 0x006A, 0x006B, 0x006C, 0x006D, 0x006E, 0x006F, 0x0070, 0x0071, 0x0072, 0x0073, 0x0074, 0x0075, 0x0076, 0x0077, 0x0078, 0x0079, 0x007A, 0x007B, 0x007C, 0x007D, 0x007E, 0x007F, 0x0080, 0x0081, 0x0082, 0x0083, 0x0084, 0x0085, 0x0086, 0x0087, 0x0088, 0x0089, 0x008A, 0x008B, 0x008C, 0x008D, 0x008E, 0x008F, 0x0090, 0x0091, 0x0092, 0x0093, 0x0094, 0x0095, 0x0096, 0x0097, 0x0098, 0x0099, 0x009A, 0x009B, 0x009C, 0x009D, 0x009E, 0x009F, 0x00A0, 0x00A1, 0x00A2, 0x00A3, 0x00A4, 0x00A5, 0x00A6, 0x00A7, 0x00A8, 0x00A9, 0x00AA, 0x00AB, 0x00AC, 0x00AD, 0x00AE, 0x00AF, 0x00B0, 0x00B1, 0x00B2, 0x00B3, 0x00B4, 0x00B5, 0x00B6, 0x00B7, 0x00B8, 0x00B9, 0x00BA, 0x00BB, 0x00BC, 0x00BD, 0x00BE, 0x00BF, 0x00C0, 0x00C1, 0x00C2, 0x00C3, 0x00C4, 0x00C5, 0x00C6, 0x00C7, 0x00C8, 0x00C9, 0x00CA, 0x00CB, 0x00CC, 0x00CD, 0x00CE, 0x00CF, 0x00D0, 0x00D1, 0x00D2, 0x00D3, 0x00D4, 0x00D5, 0x00D6, 0x00D7, 0x00D8, 0x00D9, 0x00DA, 0x00DB, 0x00DC, 0x00DD, 0x00DE, 0x00DF, 0x00E0, 0x00E1, 0x00E2, 0x00E3, 0x00E4, 0x00E5, 0x00E6, 0x00E7, 0x00E8, 0x00E9, 0x00EA, 0x00EB, 0x00EC, 0x00ED, 0x00EE, 0x00EF, 0x00F0, 0x00F1, 0x00F2, 0x00F3, 0x00F4, 0x00F5, 0x00F6, 0x00F7, 0x00F8, 0x00F9, 0x00FA, 0x00FB, 0x00FC, 0x00FD, 0x00FE, 0x00FF, 0x0100, 0x0101, 0x0102, 0x0103, 0x0104, 0x0105, 0x0106, 0x0107, 0x0108, 0x0109, 0x010A, 0x010B, 0x010C, 0x010D, 0x010E, 0x010F, 0x0110, 0x0111, 0x0112, 0x0113, 0x0114, 0x0115, 0x0116, 0x0117, 0x0118, 0x0119, 0x011A, 0x011B, 0x011C, 0x011D, 0x011E, 0x011F, 0x0120, 0x0121, 0x0122, 0x0123, 0x0124, 0x0125, 0x0126, 0x0127, 0x0128, 0x0129, 0x012A, 0x012B, 0x012C, 0x012D, 0x012E, 0x012F, 0x0130, 0x0131, 0x0132, 0x0133, 0x0134, 0x0135, 0x0136, 0x0137, 0x0138, 0x0139, 0x013A, 0x013B, 0x013C, 0x013D, 0x013E, 0x013F, 0x0140, 0x0141, 0x0142, 0x0143, 0x0144, 0x0145, 0x0146, 0x0147, 0x0148, 0x0149, 0x014A, 0x014B, 0x014C, 0x014D, 0x014E, 0x014F, 0x0150, 0x0151, 0x0152, 0x0153, 0x0154, 0x0155, 0x0156, 0x0157, 0x0158, 0x0159, 0x015A, 0x015B, 0x015C, 0x015D, 0x015E, 0x015F, 0x0160, 0x0161, 0x0162, 0x0163, 0x0164, 0x0165, 0x0166, 0x0167, 0x0168, 0x0169, 0x016A, 0x016B, 0x016C, 0x016D, 0x016E, 0x016F, 0x0170, 0x0171, 0x0172, 0x0173, 0x0174, 0x0175, 0x0176, 0x0177, 0x0178, 0x0179, 0x017A, 0x017B, 0x017C, 0x017D, 0x017E, 0x017F, 0x0180, 0x0181, 0x0182, 0x0183, 0x0184, 0x0185, 0x0186, 0x0187, 0x0188, 0x0189, 0x018A, 0x018B, 0x018C, 0x018D, 0x018E, 0x018F, 0x0190, 0x0191, 0x0192, 0x0193, 0x0194, 0x0195, 0x0196, 0x0197, 0x0198, 0x0199, 0x019A, 0x019B, 0x019C, 0x019D, 0x019E, 0x019F, 0x01A0, 0x01A1, 0x01A2, 0x01A3, 0x01A4, 0x01A5, 0x01A6, 0x01A7, 0x01A8, 0x01A9, 0x01AA, 0x01AB, 0x01AC, 0x01AD, 0x01AE, 0x01AF, 0x01B0, 0x01B1, 0x01B2, 0x01B3, 0x01B4, 0x01B5, 0x01B6, 0x01B7, 0x01B8, 0x01B9, 0x01BA, 0x01BB, 0x01BC, 0x01BD, 0x01BE, 0x01BF, 0x01C0, 0x01C1, 0x01C2, 0x01C3, 0x01C4, 0x01C5, 0x01C6, 0x01C7, 0x01C8, 0x01C9, 0x01CA, 0x01CB, 0x01CC, 0x01CD, 0x01CE, 0x01CF, 0x01D0, 0x01D1, 0x01D2, 0x01D3, 0x01D4, 0x01D5, 0x01D6, 0x01D7, 0x01D8, 0x01D9, 0x01DA, 0x01DB, 0x01DC, 0x01DD, 0x01DE, 0x01DF, 0x01E0, 0x01E1, 0x01E2, 0x01E3, 0x01E4, 0x01E5, 0x01E6, 0x01E7, 0x01E8, 0x01E9, 0x01EA, 0x01EB, 0x01EC, 0x01ED, 0x01EE, 0x01EF, 0x01F0, 0x01F1, 0x01F2, 0x01F3, 0x01F4, 0x01F5, 0x01F6, 0x01F7, 0x01F8, 0x01F9, 0x01FA, 0x01FB, 0x01FC, 0x01FD, 0x01FE, 0x01FF

},

{

0x0000, 0xFFFF, 0x0001, 0xFFFF, 0x0002, 0xFFFF, 0x0003, 0xFFFF, 0x0004, 0xFFFF, 0x0005, 0xFFFF, 0x0006, 0xFFFF, 0x0007, 0xFFFF, 0x0008, 0xFFFF, 0x0009, 0xFFFF, 0x000A, 0xFFFF, 0x000B, 0xFFFF, 0x000C, 0xFFFF, 0x000D, 0xFFFF, 0x000E, 0xFFFF, 0x000F, 0xFFFF, 0x0010, 0xFFFF, 0x0011, 0xFFFF, 0x0012, 0xFFFF, 0x0013, 0xFFFF, 0x0014, 0xFFFF, 0x0015, 0xFFFF, 0x0016, 0xFFFF, 0x0017, 0xFFFF, 0x0018, 0xFFFF, 0x0019, 0xFFFF, 0x001A, 0xFFFF, 0x001B, 0xFFFF, 0x001C, 0xFFFF, 0x001D, 0xFFFF, 0x001E, 0xFFFF, 0x001F, 0xFFFF, 0x0020, 0xFFFF, 0x0021, 0xFFFF, 0x0022, 0xFFFF, 0x0023, 0xFFFF, 0x0024, 0xFFFF, 0x0025, 0xFFFF, 0x0026, 0xFFFF, 0x0027, 0xFFFF, 0x0028, 0xFFFF, 0x0029, 0xFFFF, 0x002A, 0xFFFF, 0x002B, 0xFFFF, 0x002C, 0xFFFF, 0x002D, 0xFFFF, 0x002E, 0xFFFF, 0x002F, 0xFFFF, 0x0030, 0xFFFF, 0x0031, 0xFFFF, 0x0032, 0xFFFF, 0x0033, 0xFFFF, 0x0034, 0xFFFF, 0x0035, 0xFFFF, 0x0036, 0xFFFF, 0x0037, 0xFFFF, 0x0038, 0xFFFF, 0x0039, 0xFFFF, 0x003A, 0xFFFF, 0x003B, 0xFFFF, 0x003C, 0xFFFF, 0x003D, 0xFFFF, 0x003E, 0xFFFF, 0x003F, 0xFFFF, 0x0040, 0xFFFF, 0x0041, 0xFFFF, 0x0042, 0xFFFF, 0x0043, 0xFFFF, 0x0044, 0xFFFF, 0x0045, 0xFFFF, 0x0046, 0xFFFF, 0x0047, 0xFFFF, 0x0048, 0xFFFF, 0x0049, 0xFFFF, 0x004A, 0xFFFF, 0x004B, 0xFFFF, 0x004C, 0xFFFF, 0x004D, 0xFFFF, 0x004E, 0xFFFF, 0x004F, 0xFFFF, 0x0050, 0xFFFF, 0x0051, 0xFFFF, 0x0052, 0xFFFF, 0x0053, 0xFFFF, 0x0054, 0xFFFF, 0x0055, 0xFFFF, 0x0056, 0xFFFF, 0x0057, 0xFFFF, 0x0058, 0xFFFF, 0x0059, 0xFFFF, 0x005A, 0xFFFF, 0x005B, 0xFFFF, 0x005C, 0xFFFF, 0x005D, 0xFFFF, 0x005E, 0xFFFF, 0x005F, 0xFFFF, 0x0060, 0xFFFF, 0x0061, 0xFFFF, 0x0062, 0xFFFF, 0x0063, 0xFFFF, 0x0064, 0xFFFF, 0x0065, 0xFFFF, 0x0066, 0xFFFF, 0x0067, 0xFFFF, 0x0068, 0xFFFF, 0x0069, 0xFFFF, 0x006A, 0xFFFF, 0x006B, 0xFFFF, 0x006C, 0xFFFF, 0x006D, 0xFFFF, 0x006E, 0xFFFF, 0x006F, 0xFFFF, 0x0070, 0xFFFF, 0x0071, 0xFFFF, 0x0072, 0xFFFF, 0x0073, 0xFFFF, 0x0074, 0xFFFF, 0x0075, 0xFFFF, 0x0076, 0xFFFF, 0x0077, 0xFFFF, 0x0078, 0xFFFF, 0x0079, 0xFFFF, 0x007A, 0xFFFF, 0x007B, 0xFFFF, 0x007C, 0xFFFF, 0x007D, 0xFFFF, 0x007E, 0xFFFF, 0x007F, 0xFFFF, 0x0080, 0xFFFF, 0x0081, 0xFFFF, 0x0082, 0xFFFF, 0x0083, 0xFFFF, 0x0084, 0xFFFF, 0x0085, 0xFFFF, 0x0086, 0xFFFF, 0x0087, 0xFFFF, 0x0088, 0xFFFF, 0x0089, 0xFFFF, 0x008A, 0xFFFF, 0x008B, 0xFFFF, 0x008C, 0xFFFF, 0x008D, 0xFFFF, 0x008E, 0xFFFF, 0x008F, 0xFFFF, 0x0090, 0xFFFF, 0x0091, 0xFFFF, 0x0092, 0xFFFF, 0x0093, 0xFFFF, 0x0094, 0xFFFF, 0x0095, 0xFFFF, 0x0096, 0xFFFF, 0x0097, 0xFFFF, 0x0098, 0xFFFF, 0x0099, 0xFFFF, 0x009A, 0xFFFF, 0x009B, 0xFFFF, 0x009C, 0xFFFF, 0x009D, 0xFFFF, 0x009E, 0xFFFF, 0x009F, 0xFFFF, 0x00A0, 0xFFFF, 0x00A1, 0xFFFF, 0x00A2, 0xFFFF, 0x00A3, 0xFFFF, 0x00A4, 0xFFFF, 0x00A5, 0xFFFF, 0x00A6, 0xFFFF, 0x00A7, 0xFFFF, 0x00A8, 0xFFFF, 0x00A9, 0xFFFF, 0x00AA, 0xFFFF, 0x00AB, 0xFFFF, 0x00AC, 0xFFFF, 0x00AD, 0xFFFF, 0x00AE, 0xFFFF, 0x00AF, 0xFFFF, 0x00B0, 0xFFFF, 0x00B1, 0xFFFF, 0x00B2, 0xFFFF, 0x00B3, 0xFFFF, 0x00B4, 0xFFFF, 0x00B5, 0xFFFF, 0x00B6, 0xFFFF, 0x00B7, 0xFFFF, 0x00B8, 0xFFFF, 0x00B9, 0xFFFF, 0x00BA, 0xFFFF, 0x00BB, 0xFFFF, 0x00BC, 0xFFFF, 0x00BD, 0xFFFF, 0x00BE, 0xFFFF, 0x00BF, 0xFFFF, 0x00C0, 0xFFFF, 0x00C1, 0xFFFF, 0x00C2, 0xFFFF, 0x00C3, 0xFFFF, 0x00C4, 0xFFFF, 0x00C5, 0xFFFF, 0x00C6, 0xFFFF, 0x00C7, 0xFFFF, 0x00C8, 0xFFFF, 0x00C9, 0xFFFF, 0x00CA, 0xFFFF, 0x00CB, 0xFFFF, 0x00CC, 0xFFFF, 0x00CD, 0xFFFF, 0x00CE, 0xFFFF, 0x00CF, 0xFFFF, 0x00D0, 0xFFFF, 0x00D1, 0xFFFF, 0x00D2, 0xFFFF, 0x00D3, 0xFFFF, 0x00D4, 0xFFFF, 0x00D5, 0xFFFF, 0x00D6, 0xFFFF, 0x00D7, 0xFFFF, 0x00D8, 0xFFFF, 0x00D9, 0xFFFF, 0x00DA, 0xFFFF, 0x00DB, 0xFFFF, 0x00DC, 0xFFFF, 0x00DD, 0xFFFF, 0x00DE, 0xFFFF, 0x00DF, 0xFFFF, 0x00E0, 0xFFFF, 0x00E1, 0xFFFF, 0x00E2, 0xFFFF, 0x00E3, 0xFFFF, 0x00E4, 0xFFFF, 0x00E5, 0xFFFF, 0x00E6, 0xFFFF, 0x00E7, 0xFFFF, 0x00E8, 0xFFFF, 0x00E9, 0xFFFF, 0x00EA, 0xFFFF, 0x00EB, 0xFFFF, 0x00EC, 0xFFFF, 0x00ED, 0xFFFF, 0x00EE, 0xFFFF, 0x00EF, 0xFFFF, 0x00F0, 0xFFFF, 0x00F1, 0xFFFF, 0x00F2, 0xFFFF, 0x00F3, 0xFFFF, 0x00F4, 0xFFFF, 0x00F5, 0xFFFF, 0x00F6, 0xFFFF, 0x00F7, 0xFFFF, 0x00F8, 0xFFFF, 0x00F9, 0xFFFF, 0x00FA, 0xFFFF, 0x00FB, 0xFFFF, 0x00FC, 0xFFFF, 0x00FD, 0xFFFF, 0x00FE, 0xFFFF, 0x00FF, 0xFFFF, 0x0000, 0x0000, 0x0001, 0x0001, 0x0002, 0x0002, 0x0003, 0x0003, 0x0004, 0x0004, 0x0005, 0x0005, 0x0006, 0x0006, 0x0007, 0x0007, 0x0008, 0x0008, 0x0009, 0x0009, 0x000A, 0x000A, 0x000B, 0x000B, 0x000C, 0x000C, 0x000D, 0x000D, 0x000E, 0x000E, 0x000F, 0x000F, 0x0010, 0x0010, 0x0011, 0x0011, 0x0012, 0x0012, 0x0013, 0x0013, 0x0014, 0x0014, 0x0015, 0x0015, 0x0016, 0x0016, 0x0017, 0x0017, 0x0018, 0x0018, 0x0019, 0x0019, 0x001A, 0x001A, 0x001B, 0x001B, 0x001C, 0x001C, 0x001D, 0x001D, 0x001E, 0x001E, 0x001F, 0x001F, 0x0020, 0x0020, 0x0021, 0x0021, 0x0022, 0x0022, 0x0023, 0x0023, 0x0024, 0x0024, 0x0025, 0x0025, 0x0026, 0x0026, 0x0027, 0x0027, 0x0028, 0x0028, 0x0029, 0x0029, 0x002A, 0x002A, 0x002B, 0x002B, 0x002C, 0x002C, 0x002D, 0x002D, 0x002E, 0x002E, 0x002F, 0x002F, 0x0030, 0x0030, 0x0031, 0x0031, 0x0032, 0x0032, 0x0033, 0x0033, 0x0034, 0x0034, 0x0035, 0x0035, 0x0036, 0x0036, 0x0037, 0x0037, 0x0038, 0x0038, 0x0039, 0x0039, 0x003A, 0x003A, 0x003B, 0x003B, 0x003C, 0x003C, 0x003D, 0x003D, 0x003E, 0x003E, 0x003F, 0x003F, 0x0040, 0x0040, 0x0041, 0x0041, 0x0042, 0x0042, 0x0043, 0x0043, 0x0044, 0x0044, 0x0045, 0x0045, 0x0046, 0x0046, 0x0047, 0x0047, 0x0048, 0x0048, 0x0049, 0x0049, 0x004A, 0x004A, 0x004B, 0x004B, 0x004C, 0x004C, 0x004D, 0x004D, 0x004E, 0x004E, 0x004F, 0x004F, 0x0050, 0x0050, 0x0051, 0x0051, 0x0052, 0x0052, 0x0053, 0x0053, 0x0054, 0x0054, 0x0055, 0x0055, 0x0056, 0x0056, 0x0057, 0x0057, 0x0058, 0x0058, 0x0059, 0x0059, 0x005A, 0x005A, 0x005B, 0x005B, 0x005C, 0x005C, 0x005D, 0x005D, 0x005E, 0x005E, 0x005F, 0x005F, 0x0060, 0x0060, 0x0061, 0x0061, 0x0062, 0x0062, 0x0063, 0x0063, 0x0064, 0x0064, 0x0065, 0x0065, 0x0066, 0x0066, 0x0067, 0x0067, 0x0068, 0x0068, 0x0069, 0x0069, 0x006A, 0x006A, 0x006B, 0x006B, 0x006C, 0x006C, 0x006D, 0x006D, 0x006E, 0x006E, 0x006F, 0x006F, 0x0070, 0x0070, 0x0071, 0x0071, 0x0072, 0x0072, 0x0073, 0x0073, 0x0074, 0x0074, 0x0075, 0x0075, 0x0076, 0x0076, 0x0077, 0x0077, 0x0078, 0x0078, 0x0079, 0x0079, 0x007A, 0x007A, 0x007B, 0x007B, 0x007C, 0x007C, 0x007D, 0x007D, 0x007E, 0x007E, 0x007F, 0x007F, 0x0080, 0x0080, 0x0081, 0x0081, 0x0082, 0x0082, 0x0083, 0x0083, 0x0084, 0x0084, 0x0085, 0x0085, 0x0086, 0x0086, 0x0087, 0x0087, 0x0088, 0x0088, 0x0089, 0x0089, 0x008A, 0x008A, 0x008B, 0x008B, 0x008C, 0x008C, 0x008D, 0x008D, 0x008E, 0x008E, 0x008F, 0x008F, 0x0090, 0x0090, 0x0091, 0x0091, 0x0092, 0x0092, 0x0093, 0x0093, 0x0094, 0x0094, 0x0095, 0x0095, 0x0096, 0x0096, 0x0097, 0x0097, 0x0098, 0x0098, 0x0099, 0x0099, 0x009A, 0x009A, 0x009B, 0x009B, 0x009C, 0x009C, 0x009D, 0x009D, 0x009E, 0x009E, 0x009F, 0x009F, 0x00A0, 0x00A0, 0x00A1, 0x00A1, 0x00A2, 0x00A2, 0x00A3, 0x00A3, 0x00A4, 0x00A4, 0x00A5, 0x00A5, 0x00A6, 0x00A6, 0x00A7, 0x00A7, 0x00A8, 0x00A8, 0x00A9, 0x00A9, 0x00AA, 0x00AA, 0x00AB, 0x00AB, 0x00AC, 0x00AC, 0x00AD, 0x00AD, 0x00AE, 0x00AE, 0x00AF, 0x00AF, 0x00B0, 0x00B0, 0x00B1, 0x00B1, 0x00B2, 0x00B2, 0x00B3, 0x00B3, 0x00B4, 0x00B4, 0x00B5, 0x00B5, 0x00B6, 0x00B6, 0x00B7, 0x00B7, 0x00B8, 0x00B8, 0x00B9, 0x00B9, 0x00BA, 0x00BA, 0x00BB, 0x00BB, 0x00BC, 0x00BC, 0x00BD, 0x00BD, 0x00BE, 0x00BE, 0x00BF, 0x00BF, 0x00C0, 0x00C0, 0x00C1, 0x00C1, 0x00C2, 0x00C2, 0x00C3, 0x00C3, 0x00C4, 0x00C4, 0x00C5, 0x00C5, 0x00C6, 0x00C6, 0x00C7, 0x00C7, 0x00C8, 0x00C8, 0x00C9, 0x00C9, 0x00CA, 0x00CA, 0x00CB, 0x00CB, 0x00CC, 0x00CC, 0x00CD, 0x00CD, 0x00CE, 0x00CE, 0x00CF, 0x00CF, 0x00D0, 0x00D0, 0x00D1, 0x00D1, 0x00D2, 0x00D2, 0x00D3, 0x00D3, 0x00D4, 0x00D4, 0x00D5, 0x00D5, 0x00D6, 0x00D6, 0x00D7, 0x00D7, 0x00D8, 0x00D8, 0x00D9, 0x00D9, 0x00DA, 0x00DA, 0x00DB, 0x00DB, 0x00DC, 0x00DC, 0x00DD, 0x00DD, 0x00DE, 0x00DE, 0x00DF, 0x00DF, 0x00E0, 0x00E0, 0x00E1, 0x00E1, 0x00E2, 0x00E2, 0x00E3, 0x00E3, 0x00E4, 0x00E4, 0x00E5, 0x00E5, 0x00E6, 0x00E6, 0x00E7, 0x00E7, 0x00E8, 0x00E8, 0x00E9, 0x00E9, 0x00EA, 0x00EA, 0x00EB, 0x00EB, 0x00EC, 0x00EC, 0x00ED, 0x00ED, 0x00EE, 0x00EE, 0x00EF, 0x00EF, 0x00F0, 0x00F0, 0x00F1, 0x00F1, 0x00F2, 0x00F2, 0x00F3, 0x00F3, 0x00F4, 0x00F4, 0x00F5, 0x00F5, 0x00F6, 0x00F6, 0x00F7, 0x00F7, 0x00F8, 0x00F8, 0x00F9, 0x00F9, 0x00FA, 0x00FA, 0x00FB, 0x00FB, 0x00FC, 0x00FC, 0x00FD, 0x00FD, 0x00FE, 0x00FE, 0x00FF, 0x00FF

},

{

0x0000, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0x0066, 0xFFFF, 0xFFFF, 0x0067, 0xFFFF, 0xFFFF, 0x0068, 0xFFFF, 0xFFFF, 0x0069, 0xFFFF, 0xFFFF, 0x006A, 0xFFFF, 0xFFFF, 0x006B, 0xFFFF, 0xFFFF, 0x006C, 0xFFFF, 0xFFFF, 0x006D, 0xFFFF, 0xFFFF, 0x006E, 0xFFFF, 0xFFFF, 0x006F, 0xFFFF, 0xFFFF, 0x0070, 0xFFFF, 0xFFFF, 0x0071, 0xFFFF, 0xFFFF, 0x0072, 0xFFFF, 0xFFFF, 0x0073, 0xFFFF, 0xFFFF, 0x0074, 0xFFFF, 0xFFFF, 0x0075, 0xFFFF, 0xFFFF, 0x0076, 0xFFFF, 0xFFFF, 0x0077, 0xFFFF, 0xFFFF, 0x0078, 0xFFFF, 0xFFFF, 0x0079, 0xFFFF, 0xFFFF, 0x007A, 0xFFFF, 0xFFFF, 0x007B, 0xFFFF, 0xFFFF, 0x007C, 0xFFFF, 0xFFFF, 0x007D, 0xFFFF, 0xFFFF, 0x007E, 0xFFFF, 0xFFFF, 0x007F, 0xFFFF, 0xFFFF, 0x0080, 0xFFFF, 0xFFFF, 0x0081, 0xFFFF, 0xFFFF, 0x0082, 0xFFFF, 0xFFFF, 0x0083, 0xFFFF, 0xFFFF, 0x0084, 0xFFFF, 0xFFFF, 0x0085, 0xFFFF, 0xFFFF, 0x0086, 0xFFFF, 0xFFFF, 0x0087, 0xFFFF, 0xFFFF, 0x0088, 0xFFFF, 0xFFFF, 0x0089, 0xFFFF, 0xFFFF, 0x008A, 0xFFFF, 0xFFFF, 0x008B, 0xFFFF, 0xFFFF, 0x008C, 0xFFFF, 0xFFFF, 0x008D, 0xFFFF, 0xFFFF, 0x008E, 0xFFFF, 0xFFFF, 0x008F, 0xFFFF, 0xFFFF, 0x0090, 0xFFFF, 0xFFFF, 0x0091, 0xFFFF, 0xFFFF, 0x0092, 0xFFFF, 0xFFFF, 0x0093, 0xFFFF, 0xFFFF, 0x0094, 0xFFFF, 0xFFFF, 0x0095, 0xFFFF, 0xFFFF, 0x0096, 0xFFFF, 0xFFFF, 0x0097, 0xFFFF, 0xFFFF, 0x0098, 0xFFFF, 0xFFFF, 0x0099, 0xFFFF, 0xFFFF, 0x009A, 0xFFFF, 0xFFFF, 0x009B, 0xFFFF, 0xFFFF, 0x009C, 0xFFFF, 0xFFFF, 0x009D, 0xFFFF, 0xFFFF, 0x009E, 0xFFFF, 0xFFFF, 0x009F, 0xFFFF, 0xFFFF, 0x00A0, 0xFFFF, 0xFFFF, 0x00A1, 0xFFFF, 0xFFFF, 0x00A2, 0xFFFF, 0xFFFF, 0x00A3, 0xFFFF, 0xFFFF, 0x00A4, 0xFFFF, 0xFFFF, 0x00A5, 0xFFFF, 0xFFFF, 0x00A6, 0xFFFF, 0xFFFF, 0x00A7, 0xFFFF, 0xFFFF, 0x00A8, 0xFFFF, 0xFFFF, 0x00A9, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0066, 0x0066, 0x0066, 0x0067, 0x0067, 0x0067, 0x0068, 0x0068, 0x0068, 0x0069, 0x0069, 0x0069, 0x006A, 0x006A, 0x006A, 0x006B, 0x006B, 0x006B, 0x006C, 0x006C, 0x006C, 0x006D, 0x006D, 0x006D, 0x006E, 0x006E, 0x006E, 0x006F, 0x006F, 0x006F, 0x0070, 0x0070, 0x0070, 0x0071, 0x0071, 0x0071, 0x0072, 0x0072, 0x0072, 0x0073, 0x0073, 0x0073, 0x0074, 0x0074, 0x0074, 0x0075, 0x0075, 0x0075, 0x0076, 0x0076, 0x0076, 0x0077, 0x0077, 0x0077, 0x0078, 0x0078, 0x0078, 0x0079, 0x0079, 0x0079, 0x007A, 0x007A, 0x007A, 0x007B, 0x007B, 0x007B, 0x007C, 0x007C, 0x007C, 0x007D, 0x007D, 0x007D, 0x007E, 0x007E, 0x007E, 0x007F, 0x007F, 0x007F, 0x0080, 0x0080, 0x0080, 0x0081, 0x0081, 0x0081, 0x0082, 0x0082, 0x0082, 0x0083, 0x0083, 0x0083, 0x0084, 0x0084, 0x0084, 0x0085, 0x0085, 0x0085, 0x0086, 0x0086, 0x0086, 0x0087, 0x0087, 0x0087, 0x0088, 0x0088, 0x0088, 0x0089, 0x0089, 0x0089, 0x008A, 0x008A, 0x008A, 0x008B, 0x008B, 0x008B, 0x008C, 0x008C, 0x008C, 0x008D, 0x008D, 0x008D, 0x008E, 0x008E, 0x008E, 0x008F, 0x008F, 0x008F, 0x0090, 0x0090, 0x0090, 0x0091, 0x0091, 0x0091, 0x0092, 0x0092, 0x0092, 0x0093, 0x0093, 0x0093, 0x0094, 0x0094, 0x0094, 0x0095, 0x0095, 0x0095, 0x0096, 0x0096, 0x0096, 0x0097, 0x0097, 0x0097, 0x0098, 0x0098, 0x0098, 0x0099, 0x0099, 0x0099, 0x009A, 0x009A, 0x009A, 0x009B, 0x009B, 0x009B, 0x009C, 0x009C, 0x009C, 0x009D, 0x009D, 0x009D, 0x009E, 0x009E, 0x009E, 0x009F, 0x009F, 0x009F, 0x00A0, 0x00A0, 0x00A0, 0x00A1, 0x00A1, 0x00A1, 0x00A2, 0x00A2, 0x00A2, 0x00A3, 0x00A3, 0x00A3, 0x00A4, 0x00A4, 0x00A4, 0x00A5, 0x00A5, 0x00A5, 0x00A6, 0x00A6, 0x00A6, 0x00A7, 0x00A7, 0x00A7, 0x00A8, 0x00A8, 0x00A8, 0x00A9, 0x00A9, 0x00A9, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0x0066, 0xFFFF, 0xFFFF, 0xFFFF, 0x0067, 0xFFFF, 0xFFFF, 0xFFFF, 0x0068, 0xFFFF, 0xFFFF, 0xFFFF, 0x0069, 0xFFFF, 0xFFFF, 0xFFFF, 0x006A, 0xFFFF, 0xFFFF, 0xFFFF, 0x006B, 0xFFFF, 0xFFFF, 0xFFFF, 0x006C, 0xFFFF, 0xFFFF, 0xFFFF, 0x006D, 0xFFFF, 0xFFFF, 0xFFFF, 0x006E, 0xFFFF, 0xFFFF, 0xFFFF, 0x006F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0070, 0xFFFF, 0xFFFF, 0xFFFF, 0x0071, 0xFFFF, 0xFFFF, 0xFFFF, 0x0072, 0xFFFF, 0xFFFF, 0xFFFF, 0x0073, 0xFFFF, 0xFFFF, 0xFFFF, 0x0074, 0xFFFF, 0xFFFF, 0xFFFF, 0x0075, 0xFFFF, 0xFFFF, 0xFFFF, 0x0076, 0xFFFF, 0xFFFF, 0xFFFF, 0x0077, 0xFFFF, 0xFFFF, 0xFFFF, 0x0078, 0xFFFF, 0xFFFF, 0xFFFF, 0x0079, 0xFFFF, 0xFFFF, 0xFFFF, 0x007A, 0xFFFF, 0xFFFF, 0xFFFF, 0x007B, 0xFFFF, 0xFFFF, 0xFFFF, 0x007C, 0xFFFF, 0xFFFF, 0xFFFF, 0x007D, 0xFFFF, 0xFFFF, 0xFFFF, 0x007E, 0xFFFF, 0xFFFF, 0xFFFF, 0x007F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0065, 0x0066, 0x0066, 0x0066, 0x0066, 0x0067, 0x0067, 0x0067, 0x0067, 0x0068, 0x0068, 0x0068, 0x0068, 0x0069, 0x0069, 0x0069, 0x0069, 0x006A, 0x006A, 0x006A, 0x006A, 0x006B, 0x006B, 0x006B, 0x006B, 0x006C, 0x006C, 0x006C, 0x006C, 0x006D, 0x006D, 0x006D, 0x006D, 0x006E, 0x006E, 0x006E, 0x006E, 0x006F, 0x006F, 0x006F, 0x006F, 0x0070, 0x0070, 0x0070, 0x0070, 0x0071, 0x0071, 0x0071, 0x0071, 0x0072, 0x0072, 0x0072, 0x0072, 0x0073, 0x0073, 0x0073, 0x0073, 0x0074, 0x0074, 0x0074, 0x0074, 0x0075, 0x0075, 0x0075, 0x0075, 0x0076, 0x0076, 0x0076, 0x0076, 0x0077, 0x0077, 0x0077, 0x0077, 0x0078, 0x0078, 0x0078, 0x0078, 0x0079, 0x0079, 0x0079, 0x0079, 0x007A, 0x007A, 0x007A, 0x007A, 0x007B, 0x007B, 0x007B, 0x007B, 0x007C, 0x007C, 0x007C, 0x007C, 0x007D, 0x007D, 0x007D, 0x007D, 0x007E, 0x007E, 0x007E, 0x007E, 0x007F, 0x007F, 0x007F, 0x007F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F

}

};

const BVUnit HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::invalidBitsData[HeapConstants::BucketCount][SmallHeapBlockT<SmallAllocationBlockAttributes>::SmallHeapBlockBitVector::wordCount] = {

{

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

},

{

0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA, 0xAAAAAAAA

},

{

0xB6DB6DB6, 0x6DB6DB6D, 0xDB6DB6DB, 0xB6DB6DB6, 0x6DB6DB6D, 0xDB6DB6DB, 0xB6DB6DB6, 0x6DB6DB6D, 0xDB6DB6DB, 0xB6DB6DB6, 0x6DB6DB6D, 0xDB6DB6DB, 0xB6DB6DB6, 0x6DB6DB6D, 0xDB6DB6DB, 0xF6DB6DB6

},

{

0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE, 0xEEEEEEEE

},

{

0xBDEF7BDE, 0xEF7BDEF7, 0x7BDEF7BD, 0xDEF7BDEF, 0xF7BDEF7B, 0xBDEF7BDE, 0xEF7BDEF7, 0x7BDEF7BD, 0xDEF7BDEF, 0xF7BDEF7B, 0xBDEF7BDE, 0xEF7BDEF7, 0x7BDEF7BD, 0xDEF7BDEF, 0xF7BDEF7B, 0xFDEF7BDE

},

{

0xBEFBEFBE, 0xEFBEFBEF, 0xFBEFBEFB, 0xBEFBEFBE, 0xEFBEFBEF, 0xFBEFBEFB, 0xBEFBEFBE, 0xEFBEFBEF, 0xFBEFBEFB, 0xBEFBEFBE, 0xEFBEFBEF, 0xFBEFBEFB, 0xBEFBEFBE, 0xEFBEFBEF, 0xFBEFBEFB, 0xFEFBEFBE

},

{

0xEFDFBF7E, 0x7EFDFBF7, 0xF7EFDFBF, 0xBF7EFDFB, 0xFBF7EFDF, 0xDFBF7EFD, 0xFDFBF7EF, 0xEFDFBF7E, 0x7EFDFBF7, 0xF7EFDFBF, 0xBF7EFDFB, 0xFBF7EFDF, 0xDFBF7EFD, 0xFDFBF7EF, 0xEFDFBF7E, 0xFEFDFBF7

},

{

0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE, 0xFEFEFEFE

},

{

0xF7FBFDFE, 0x7FBFDFEF, 0xFBFDFEFF, 0xBFDFEFF7, 0xFDFEFF7F, 0xDFEFF7FB, 0xFEFF7FBF, 0xEFF7FBFD, 0xFF7FBFDF, 0xF7FBFDFE, 0x7FBFDFEF, 0xFBFDFEFF, 0xBFDFEFF7, 0xFDFEFF7F, 0xDFEFF7FB, 0xFFFF7FBF

},

{

0xBFEFFBFE, 0xEFFBFEFF, 0xFBFEFFBF, 0xFEFFBFEF, 0xFFBFEFFB, 0xBFEFFBFE, 0xEFFBFEFF, 0xFBFEFFBF, 0xFEFFBFEF, 0xFFBFEFFB, 0xBFEFFBFE, 0xEFFBFEFF, 0xFBFEFFBF, 0xFEFFBFEF, 0xFFBFEFFB, 0xFFEFFBFE

},

{

0xFFBFF7FE, 0xFF7FEFFD, 0xFEFFDFFB, 0xFDFFBFF7, 0xFBFF7FEF, 0xF7FEFFDF, 0xEFFDFFBF, 0xDFFBFF7F, 0xBFF7FEFF, 0x7FEFFDFF, 0xFFDFFBFF, 0xFFBFF7FE, 0xFF7FEFFD, 0xFEFFDFFB, 0xFDFFBFF7, 0xFFFF7FEF

},

{

0xFEFFEFFE, 0xEFFEFFEF, 0xFFEFFEFF, 0xFEFFEFFE, 0xEFFEFFEF, 0xFFEFFEFF, 0xFEFFEFFE, 0xEFFEFFEF, 0xFFEFFEFF, 0xFEFFEFFE, 0xEFFEFFEF, 0xFFEFFEFF, 0xFEFFEFFE, 0xEFFEFFEF, 0xFFEFFEFF, 0xFFFFEFFE

},

{

0xFBFFDFFE, 0xFFEFFF7F, 0xF7FFBFFD, 0xFFDFFEFF, 0xEFFF7FFB, 0xFFBFFDFF, 0xDFFEFFF7, 0xFF7FFBFF, 0xBFFDFFEF, 0xFEFFF7FF, 0x7FFBFFDF, 0xFDFFEFFF, 0xFFF7FFBF, 0xFBFFDFFE, 0xFFEFFF7F, 0xFFFFBFFD

},

{

0xEFFFBFFE, 0xFEFFFBFF, 0xFFEFFFBF, 0xBFFEFFFB, 0xFBFFEFFF, 0xFFBFFEFF, 0xFFFBFFEF, 0xEFFFBFFE, 0xFEFFFBFF, 0xFFEFFFBF, 0xBFFEFFFB, 0xFBFFEFFF, 0xFFBFFEFF, 0xFFFBFFEF, 0xEFFFBFFE, 0xFFFFFBFF

},

{

0xBFFF7FFE, 0xEFFFDFFF, 0xFBFFF7FF, 0xFEFFFDFF, 0xFFBFFF7F, 0xFFEFFFDF, 0xFFFBFFF7, 0x7FFEFFFD, 0xDFFFBFFF, 0xF7FFEFFF, 0xFDFFFBFF, 0xFF7FFEFF, 0xFFDFFFBF, 0xFFF7FFEF, 0xFFFDFFFB, 0xFFFF7FFE

},

{

0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE, 0xFFFEFFFE

},

{

0xFFFDFFFE, 0xFFF7FFFB, 0xFFDFFFEF, 0xFF7FFFBF, 0xFDFFFEFF, 0xF7FFFBFF, 0xDFFFEFFF, 0x7FFFBFFF, 0xFFFEFFFF, 0xFFFBFFFD, 0xFFEFFFF7, 0xFFBFFFDF, 0xFEFFFF7F, 0xFBFFFDFF, 0xEFFFF7FF, 0xFFFFDFFF

},

{

0xFFFBFFFE, 0xFFBFFFEF, 0xFBFFFEFF, 0xBFFFEFFF, 0xFFFEFFFF, 0xFFEFFFFB, 0xFEFFFFBF, 0xEFFFFBFF, 0xFFFFBFFF, 0xFFFBFFFE, 0xFFBFFFEF, 0xFBFFFEFF, 0xBFFFEFFF, 0xFFFEFFFF, 0xFFEFFFFB, 0xFFFFFFBF

},

{

0xFFF7FFFE, 0xFDFFFFBF, 0x7FFFEFFF, 0xFFFBFFFF, 0xFEFFFFDF, 0xBFFFF7FF, 0xFFFDFFFF, 0xFF7FFFEF, 0xDFFFFBFF, 0xFFFEFFFF, 0xFFBFFFF7, 0xEFFFFDFF, 0xFFFF7FFF, 0xFFDFFFFB, 0xF7FFFEFF, 0xFFFFFFFF

},

{

0xFFEFFFFE, 0xEFFFFEFF, 0xFFFEFFFF, 0xFEFFFFEF, 0xFFFFEFFF, 0xFFEFFFFE, 0xEFFFFEFF, 0xFFFEFFFF, 0xFEFFFFEF, 0xFFFFEFFF, 0xFFEFFFFE, 0xEFFFFEFF, 0xFFFEFFFF, 0xFEFFFFEF, 0xFFFFEFFF, 0xFFFFFFFE

},

{

0xFFDFFFFE, 0x7FFFFBFF, 0xFFEFFFFF, 0xBFFFFDFF, 0xFFF7FFFF, 0xDFFFFEFF, 0xFFFBFFFF, 0xEFFFFF7F, 0xFFFDFFFF, 0xF7FFFFBF, 0xFFFEFFFF, 0xFBFFFFDF, 0xFFFF7FFF, 0xFDFFFFEF, 0xFFFFBFFF, 0xFFFFFFF7

},

{

0xFFBFFFFE, 0xFFFFEFFF, 0xFEFFFFFB, 0xFFFFBFFF, 0xFBFFFFEF, 0xFFFEFFFF, 0xEFFFFFBF, 0xFFFBFFFF, 0xBFFFFEFF, 0xFFEFFFFF, 0xFFFFFBFF, 0xFFBFFFFE, 0xFFFFEFFF, 0xFEFFFFFB, 0xFFFFBFFF, 0xFFFFFFEF

},

{

0xFF7FFFFE, 0xFFFFBFFF, 0xEFFFFFDF, 0xFFF7FFFF, 0xFFFFFBFF, 0xFEFFFFFD, 0xFFFF7FFF, 0xDFFFFFBF, 0xFFEFFFFF, 0xFFFFF7FF, 0xFDFFFFFB, 0xFFFEFFFF, 0xBFFFFF7F, 0xFFDFFFFF, 0xFFFFEFFF, 0xFFFFFFF7

},

{

0xFEFFFFFE, 0xFFFEFFFF, 0xFFFFFEFF, 0xFEFFFFFE, 0xFFFEFFFF, 0xFFFFFEFF, 0xFEFFFFFE, 0xFFFEFFFF, 0xFFFFFEFF, 0xFEFFFFFE, 0xFFFEFFFF, 0xFFFFFEFF, 0xFEFFFFFE, 0xFFFEFFFF, 0xFFFFFEFF, 0xFFFFFFFE

},

{

0xFDFFFFFE, 0xFFFBFFFF, 0xFFFFF7FF, 0xDFFFFFEF, 0xFFBFFFFF, 0xFFFF7FFF, 0xFFFFFEFF, 0xFBFFFFFD, 0xFFF7FFFF, 0xFFFFEFFF, 0xBFFFFFDF, 0xFF7FFFFF, 0xFFFEFFFF, 0xFFFFFDFF, 0xF7FFFFFB, 0xFFFFFFFF

},

{

0xFBFFFFFE, 0xFFEFFFFF, 0xFFFFBFFF, 0xFFFFFEFF, 0xEFFFFFFB, 0xFFBFFFFF, 0xFFFEFFFF, 0xFFFFFBFF, 0xBFFFFFEF, 0xFEFFFFFF, 0xFFFBFFFF, 0xFFFFEFFF, 0xFFFFFFBF, 0xFBFFFFFE, 0xFFEFFFFF, 0xFFFFFFFF

},

{

0xF7FFFFFE, 0xFFBFFFFF, 0xFFFDFFFF, 0xFFFFEFFF, 0xFFFFFF7F, 0xDFFFFFFB, 0xFEFFFFFF, 0xFFF7FFFF, 0xFFFFBFFF, 0xFFFFFDFF, 0x7FFFFFEF, 0xFBFFFFFF, 0xFFDFFFFF, 0xFFFEFFFF, 0xFFFFF7FF, 0xFFFFFFFF

},

{

0xEFFFFFFE, 0xFEFFFFFF, 0xFFEFFFFF, 0xFFFEFFFF, 0xFFFFEFFF, 0xFFFFFEFF, 0xFFFFFFEF, 0xEFFFFFFE, 0xFEFFFFFF, 0xFFEFFFFF, 0xFFFEFFFF, 0xFFFFEFFF, 0xFFFFFEFF, 0xFFFFFFEF, 0xEFFFFFFE, 0xFFFFFFFF

},

{

0xDFFFFFFE, 0xFBFFFFFF, 0xFF7FFFFF, 0xFFEFFFFF, 0xFFFDFFFF, 0xFFFFBFFF, 0xFFFFF7FF, 0xFFFFFEFF, 0xFFFFFFDF, 0x7FFFFFFB, 0xEFFFFFFF, 0xFDFFFFFF, 0xFFBFFFFF, 0xFFF7FFFF, 0xFFFEFFFF, 0xFFFFFFFF

},

{

0xBFFFFFFE, 0xEFFFFFFF, 0xFBFFFFFF, 0xFEFFFFFF, 0xFFBFFFFF, 0xFFEFFFFF, 0xFFFBFFFF, 0xFFFEFFFF, 0xFFFFBFFF, 0xFFFFEFFF, 0xFFFFFBFF, 0xFFFFFEFF, 0xFFFFFFBF, 0xFFFFFFEF, 0xFFFFFFFB, 0xFFFFFFFE

},

{

0x7FFFFFFE, 0xBFFFFFFF, 0xDFFFFFFF, 0xEFFFFFFF, 0xF7FFFFFF, 0xFBFFFFFF, 0xFDFFFFFF, 0xFEFFFFFF, 0xFF7FFFFF, 0xFFBFFFFF, 0xFFDFFFFF, 0xFFEFFFFF, 0xFFF7FFFF, 0xFFFBFFFF, 0xFFFDFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE, 0xFFFFFFFE

}

};

// The following is used to construct the InvalidBitsTable statically without forcing BVStatic to be an aggregate

const HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::InvalidBitsTable \* const HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::invalidBitsBuffers =

reinterpret\_cast<const HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::InvalidBitsTable \*>(&HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::invalidBitsData);

const SmallHeapBlockT<SmallAllocationBlockAttributes>::BlockInfo HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::blockInfoBuffer[SmallAllocationBlockAttributes::BucketCount][SmallAllocationBlockAttributes::PageCount] = {

// Bucket: 0, Size: 16

{

{ 0x00FF, 0x0100 },

{ 0x01FF, 0x0100 }

},

// Bucket: 1, Size: 32

{

{ 0x007F, 0x0080 },

{ 0x00FF, 0x0080 }

},

// Bucket: 2, Size: 48

{

{ 0x0055, 0x0056 },

{ 0x00A9, 0x0054 }

},

// Bucket: 3, Size: 64

{

{ 0x003F, 0x0040 },

{ 0x007F, 0x0040 }

},

// Bucket: 4, Size: 80

{

{ 0x0033, 0x0034 },

{ 0x0065, 0x0032 }

},

// Bucket: 5, Size: 96

{

{ 0x002A, 0x002B },

{ 0x0054, 0x002A }

},

// Bucket: 6, Size: 112

{

{ 0x0024, 0x0025 },

{ 0x0048, 0x0024 }

},

// Bucket: 7, Size: 128

{

{ 0x001F, 0x0020 },

{ 0x003F, 0x0020 }

},

// Bucket: 8, Size: 144

{

{ 0x001C, 0x001D },

{ 0x0037, 0x001B }

},

// Bucket: 9, Size: 160

{

{ 0x0019, 0x001A },

{ 0x0032, 0x0019 }

},

// Bucket: 10, Size: 176

{

{ 0x0017, 0x0018 },

{ 0x002D, 0x0016 }

},

// Bucket: 11, Size: 192

{

{ 0x0015, 0x0016 },

{ 0x0029, 0x0014 }

},

// Bucket: 12, Size: 208

{

{ 0x0013, 0x0014 },

{ 0x0026, 0x0013 }

},

// Bucket: 13, Size: 224

{

{ 0x0012, 0x0013 },

{ 0x0023, 0x0011 }

},

// Bucket: 14, Size: 240

{

{ 0x0011, 0x0012 },

{ 0x0021, 0x0010 }

},

// Bucket: 15, Size: 256

{

{ 0x000F, 0x0010 },

{ 0x001F, 0x0010 }

},

// Bucket: 16, Size: 272

{

{ 0x000F, 0x0010 },

{ 0x001D, 0x000E }

},

// Bucket: 17, Size: 288

{

{ 0x000E, 0x000F },

{ 0x001B, 0x000D }

},

// Bucket: 18, Size: 304

{

{ 0x000D, 0x000E },

{ 0x0019, 0x000C }

},

// Bucket: 19, Size: 320

{

{ 0x000C, 0x000D },

{ 0x0018, 0x000C }

},

// Bucket: 20, Size: 336

{

{ 0x000C, 0x000D },

{ 0x0017, 0x000B }

},

// Bucket: 21, Size: 352

{

{ 0x000B, 0x000C },

{ 0x0016, 0x000B }

},

// Bucket: 22, Size: 368

{

{ 0x000B, 0x000C },

{ 0x0015, 0x000A }

},

// Bucket: 23, Size: 384

{

{ 0x000A, 0x000B },

{ 0x0014, 0x000A }

},

// Bucket: 24, Size: 400

{

{ 0x000A, 0x000B },

{ 0x0013, 0x0009 }

},

// Bucket: 25, Size: 416

{

{ 0x0009, 0x000A },

{ 0x0012, 0x0009 }

},

// Bucket: 26, Size: 432

{

{ 0x0009, 0x000A },

{ 0x0011, 0x0008 }

},

// Bucket: 27, Size: 448

{

{ 0x0009, 0x000A },

{ 0x0011, 0x0008 }

},

// Bucket: 28, Size: 464

{

{ 0x0008, 0x0009 },

{ 0x0010, 0x0008 }

},

// Bucket: 29, Size: 480

{

{ 0x0008, 0x0009 },

{ 0x0010, 0x0008 }

},

// Bucket: 30, Size: 496

{

{ 0x0008, 0x0009 },

{ 0x000F, 0x0007 }

},

// Bucket: 31, Size: 512

{

{ 0x0007, 0x0008 },

{ 0x000F, 0x0008 }

}

};

const ushort HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::validPointersBuffer[MediumAllocationBlockAttributes::BucketCount][HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::rowSize] =

{

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003

}

};

const BVUnit HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::invalidBitsData[MediumAllocationBlockAttributes::BucketCount][SmallHeapBlockT<MediumAllocationBlockAttributes>::SmallHeapBlockBitVector::wordCount] = {

{

0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFEFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

},

{

0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFE, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF

}

};

// The following is used to construct the InvalidBitsTable statically without forcing BVStatic to be an aggregate

const HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::InvalidBitsTable \* const HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::invalidBitsBuffers =

reinterpret\_cast<const HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::InvalidBitsTable \*>(&HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::invalidBitsData);

const SmallHeapBlockT<MediumAllocationBlockAttributes>::BlockInfo HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::blockInfoBuffer[MediumAllocationBlockAttributes::BucketCount][MediumAllocationBlockAttributes::PageCount] = {

// Bucket: 0, Size: 768

{

{ 0x0005, 0x0006 },

{ 0x000A, 0x0005 },

{ 0x000F, 0x0005 },

{ 0x0015, 0x0006 },

{ 0x001A, 0x0005 },

{ 0x001F, 0x0005 },

{ 0x0025, 0x0006 },

{ 0x0029, 0x0004 }

},

// Bucket: 1, Size: 1024

{

{ 0x0003, 0x0004 },

{ 0x0007, 0x0004 },

{ 0x000B, 0x0004 },

{ 0x000F, 0x0004 },

{ 0x0013, 0x0004 },

{ 0x0017, 0x0004 },

{ 0x001B, 0x0004 },

{ 0x001F, 0x0004 }

},

// Bucket: 2, Size: 1280

{

{ 0x0003, 0x0004 },

{ 0x0006, 0x0003 },

{ 0x0009, 0x0003 },

{ 0x000C, 0x0003 },

{ 0x000F, 0x0003 },

{ 0x0013, 0x0004 },

{ 0x0016, 0x0003 },

{ 0x0018, 0x0002 }

},

// Bucket: 3, Size: 1536

{

{ 0x0002, 0x0003 },

{ 0x0005, 0x0003 },

{ 0x0007, 0x0002 },

{ 0x000A, 0x0003 },

{ 0x000D, 0x0003 },

{ 0x000F, 0x0002 },

{ 0x0012, 0x0003 },

{ 0x0014, 0x0002 }

},

// Bucket: 4, Size: 1792

{

{ 0x0002, 0x0003 },

{ 0x0004, 0x0002 },

{ 0x0006, 0x0002 },

{ 0x0009, 0x0003 },

{ 0x000B, 0x0002 },

{ 0x000D, 0x0002 },

{ 0x000F, 0x0002 },

{ 0x0011, 0x0002 }

},

// Bucket: 5, Size: 2048

{

{ 0x0001, 0x0002 },

{ 0x0003, 0x0002 },

{ 0x0005, 0x0002 },

{ 0x0007, 0x0002 },

{ 0x0009, 0x0002 },

{ 0x000B, 0x0002 },

{ 0x000D, 0x0002 },

{ 0x000F, 0x0002 }

},

// Bucket: 6, Size: 2304

{

{ 0x0001, 0x0002 },

{ 0x0003, 0x0002 },

{ 0x0005, 0x0002 },

{ 0x0007, 0x0002 },

{ 0x0008, 0x0001 },

{ 0x000A, 0x0002 },

{ 0x000C, 0x0002 },

{ 0x000D, 0x0001 }

},

// Bucket: 7, Size: 2560

{

{ 0x0001, 0x0002 },

{ 0x0003, 0x0002 },

{ 0x0004, 0x0001 },

{ 0x0006, 0x0002 },

{ 0x0007, 0x0001 },

{ 0x0009, 0x0002 },

{ 0x000B, 0x0002 },

{ 0x0000, 0x0000 }

},

// Bucket: 8, Size: 2816

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0004, 0x0002 },

{ 0x0005, 0x0001 },

{ 0x0007, 0x0002 },

{ 0x0008, 0x0001 },

{ 0x000A, 0x0002 },

{ 0x0000, 0x0000 }

},

// Bucket: 9, Size: 3072

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0005, 0x0002 },

{ 0x0006, 0x0001 },

{ 0x0007, 0x0001 },

{ 0x0009, 0x0002 },

{ 0x0000, 0x0000 }

},

// Bucket: 10, Size: 3328

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0006, 0x0002 },

{ 0x0007, 0x0001 },

{ 0x0008, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 11, Size: 3584

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0007, 0x0001 },

{ 0x0008, 0x0001 }

},

// Bucket: 12, Size: 3840

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0007, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 13, Size: 4096

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0007, 0x0001 }

},

// Bucket: 14, Size: 4352

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 15, Size: 4608

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 16, Size: 4864

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 17, Size: 5120

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 18, Size: 5376

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 19, Size: 5632

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 20, Size: 5888

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 21, Size: 6144

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0004, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 22, Size: 6400

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0004, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 23, Size: 6656

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 24, Size: 6912

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 25, Size: 7168

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 26, Size: 7424

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 27, Size: 7680

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 28, Size: 7936

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 29, Size: 8192

{

{ 0x0000, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 }

}

};

#endif // USE\_STATIC\_VPM

//-------------------------------------------------------------------------------------------------------

// Copyright (C) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE.txt file in the project root for full license information.

//-------------------------------------------------------------------------------------------------------

// Generated via jshost -GenerateValidPointersMapHeader

// Target platform: 64bit - amd64 & arm64

#if USE\_STATIC\_VPM

const ushort HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::validPointersBuffer[HeapConstants::BucketCount][HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::rowSize] =

{

{

0x0000, 0x0001, 0x0002, 0x0003, 0x0004, 0x0005, 0x0006, 0x0007, 0x0008, 0x0009, 0x000A, 0x000B, 0x000C, 0x000D, 0x000E, 0x000F, 0x0010, 0x0011, 0x0012, 0x0013, 0x0014, 0x0015, 0x0016, 0x0017, 0x0018, 0x0019, 0x001A, 0x001B, 0x001C, 0x001D, 0x001E, 0x001F, 0x0020, 0x0021, 0x0022, 0x0023, 0x0024, 0x0025, 0x0026, 0x0027, 0x0028, 0x0029, 0x002A, 0x002B, 0x002C, 0x002D, 0x002E, 0x002F, 0x0030, 0x0031, 0x0032, 0x0033, 0x0034, 0x0035, 0x0036, 0x0037, 0x0038, 0x0039, 0x003A, 0x003B, 0x003C, 0x003D, 0x003E, 0x003F, 0x0040, 0x0041, 0x0042, 0x0043, 0x0044, 0x0045, 0x0046, 0x0047, 0x0048, 0x0049, 0x004A, 0x004B, 0x004C, 0x004D, 0x004E, 0x004F, 0x0050, 0x0051, 0x0052, 0x0053, 0x0054, 0x0055, 0x0056, 0x0057, 0x0058, 0x0059, 0x005A, 0x005B, 0x005C, 0x005D, 0x005E, 0x005F, 0x0060, 0x0061, 0x0062, 0x0063, 0x0064, 0x0065, 0x0066, 0x0067, 0x0068, 0x0069, 0x006A, 0x006B, 0x006C, 0x006D, 0x006E, 0x006F, 0x0070, 0x0071, 0x0072, 0x0073, 0x0074, 0x0075, 0x0076, 0x0077, 0x0078, 0x0079, 0x007A, 0x007B, 0x007C, 0x007D, 0x007E, 0x007F, 0x0080, 0x0081, 0x0082, 0x0083, 0x0084, 0x0085, 0x0086, 0x0087, 0x0088, 0x0089, 0x008A, 0x008B, 0x008C, 0x008D, 0x008E, 0x008F, 0x0090, 0x0091, 0x0092, 0x0093, 0x0094, 0x0095, 0x0096, 0x0097, 0x0098, 0x0099, 0x009A, 0x009B, 0x009C, 0x009D, 0x009E, 0x009F, 0x00A0, 0x00A1, 0x00A2, 0x00A3, 0x00A4, 0x00A5, 0x00A6, 0x00A7, 0x00A8, 0x00A9, 0x00AA, 0x00AB, 0x00AC, 0x00AD, 0x00AE, 0x00AF, 0x00B0, 0x00B1, 0x00B2, 0x00B3, 0x00B4, 0x00B5, 0x00B6, 0x00B7, 0x00B8, 0x00B9, 0x00BA, 0x00BB, 0x00BC, 0x00BD, 0x00BE, 0x00BF, 0x00C0, 0x00C1, 0x00C2, 0x00C3, 0x00C4, 0x00C5, 0x00C6, 0x00C7, 0x00C8, 0x00C9, 0x00CA, 0x00CB, 0x00CC, 0x00CD, 0x00CE, 0x00CF, 0x00D0, 0x00D1, 0x00D2, 0x00D3, 0x00D4, 0x00D5, 0x00D6, 0x00D7, 0x00D8, 0x00D9, 0x00DA, 0x00DB, 0x00DC, 0x00DD, 0x00DE, 0x00DF, 0x00E0, 0x00E1, 0x00E2, 0x00E3, 0x00E4, 0x00E5, 0x00E6, 0x00E7, 0x00E8, 0x00E9, 0x00EA, 0x00EB, 0x00EC, 0x00ED, 0x00EE, 0x00EF, 0x00F0, 0x00F1, 0x00F2, 0x00F3, 0x00F4, 0x00F5, 0x00F6, 0x00F7, 0x00F8, 0x00F9, 0x00FA, 0x00FB, 0x00FC, 0x00FD, 0x00FE, 0x00FF, 0x0100, 0x0101, 0x0102, 0x0103, 0x0104, 0x0105, 0x0106, 0x0107, 0x0108, 0x0109, 0x010A, 0x010B, 0x010C, 0x010D, 0x010E, 0x010F, 0x0110, 0x0111, 0x0112, 0x0113, 0x0114, 0x0115, 0x0116, 0x0117, 0x0118, 0x0119, 0x011A, 0x011B, 0x011C, 0x011D, 0x011E, 0x011F, 0x0120, 0x0121, 0x0122, 0x0123, 0x0124, 0x0125, 0x0126, 0x0127, 0x0128, 0x0129, 0x012A, 0x012B, 0x012C, 0x012D, 0x012E, 0x012F, 0x0130, 0x0131, 0x0132, 0x0133, 0x0134, 0x0135, 0x0136, 0x0137, 0x0138, 0x0139, 0x013A, 0x013B, 0x013C, 0x013D, 0x013E, 0x013F, 0x0140, 0x0141, 0x0142, 0x0143, 0x0144, 0x0145, 0x0146, 0x0147, 0x0148, 0x0149, 0x014A, 0x014B, 0x014C, 0x014D, 0x014E, 0x014F, 0x0150, 0x0151, 0x0152, 0x0153, 0x0154, 0x0155, 0x0156, 0x0157, 0x0158, 0x0159, 0x015A, 0x015B, 0x015C, 0x015D, 0x015E, 0x015F, 0x0160, 0x0161, 0x0162, 0x0163, 0x0164, 0x0165, 0x0166, 0x0167, 0x0168, 0x0169, 0x016A, 0x016B, 0x016C, 0x016D, 0x016E, 0x016F, 0x0170, 0x0171, 0x0172, 0x0173, 0x0174, 0x0175, 0x0176, 0x0177, 0x0178, 0x0179, 0x017A, 0x017B, 0x017C, 0x017D, 0x017E, 0x017F, 0x0180, 0x0181, 0x0182, 0x0183, 0x0184, 0x0185, 0x0186, 0x0187, 0x0188, 0x0189, 0x018A, 0x018B, 0x018C, 0x018D, 0x018E, 0x018F, 0x0190, 0x0191, 0x0192, 0x0193, 0x0194, 0x0195, 0x0196, 0x0197, 0x0198, 0x0199, 0x019A, 0x019B, 0x019C, 0x019D, 0x019E, 0x019F, 0x01A0, 0x01A1, 0x01A2, 0x01A3, 0x01A4, 0x01A5, 0x01A6, 0x01A7, 0x01A8, 0x01A9, 0x01AA, 0x01AB, 0x01AC, 0x01AD, 0x01AE, 0x01AF, 0x01B0, 0x01B1, 0x01B2, 0x01B3, 0x01B4, 0x01B5, 0x01B6, 0x01B7, 0x01B8, 0x01B9, 0x01BA, 0x01BB, 0x01BC, 0x01BD, 0x01BE, 0x01BF, 0x01C0, 0x01C1, 0x01C2, 0x01C3, 0x01C4, 0x01C5, 0x01C6, 0x01C7, 0x01C8, 0x01C9, 0x01CA, 0x01CB, 0x01CC, 0x01CD, 0x01CE, 0x01CF, 0x01D0, 0x01D1, 0x01D2, 0x01D3, 0x01D4, 0x01D5, 0x01D6, 0x01D7, 0x01D8, 0x01D9, 0x01DA, 0x01DB, 0x01DC, 0x01DD, 0x01DE, 0x01DF, 0x01E0, 0x01E1, 0x01E2, 0x01E3, 0x01E4, 0x01E5, 0x01E6, 0x01E7, 0x01E8, 0x01E9, 0x01EA, 0x01EB, 0x01EC, 0x01ED, 0x01EE, 0x01EF, 0x01F0, 0x01F1, 0x01F2, 0x01F3, 0x01F4, 0x01F5, 0x01F6, 0x01F7, 0x01F8, 0x01F9, 0x01FA, 0x01FB, 0x01FC, 0x01FD, 0x01FE, 0x01FF, 0x0200, 0x0201, 0x0202, 0x0203, 0x0204, 0x0205, 0x0206, 0x0207, 0x0208, 0x0209, 0x020A, 0x020B, 0x020C, 0x020D, 0x020E, 0x020F, 0x0210, 0x0211, 0x0212, 0x0213, 0x0214, 0x0215, 0x0216, 0x0217, 0x0218, 0x0219, 0x021A, 0x021B, 0x021C, 0x021D, 0x021E, 0x021F, 0x0220, 0x0221, 0x0222, 0x0223, 0x0224, 0x0225, 0x0226, 0x0227, 0x0228, 0x0229, 0x022A, 0x022B, 0x022C, 0x022D, 0x022E, 0x022F, 0x0230, 0x0231, 0x0232, 0x0233, 0x0234, 0x0235, 0x0236, 0x0237, 0x0238, 0x0239, 0x023A, 0x023B, 0x023C, 0x023D, 0x023E, 0x023F, 0x0240, 0x0241, 0x0242, 0x0243, 0x0244, 0x0245, 0x0246, 0x0247, 0x0248, 0x0249, 0x024A, 0x024B, 0x024C, 0x024D, 0x024E, 0x024F, 0x0250, 0x0251, 0x0252, 0x0253, 0x0254, 0x0255, 0x0256, 0x0257, 0x0258, 0x0259, 0x025A, 0x025B, 0x025C, 0x025D, 0x025E, 0x025F, 0x0260, 0x0261, 0x0262, 0x0263, 0x0264, 0x0265, 0x0266, 0x0267, 0x0268, 0x0269, 0x026A, 0x026B, 0x026C, 0x026D, 0x026E, 0x026F, 0x0270, 0x0271, 0x0272, 0x0273, 0x0274, 0x0275, 0x0276, 0x0277, 0x0278, 0x0279, 0x027A, 0x027B, 0x027C, 0x027D, 0x027E, 0x027F, 0x0280, 0x0281, 0x0282, 0x0283, 0x0284, 0x0285, 0x0286, 0x0287, 0x0288, 0x0289, 0x028A, 0x028B, 0x028C, 0x028D, 0x028E, 0x028F, 0x0290, 0x0291, 0x0292, 0x0293, 0x0294, 0x0295, 0x0296, 0x0297, 0x0298, 0x0299, 0x029A, 0x029B, 0x029C, 0x029D, 0x029E, 0x029F, 0x02A0, 0x02A1, 0x02A2, 0x02A3, 0x02A4, 0x02A5, 0x02A6, 0x02A7, 0x02A8, 0x02A9, 0x02AA, 0x02AB, 0x02AC, 0x02AD, 0x02AE, 0x02AF, 0x02B0, 0x02B1, 0x02B2, 0x02B3, 0x02B4, 0x02B5, 0x02B6, 0x02B7, 0x02B8, 0x02B9, 0x02BA, 0x02BB, 0x02BC, 0x02BD, 0x02BE, 0x02BF, 0x02C0, 0x02C1, 0x02C2, 0x02C3, 0x02C4, 0x02C5, 0x02C6, 0x02C7, 0x02C8, 0x02C9, 0x02CA, 0x02CB, 0x02CC, 0x02CD, 0x02CE, 0x02CF, 0x02D0, 0x02D1, 0x02D2, 0x02D3, 0x02D4, 0x02D5, 0x02D6, 0x02D7, 0x02D8, 0x02D9, 0x02DA, 0x02DB, 0x02DC, 0x02DD, 0x02DE, 0x02DF, 0x02E0, 0x02E1, 0x02E2, 0x02E3, 0x02E4, 0x02E5, 0x02E6, 0x02E7, 0x02E8, 0x02E9, 0x02EA, 0x02EB, 0x02EC, 0x02ED, 0x02EE, 0x02EF, 0x02F0, 0x02F1, 0x02F2, 0x02F3, 0x02F4, 0x02F5, 0x02F6, 0x02F7, 0x02F8, 0x02F9, 0x02FA, 0x02FB, 0x02FC, 0x02FD, 0x02FE, 0x02FF, 0x0300, 0x0301, 0x0302, 0x0303, 0x0304, 0x0305, 0x0306, 0x0307, 0x0308, 0x0309, 0x030A, 0x030B, 0x030C, 0x030D, 0x030E, 0x030F, 0x0310, 0x0311, 0x0312, 0x0313, 0x0314, 0x0315, 0x0316, 0x0317, 0x0318, 0x0319, 0x031A, 0x031B, 0x031C, 0x031D, 0x031E, 0x031F, 0x0320, 0x0321, 0x0322, 0x0323, 0x0324, 0x0325, 0x0326, 0x0327, 0x0328, 0x0329, 0x032A, 0x032B, 0x032C, 0x032D, 0x032E, 0x032F, 0x0330, 0x0331, 0x0332, 0x0333, 0x0334, 0x0335, 0x0336, 0x0337, 0x0338, 0x0339, 0x033A, 0x033B, 0x033C, 0x033D, 0x033E, 0x033F, 0x0340, 0x0341, 0x0342, 0x0343, 0x0344, 0x0345, 0x0346, 0x0347, 0x0348, 0x0349, 0x034A, 0x034B, 0x034C, 0x034D, 0x034E, 0x034F, 0x0350, 0x0351, 0x0352, 0x0353, 0x0354, 0x0355, 0x0356, 0x0357, 0x0358, 0x0359, 0x035A, 0x035B, 0x035C, 0x035D, 0x035E, 0x035F, 0x0360, 0x0361, 0x0362, 0x0363, 0x0364, 0x0365, 0x0366, 0x0367, 0x0368, 0x0369, 0x036A, 0x036B, 0x036C, 0x036D, 0x036E, 0x036F, 0x0370, 0x0371, 0x0372, 0x0373, 0x0374, 0x0375, 0x0376, 0x0377, 0x0378, 0x0379, 0x037A, 0x037B, 0x037C, 0x037D, 0x037E, 0x037F, 0x0380, 0x0381, 0x0382, 0x0383, 0x0384, 0x0385, 0x0386, 0x0387, 0x0388, 0x0389, 0x038A, 0x038B, 0x038C, 0x038D, 0x038E, 0x038F, 0x0390, 0x0391, 0x0392, 0x0393, 0x0394, 0x0395, 0x0396, 0x0397, 0x0398, 0x0399, 0x039A, 0x039B, 0x039C, 0x039D, 0x039E, 0x039F, 0x03A0, 0x03A1, 0x03A2, 0x03A3, 0x03A4, 0x03A5, 0x03A6, 0x03A7, 0x03A8, 0x03A9, 0x03AA, 0x03AB, 0x03AC, 0x03AD, 0x03AE, 0x03AF, 0x03B0, 0x03B1, 0x03B2, 0x03B3, 0x03B4, 0x03B5, 0x03B6, 0x03B7, 0x03B8, 0x03B9, 0x03BA, 0x03BB, 0x03BC, 0x03BD, 0x03BE, 0x03BF, 0x03C0, 0x03C1, 0x03C2, 0x03C3, 0x03C4, 0x03C5, 0x03C6, 0x03C7, 0x03C8, 0x03C9, 0x03CA, 0x03CB, 0x03CC, 0x03CD, 0x03CE, 0x03CF, 0x03D0, 0x03D1, 0x03D2, 0x03D3, 0x03D4, 0x03D5, 0x03D6, 0x03D7, 0x03D8, 0x03D9, 0x03DA, 0x03DB, 0x03DC, 0x03DD, 0x03DE, 0x03DF, 0x03E0, 0x03E1, 0x03E2, 0x03E3, 0x03E4, 0x03E5, 0x03E6, 0x03E7, 0x03E8, 0x03E9, 0x03EA, 0x03EB, 0x03EC, 0x03ED, 0x03EE, 0x03EF, 0x03F0, 0x03F1, 0x03F2, 0x03F3, 0x03F4, 0x03F5, 0x03F6, 0x03F7, 0x03F8, 0x03F9, 0x03FA, 0x03FB, 0x03FC, 0x03FD, 0x03FE, 0x03FF, 0x0000, 0x0001, 0x0002, 0x0003, 0x0004, 0x0005, 0x0006, 0x0007, 0x0008, 0x0009, 0x000A, 0x000B, 0x000C, 0x000D, 0x000E, 0x000F, 0x0010, 0x0011, 0x0012, 0x0013, 0x0014, 0x0015, 0x0016, 0x0017, 0x0018, 0x0019, 0x001A, 0x001B, 0x001C, 0x001D, 0x001E, 0x001F, 0x0020, 0x0021, 0x0022, 0x0023, 0x0024, 0x0025, 0x0026, 0x0027, 0x0028, 0x0029, 0x002A, 0x002B, 0x002C, 0x002D, 0x002E, 0x002F, 0x0030, 0x0031, 0x0032, 0x0033, 0x0034, 0x0035, 0x0036, 0x0037, 0x0038, 0x0039, 0x003A, 0x003B, 0x003C, 0x003D, 0x003E, 0x003F, 0x0040, 0x0041, 0x0042, 0x0043, 0x0044, 0x0045, 0x0046, 0x0047, 0x0048, 0x0049, 0x004A, 0x004B, 0x004C, 0x004D, 0x004E, 0x004F, 0x0050, 0x0051, 0x0052, 0x0053, 0x0054, 0x0055, 0x0056, 0x0057, 0x0058, 0x0059, 0x005A, 0x005B, 0x005C, 0x005D, 0x005E, 0x005F, 0x0060, 0x0061, 0x0062, 0x0063, 0x0064, 0x0065, 0x0066, 0x0067, 0x0068, 0x0069, 0x006A, 0x006B, 0x006C, 0x006D, 0x006E, 0x006F, 0x0070, 0x0071, 0x0072, 0x0073, 0x0074, 0x0075, 0x0076, 0x0077, 0x0078, 0x0079, 0x007A, 0x007B, 0x007C, 0x007D, 0x007E, 0x007F, 0x0080, 0x0081, 0x0082, 0x0083, 0x0084, 0x0085, 0x0086, 0x0087, 0x0088, 0x0089, 0x008A, 0x008B, 0x008C, 0x008D, 0x008E, 0x008F, 0x0090, 0x0091, 0x0092, 0x0093, 0x0094, 0x0095, 0x0096, 0x0097, 0x0098, 0x0099, 0x009A, 0x009B, 0x009C, 0x009D, 0x009E, 0x009F, 0x00A0, 0x00A1, 0x00A2, 0x00A3, 0x00A4, 0x00A5, 0x00A6, 0x00A7, 0x00A8, 0x00A9, 0x00AA, 0x00AB, 0x00AC, 0x00AD, 0x00AE, 0x00AF, 0x00B0, 0x00B1, 0x00B2, 0x00B3, 0x00B4, 0x00B5, 0x00B6, 0x00B7, 0x00B8, 0x00B9, 0x00BA, 0x00BB, 0x00BC, 0x00BD, 0x00BE, 0x00BF, 0x00C0, 0x00C1, 0x00C2, 0x00C3, 0x00C4, 0x00C5, 0x00C6, 0x00C7, 0x00C8, 0x00C9, 0x00CA, 0x00CB, 0x00CC, 0x00CD, 0x00CE, 0x00CF, 0x00D0, 0x00D1, 0x00D2, 0x00D3, 0x00D4, 0x00D5, 0x00D6, 0x00D7, 0x00D8, 0x00D9, 0x00DA, 0x00DB, 0x00DC, 0x00DD, 0x00DE, 0x00DF, 0x00E0, 0x00E1, 0x00E2, 0x00E3, 0x00E4, 0x00E5, 0x00E6, 0x00E7, 0x00E8, 0x00E9, 0x00EA, 0x00EB, 0x00EC, 0x00ED, 0x00EE, 0x00EF, 0x00F0, 0x00F1, 0x00F2, 0x00F3, 0x00F4, 0x00F5, 0x00F6, 0x00F7, 0x00F8, 0x00F9, 0x00FA, 0x00FB, 0x00FC, 0x00FD, 0x00FE, 0x00FF, 0x0100, 0x0101, 0x0102, 0x0103, 0x0104, 0x0105, 0x0106, 0x0107, 0x0108, 0x0109, 0x010A, 0x010B, 0x010C, 0x010D, 0x010E, 0x010F, 0x0110, 0x0111, 0x0112, 0x0113, 0x0114, 0x0115, 0x0116, 0x0117, 0x0118, 0x0119, 0x011A, 0x011B, 0x011C, 0x011D, 0x011E, 0x011F, 0x0120, 0x0121, 0x0122, 0x0123, 0x0124, 0x0125, 0x0126, 0x0127, 0x0128, 0x0129, 0x012A, 0x012B, 0x012C, 0x012D, 0x012E, 0x012F, 0x0130, 0x0131, 0x0132, 0x0133, 0x0134, 0x0135, 0x0136, 0x0137, 0x0138, 0x0139, 0x013A, 0x013B, 0x013C, 0x013D, 0x013E, 0x013F, 0x0140, 0x0141, 0x0142, 0x0143, 0x0144, 0x0145, 0x0146, 0x0147, 0x0148, 0x0149, 0x014A, 0x014B, 0x014C, 0x014D, 0x014E, 0x014F, 0x0150, 0x0151, 0x0152, 0x0153, 0x0154, 0x0155, 0x0156, 0x0157, 0x0158, 0x0159, 0x015A, 0x015B, 0x015C, 0x015D, 0x015E, 0x015F, 0x0160, 0x0161, 0x0162, 0x0163, 0x0164, 0x0165, 0x0166, 0x0167, 0x0168, 0x0169, 0x016A, 0x016B, 0x016C, 0x016D, 0x016E, 0x016F, 0x0170, 0x0171, 0x0172, 0x0173, 0x0174, 0x0175, 0x0176, 0x0177, 0x0178, 0x0179, 0x017A, 0x017B, 0x017C, 0x017D, 0x017E, 0x017F, 0x0180, 0x0181, 0x0182, 0x0183, 0x0184, 0x0185, 0x0186, 0x0187, 0x0188, 0x0189, 0x018A, 0x018B, 0x018C, 0x018D, 0x018E, 0x018F, 0x0190, 0x0191, 0x0192, 0x0193, 0x0194, 0x0195, 0x0196, 0x0197, 0x0198, 0x0199, 0x019A, 0x019B, 0x019C, 0x019D, 0x019E, 0x019F, 0x01A0, 0x01A1, 0x01A2, 0x01A3, 0x01A4, 0x01A5, 0x01A6, 0x01A7, 0x01A8, 0x01A9, 0x01AA, 0x01AB, 0x01AC, 0x01AD, 0x01AE, 0x01AF, 0x01B0, 0x01B1, 0x01B2, 0x01B3, 0x01B4, 0x01B5, 0x01B6, 0x01B7, 0x01B8, 0x01B9, 0x01BA, 0x01BB, 0x01BC, 0x01BD, 0x01BE, 0x01BF, 0x01C0, 0x01C1, 0x01C2, 0x01C3, 0x01C4, 0x01C5, 0x01C6, 0x01C7, 0x01C8, 0x01C9, 0x01CA, 0x01CB, 0x01CC, 0x01CD, 0x01CE, 0x01CF, 0x01D0, 0x01D1, 0x01D2, 0x01D3, 0x01D4, 0x01D5, 0x01D6, 0x01D7, 0x01D8, 0x01D9, 0x01DA, 0x01DB, 0x01DC, 0x01DD, 0x01DE, 0x01DF, 0x01E0, 0x01E1, 0x01E2, 0x01E3, 0x01E4, 0x01E5, 0x01E6, 0x01E7, 0x01E8, 0x01E9, 0x01EA, 0x01EB, 0x01EC, 0x01ED, 0x01EE, 0x01EF, 0x01F0, 0x01F1, 0x01F2, 0x01F3, 0x01F4, 0x01F5, 0x01F6, 0x01F7, 0x01F8, 0x01F9, 0x01FA, 0x01FB, 0x01FC, 0x01FD, 0x01FE, 0x01FF, 0x0200, 0x0201, 0x0202, 0x0203, 0x0204, 0x0205, 0x0206, 0x0207, 0x0208, 0x0209, 0x020A, 0x020B, 0x020C, 0x020D, 0x020E, 0x020F, 0x0210, 0x0211, 0x0212, 0x0213, 0x0214, 0x0215, 0x0216, 0x0217, 0x0218, 0x0219, 0x021A, 0x021B, 0x021C, 0x021D, 0x021E, 0x021F, 0x0220, 0x0221, 0x0222, 0x0223, 0x0224, 0x0225, 0x0226, 0x0227, 0x0228, 0x0229, 0x022A, 0x022B, 0x022C, 0x022D, 0x022E, 0x022F, 0x0230, 0x0231, 0x0232, 0x0233, 0x0234, 0x0235, 0x0236, 0x0237, 0x0238, 0x0239, 0x023A, 0x023B, 0x023C, 0x023D, 0x023E, 0x023F, 0x0240, 0x0241, 0x0242, 0x0243, 0x0244, 0x0245, 0x0246, 0x0247, 0x0248, 0x0249, 0x024A, 0x024B, 0x024C, 0x024D, 0x024E, 0x024F, 0x0250, 0x0251, 0x0252, 0x0253, 0x0254, 0x0255, 0x0256, 0x0257, 0x0258, 0x0259, 0x025A, 0x025B, 0x025C, 0x025D, 0x025E, 0x025F, 0x0260, 0x0261, 0x0262, 0x0263, 0x0264, 0x0265, 0x0266, 0x0267, 0x0268, 0x0269, 0x026A, 0x026B, 0x026C, 0x026D, 0x026E, 0x026F, 0x0270, 0x0271, 0x0272, 0x0273, 0x0274, 0x0275, 0x0276, 0x0277, 0x0278, 0x0279, 0x027A, 0x027B, 0x027C, 0x027D, 0x027E, 0x027F, 0x0280, 0x0281, 0x0282, 0x0283, 0x0284, 0x0285, 0x0286, 0x0287, 0x0288, 0x0289, 0x028A, 0x028B, 0x028C, 0x028D, 0x028E, 0x028F, 0x0290, 0x0291, 0x0292, 0x0293, 0x0294, 0x0295, 0x0296, 0x0297, 0x0298, 0x0299, 0x029A, 0x029B, 0x029C, 0x029D, 0x029E, 0x029F, 0x02A0, 0x02A1, 0x02A2, 0x02A3, 0x02A4, 0x02A5, 0x02A6, 0x02A7, 0x02A8, 0x02A9, 0x02AA, 0x02AB, 0x02AC, 0x02AD, 0x02AE, 0x02AF, 0x02B0, 0x02B1, 0x02B2, 0x02B3, 0x02B4, 0x02B5, 0x02B6, 0x02B7, 0x02B8, 0x02B9, 0x02BA, 0x02BB, 0x02BC, 0x02BD, 0x02BE, 0x02BF, 0x02C0, 0x02C1, 0x02C2, 0x02C3, 0x02C4, 0x02C5, 0x02C6, 0x02C7, 0x02C8, 0x02C9, 0x02CA, 0x02CB, 0x02CC, 0x02CD, 0x02CE, 0x02CF, 0x02D0, 0x02D1, 0x02D2, 0x02D3, 0x02D4, 0x02D5, 0x02D6, 0x02D7, 0x02D8, 0x02D9, 0x02DA, 0x02DB, 0x02DC, 0x02DD, 0x02DE, 0x02DF, 0x02E0, 0x02E1, 0x02E2, 0x02E3, 0x02E4, 0x02E5, 0x02E6, 0x02E7, 0x02E8, 0x02E9, 0x02EA, 0x02EB, 0x02EC, 0x02ED, 0x02EE, 0x02EF, 0x02F0, 0x02F1, 0x02F2, 0x02F3, 0x02F4, 0x02F5, 0x02F6, 0x02F7, 0x02F8, 0x02F9, 0x02FA, 0x02FB, 0x02FC, 0x02FD, 0x02FE, 0x02FF, 0x0300, 0x0301, 0x0302, 0x0303, 0x0304, 0x0305, 0x0306, 0x0307, 0x0308, 0x0309, 0x030A, 0x030B, 0x030C, 0x030D, 0x030E, 0x030F, 0x0310, 0x0311, 0x0312, 0x0313, 0x0314, 0x0315, 0x0316, 0x0317, 0x0318, 0x0319, 0x031A, 0x031B, 0x031C, 0x031D, 0x031E, 0x031F, 0x0320, 0x0321, 0x0322, 0x0323, 0x0324, 0x0325, 0x0326, 0x0327, 0x0328, 0x0329, 0x032A, 0x032B, 0x032C, 0x032D, 0x032E, 0x032F, 0x0330, 0x0331, 0x0332, 0x0333, 0x0334, 0x0335, 0x0336, 0x0337, 0x0338, 0x0339, 0x033A, 0x033B, 0x033C, 0x033D, 0x033E, 0x033F, 0x0340, 0x0341, 0x0342, 0x0343, 0x0344, 0x0345, 0x0346, 0x0347, 0x0348, 0x0349, 0x034A, 0x034B, 0x034C, 0x034D, 0x034E, 0x034F, 0x0350, 0x0351, 0x0352, 0x0353, 0x0354, 0x0355, 0x0356, 0x0357, 0x0358, 0x0359, 0x035A, 0x035B, 0x035C, 0x035D, 0x035E, 0x035F, 0x0360, 0x0361, 0x0362, 0x0363, 0x0364, 0x0365, 0x0366, 0x0367, 0x0368, 0x0369, 0x036A, 0x036B, 0x036C, 0x036D, 0x036E, 0x036F, 0x0370, 0x0371, 0x0372, 0x0373, 0x0374, 0x0375, 0x0376, 0x0377, 0x0378, 0x0379, 0x037A, 0x037B, 0x037C, 0x037D, 0x037E, 0x037F, 0x0380, 0x0381, 0x0382, 0x0383, 0x0384, 0x0385, 0x0386, 0x0387, 0x0388, 0x0389, 0x038A, 0x038B, 0x038C, 0x038D, 0x038E, 0x038F, 0x0390, 0x0391, 0x0392, 0x0393, 0x0394, 0x0395, 0x0396, 0x0397, 0x0398, 0x0399, 0x039A, 0x039B, 0x039C, 0x039D, 0x039E, 0x039F, 0x03A0, 0x03A1, 0x03A2, 0x03A3, 0x03A4, 0x03A5, 0x03A6, 0x03A7, 0x03A8, 0x03A9, 0x03AA, 0x03AB, 0x03AC, 0x03AD, 0x03AE, 0x03AF, 0x03B0, 0x03B1, 0x03B2, 0x03B3, 0x03B4, 0x03B5, 0x03B6, 0x03B7, 0x03B8, 0x03B9, 0x03BA, 0x03BB, 0x03BC, 0x03BD, 0x03BE, 0x03BF, 0x03C0, 0x03C1, 0x03C2, 0x03C3, 0x03C4, 0x03C5, 0x03C6, 0x03C7, 0x03C8, 0x03C9, 0x03CA, 0x03CB, 0x03CC, 0x03CD, 0x03CE, 0x03CF, 0x03D0, 0x03D1, 0x03D2, 0x03D3, 0x03D4, 0x03D5, 0x03D6, 0x03D7, 0x03D8, 0x03D9, 0x03DA, 0x03DB, 0x03DC, 0x03DD, 0x03DE, 0x03DF, 0x03E0, 0x03E1, 0x03E2, 0x03E3, 0x03E4, 0x03E5, 0x03E6, 0x03E7, 0x03E8, 0x03E9, 0x03EA, 0x03EB, 0x03EC, 0x03ED, 0x03EE, 0x03EF, 0x03F0, 0x03F1, 0x03F2, 0x03F3, 0x03F4, 0x03F5, 0x03F6, 0x03F7, 0x03F8, 0x03F9, 0x03FA, 0x03FB, 0x03FC, 0x03FD, 0x03FE, 0x03FF

},

{

0x0000, 0xFFFF, 0x0001, 0xFFFF, 0x0002, 0xFFFF, 0x0003, 0xFFFF, 0x0004, 0xFFFF, 0x0005, 0xFFFF, 0x0006, 0xFFFF, 0x0007, 0xFFFF, 0x0008, 0xFFFF, 0x0009, 0xFFFF, 0x000A, 0xFFFF, 0x000B, 0xFFFF, 0x000C, 0xFFFF, 0x000D, 0xFFFF, 0x000E, 0xFFFF, 0x000F, 0xFFFF, 0x0010, 0xFFFF, 0x0011, 0xFFFF, 0x0012, 0xFFFF, 0x0013, 0xFFFF, 0x0014, 0xFFFF, 0x0015, 0xFFFF, 0x0016, 0xFFFF, 0x0017, 0xFFFF, 0x0018, 0xFFFF, 0x0019, 0xFFFF, 0x001A, 0xFFFF, 0x001B, 0xFFFF, 0x001C, 0xFFFF, 0x001D, 0xFFFF, 0x001E, 0xFFFF, 0x001F, 0xFFFF, 0x0020, 0xFFFF, 0x0021, 0xFFFF, 0x0022, 0xFFFF, 0x0023, 0xFFFF, 0x0024, 0xFFFF, 0x0025, 0xFFFF, 0x0026, 0xFFFF, 0x0027, 0xFFFF, 0x0028, 0xFFFF, 0x0029, 0xFFFF, 0x002A, 0xFFFF, 0x002B, 0xFFFF, 0x002C, 0xFFFF, 0x002D, 0xFFFF, 0x002E, 0xFFFF, 0x002F, 0xFFFF, 0x0030, 0xFFFF, 0x0031, 0xFFFF, 0x0032, 0xFFFF, 0x0033, 0xFFFF, 0x0034, 0xFFFF, 0x0035, 0xFFFF, 0x0036, 0xFFFF, 0x0037, 0xFFFF, 0x0038, 0xFFFF, 0x0039, 0xFFFF, 0x003A, 0xFFFF, 0x003B, 0xFFFF, 0x003C, 0xFFFF, 0x003D, 0xFFFF, 0x003E, 0xFFFF, 0x003F, 0xFFFF, 0x0040, 0xFFFF, 0x0041, 0xFFFF, 0x0042, 0xFFFF, 0x0043, 0xFFFF, 0x0044, 0xFFFF, 0x0045, 0xFFFF, 0x0046, 0xFFFF, 0x0047, 0xFFFF, 0x0048, 0xFFFF, 0x0049, 0xFFFF, 0x004A, 0xFFFF, 0x004B, 0xFFFF, 0x004C, 0xFFFF, 0x004D, 0xFFFF, 0x004E, 0xFFFF, 0x004F, 0xFFFF, 0x0050, 0xFFFF, 0x0051, 0xFFFF, 0x0052, 0xFFFF, 0x0053, 0xFFFF, 0x0054, 0xFFFF, 0x0055, 0xFFFF, 0x0056, 0xFFFF, 0x0057, 0xFFFF, 0x0058, 0xFFFF, 0x0059, 0xFFFF, 0x005A, 0xFFFF, 0x005B, 0xFFFF, 0x005C, 0xFFFF, 0x005D, 0xFFFF, 0x005E, 0xFFFF, 0x005F, 0xFFFF, 0x0060, 0xFFFF, 0x0061, 0xFFFF, 0x0062, 0xFFFF, 0x0063, 0xFFFF, 0x0064, 0xFFFF, 0x0065, 0xFFFF, 0x0066, 0xFFFF, 0x0067, 0xFFFF, 0x0068, 0xFFFF, 0x0069, 0xFFFF, 0x006A, 0xFFFF, 0x006B, 0xFFFF, 0x006C, 0xFFFF, 0x006D, 0xFFFF, 0x006E, 0xFFFF, 0x006F, 0xFFFF, 0x0070, 0xFFFF, 0x0071, 0xFFFF, 0x0072, 0xFFFF, 0x0073, 0xFFFF, 0x0074, 0xFFFF, 0x0075, 0xFFFF, 0x0076, 0xFFFF, 0x0077, 0xFFFF, 0x0078, 0xFFFF, 0x0079, 0xFFFF, 0x007A, 0xFFFF, 0x007B, 0xFFFF, 0x007C, 0xFFFF, 0x007D, 0xFFFF, 0x007E, 0xFFFF, 0x007F, 0xFFFF, 0x0080, 0xFFFF, 0x0081, 0xFFFF, 0x0082, 0xFFFF, 0x0083, 0xFFFF, 0x0084, 0xFFFF, 0x0085, 0xFFFF, 0x0086, 0xFFFF, 0x0087, 0xFFFF, 0x0088, 0xFFFF, 0x0089, 0xFFFF, 0x008A, 0xFFFF, 0x008B, 0xFFFF, 0x008C, 0xFFFF, 0x008D, 0xFFFF, 0x008E, 0xFFFF, 0x008F, 0xFFFF, 0x0090, 0xFFFF, 0x0091, 0xFFFF, 0x0092, 0xFFFF, 0x0093, 0xFFFF, 0x0094, 0xFFFF, 0x0095, 0xFFFF, 0x0096, 0xFFFF, 0x0097, 0xFFFF, 0x0098, 0xFFFF, 0x0099, 0xFFFF, 0x009A, 0xFFFF, 0x009B, 0xFFFF, 0x009C, 0xFFFF, 0x009D, 0xFFFF, 0x009E, 0xFFFF, 0x009F, 0xFFFF, 0x00A0, 0xFFFF, 0x00A1, 0xFFFF, 0x00A2, 0xFFFF, 0x00A3, 0xFFFF, 0x00A4, 0xFFFF, 0x00A5, 0xFFFF, 0x00A6, 0xFFFF, 0x00A7, 0xFFFF, 0x00A8, 0xFFFF, 0x00A9, 0xFFFF, 0x00AA, 0xFFFF, 0x00AB, 0xFFFF, 0x00AC, 0xFFFF, 0x00AD, 0xFFFF, 0x00AE, 0xFFFF, 0x00AF, 0xFFFF, 0x00B0, 0xFFFF, 0x00B1, 0xFFFF, 0x00B2, 0xFFFF, 0x00B3, 0xFFFF, 0x00B4, 0xFFFF, 0x00B5, 0xFFFF, 0x00B6, 0xFFFF, 0x00B7, 0xFFFF, 0x00B8, 0xFFFF, 0x00B9, 0xFFFF, 0x00BA, 0xFFFF, 0x00BB, 0xFFFF, 0x00BC, 0xFFFF, 0x00BD, 0xFFFF, 0x00BE, 0xFFFF, 0x00BF, 0xFFFF, 0x00C0, 0xFFFF, 0x00C1, 0xFFFF, 0x00C2, 0xFFFF, 0x00C3, 0xFFFF, 0x00C4, 0xFFFF, 0x00C5, 0xFFFF, 0x00C6, 0xFFFF, 0x00C7, 0xFFFF, 0x00C8, 0xFFFF, 0x00C9, 0xFFFF, 0x00CA, 0xFFFF, 0x00CB, 0xFFFF, 0x00CC, 0xFFFF, 0x00CD, 0xFFFF, 0x00CE, 0xFFFF, 0x00CF, 0xFFFF, 0x00D0, 0xFFFF, 0x00D1, 0xFFFF, 0x00D2, 0xFFFF, 0x00D3, 0xFFFF, 0x00D4, 0xFFFF, 0x00D5, 0xFFFF, 0x00D6, 0xFFFF, 0x00D7, 0xFFFF, 0x00D8, 0xFFFF, 0x00D9, 0xFFFF, 0x00DA, 0xFFFF, 0x00DB, 0xFFFF, 0x00DC, 0xFFFF, 0x00DD, 0xFFFF, 0x00DE, 0xFFFF, 0x00DF, 0xFFFF, 0x00E0, 0xFFFF, 0x00E1, 0xFFFF, 0x00E2, 0xFFFF, 0x00E3, 0xFFFF, 0x00E4, 0xFFFF, 0x00E5, 0xFFFF, 0x00E6, 0xFFFF, 0x00E7, 0xFFFF, 0x00E8, 0xFFFF, 0x00E9, 0xFFFF, 0x00EA, 0xFFFF, 0x00EB, 0xFFFF, 0x00EC, 0xFFFF, 0x00ED, 0xFFFF, 0x00EE, 0xFFFF, 0x00EF, 0xFFFF, 0x00F0, 0xFFFF, 0x00F1, 0xFFFF, 0x00F2, 0xFFFF, 0x00F3, 0xFFFF, 0x00F4, 0xFFFF, 0x00F5, 0xFFFF, 0x00F6, 0xFFFF, 0x00F7, 0xFFFF, 0x00F8, 0xFFFF, 0x00F9, 0xFFFF, 0x00FA, 0xFFFF, 0x00FB, 0xFFFF, 0x00FC, 0xFFFF, 0x00FD, 0xFFFF, 0x00FE, 0xFFFF, 0x00FF, 0xFFFF, 0x0100, 0xFFFF, 0x0101, 0xFFFF, 0x0102, 0xFFFF, 0x0103, 0xFFFF, 0x0104, 0xFFFF, 0x0105, 0xFFFF, 0x0106, 0xFFFF, 0x0107, 0xFFFF, 0x0108, 0xFFFF, 0x0109, 0xFFFF, 0x010A, 0xFFFF, 0x010B, 0xFFFF, 0x010C, 0xFFFF, 0x010D, 0xFFFF, 0x010E, 0xFFFF, 0x010F, 0xFFFF, 0x0110, 0xFFFF, 0x0111, 0xFFFF, 0x0112, 0xFFFF, 0x0113, 0xFFFF, 0x0114, 0xFFFF, 0x0115, 0xFFFF, 0x0116, 0xFFFF, 0x0117, 0xFFFF, 0x0118, 0xFFFF, 0x0119, 0xFFFF, 0x011A, 0xFFFF, 0x011B, 0xFFFF, 0x011C, 0xFFFF, 0x011D, 0xFFFF, 0x011E, 0xFFFF, 0x011F, 0xFFFF, 0x0120, 0xFFFF, 0x0121, 0xFFFF, 0x0122, 0xFFFF, 0x0123, 0xFFFF, 0x0124, 0xFFFF, 0x0125, 0xFFFF, 0x0126, 0xFFFF, 0x0127, 0xFFFF, 0x0128, 0xFFFF, 0x0129, 0xFFFF, 0x012A, 0xFFFF, 0x012B, 0xFFFF, 0x012C, 0xFFFF, 0x012D, 0xFFFF, 0x012E, 0xFFFF, 0x012F, 0xFFFF, 0x0130, 0xFFFF, 0x0131, 0xFFFF, 0x0132, 0xFFFF, 0x0133, 0xFFFF, 0x0134, 0xFFFF, 0x0135, 0xFFFF, 0x0136, 0xFFFF, 0x0137, 0xFFFF, 0x0138, 0xFFFF, 0x0139, 0xFFFF, 0x013A, 0xFFFF, 0x013B, 0xFFFF, 0x013C, 0xFFFF, 0x013D, 0xFFFF, 0x013E, 0xFFFF, 0x013F, 0xFFFF, 0x0140, 0xFFFF, 0x0141, 0xFFFF, 0x0142, 0xFFFF, 0x0143, 0xFFFF, 0x0144, 0xFFFF, 0x0145, 0xFFFF, 0x0146, 0xFFFF, 0x0147, 0xFFFF, 0x0148, 0xFFFF, 0x0149, 0xFFFF, 0x014A, 0xFFFF, 0x014B, 0xFFFF, 0x014C, 0xFFFF, 0x014D, 0xFFFF, 0x014E, 0xFFFF, 0x014F, 0xFFFF, 0x0150, 0xFFFF, 0x0151, 0xFFFF, 0x0152, 0xFFFF, 0x0153, 0xFFFF, 0x0154, 0xFFFF, 0x0155, 0xFFFF, 0x0156, 0xFFFF, 0x0157, 0xFFFF, 0x0158, 0xFFFF, 0x0159, 0xFFFF, 0x015A, 0xFFFF, 0x015B, 0xFFFF, 0x015C, 0xFFFF, 0x015D, 0xFFFF, 0x015E, 0xFFFF, 0x015F, 0xFFFF, 0x0160, 0xFFFF, 0x0161, 0xFFFF, 0x0162, 0xFFFF, 0x0163, 0xFFFF, 0x0164, 0xFFFF, 0x0165, 0xFFFF, 0x0166, 0xFFFF, 0x0167, 0xFFFF, 0x0168, 0xFFFF, 0x0169, 0xFFFF, 0x016A, 0xFFFF, 0x016B, 0xFFFF, 0x016C, 0xFFFF, 0x016D, 0xFFFF, 0x016E, 0xFFFF, 0x016F, 0xFFFF, 0x0170, 0xFFFF, 0x0171, 0xFFFF, 0x0172, 0xFFFF, 0x0173, 0xFFFF, 0x0174, 0xFFFF, 0x0175, 0xFFFF, 0x0176, 0xFFFF, 0x0177, 0xFFFF, 0x0178, 0xFFFF, 0x0179, 0xFFFF, 0x017A, 0xFFFF, 0x017B, 0xFFFF, 0x017C, 0xFFFF, 0x017D, 0xFFFF, 0x017E, 0xFFFF, 0x017F, 0xFFFF, 0x0180, 0xFFFF, 0x0181, 0xFFFF, 0x0182, 0xFFFF, 0x0183, 0xFFFF, 0x0184, 0xFFFF, 0x0185, 0xFFFF, 0x0186, 0xFFFF, 0x0187, 0xFFFF, 0x0188, 0xFFFF, 0x0189, 0xFFFF, 0x018A, 0xFFFF, 0x018B, 0xFFFF, 0x018C, 0xFFFF, 0x018D, 0xFFFF, 0x018E, 0xFFFF, 0x018F, 0xFFFF, 0x0190, 0xFFFF, 0x0191, 0xFFFF, 0x0192, 0xFFFF, 0x0193, 0xFFFF, 0x0194, 0xFFFF, 0x0195, 0xFFFF, 0x0196, 0xFFFF, 0x0197, 0xFFFF, 0x0198, 0xFFFF, 0x0199, 0xFFFF, 0x019A, 0xFFFF, 0x019B, 0xFFFF, 0x019C, 0xFFFF, 0x019D, 0xFFFF, 0x019E, 0xFFFF, 0x019F, 0xFFFF, 0x01A0, 0xFFFF, 0x01A1, 0xFFFF, 0x01A2, 0xFFFF, 0x01A3, 0xFFFF, 0x01A4, 0xFFFF, 0x01A5, 0xFFFF, 0x01A6, 0xFFFF, 0x01A7, 0xFFFF, 0x01A8, 0xFFFF, 0x01A9, 0xFFFF, 0x01AA, 0xFFFF, 0x01AB, 0xFFFF, 0x01AC, 0xFFFF, 0x01AD, 0xFFFF, 0x01AE, 0xFFFF, 0x01AF, 0xFFFF, 0x01B0, 0xFFFF, 0x01B1, 0xFFFF, 0x01B2, 0xFFFF, 0x01B3, 0xFFFF, 0x01B4, 0xFFFF, 0x01B5, 0xFFFF, 0x01B6, 0xFFFF, 0x01B7, 0xFFFF, 0x01B8, 0xFFFF, 0x01B9, 0xFFFF, 0x01BA, 0xFFFF, 0x01BB, 0xFFFF, 0x01BC, 0xFFFF, 0x01BD, 0xFFFF, 0x01BE, 0xFFFF, 0x01BF, 0xFFFF, 0x01C0, 0xFFFF, 0x01C1, 0xFFFF, 0x01C2, 0xFFFF, 0x01C3, 0xFFFF, 0x01C4, 0xFFFF, 0x01C5, 0xFFFF, 0x01C6, 0xFFFF, 0x01C7, 0xFFFF, 0x01C8, 0xFFFF, 0x01C9, 0xFFFF, 0x01CA, 0xFFFF, 0x01CB, 0xFFFF, 0x01CC, 0xFFFF, 0x01CD, 0xFFFF, 0x01CE, 0xFFFF, 0x01CF, 0xFFFF, 0x01D0, 0xFFFF, 0x01D1, 0xFFFF, 0x01D2, 0xFFFF, 0x01D3, 0xFFFF, 0x01D4, 0xFFFF, 0x01D5, 0xFFFF, 0x01D6, 0xFFFF, 0x01D7, 0xFFFF, 0x01D8, 0xFFFF, 0x01D9, 0xFFFF, 0x01DA, 0xFFFF, 0x01DB, 0xFFFF, 0x01DC, 0xFFFF, 0x01DD, 0xFFFF, 0x01DE, 0xFFFF, 0x01DF, 0xFFFF, 0x01E0, 0xFFFF, 0x01E1, 0xFFFF, 0x01E2, 0xFFFF, 0x01E3, 0xFFFF, 0x01E4, 0xFFFF, 0x01E5, 0xFFFF, 0x01E6, 0xFFFF, 0x01E7, 0xFFFF, 0x01E8, 0xFFFF, 0x01E9, 0xFFFF, 0x01EA, 0xFFFF, 0x01EB, 0xFFFF, 0x01EC, 0xFFFF, 0x01ED, 0xFFFF, 0x01EE, 0xFFFF, 0x01EF, 0xFFFF, 0x01F0, 0xFFFF, 0x01F1, 0xFFFF, 0x01F2, 0xFFFF, 0x01F3, 0xFFFF, 0x01F4, 0xFFFF, 0x01F5, 0xFFFF, 0x01F6, 0xFFFF, 0x01F7, 0xFFFF, 0x01F8, 0xFFFF, 0x01F9, 0xFFFF, 0x01FA, 0xFFFF, 0x01FB, 0xFFFF, 0x01FC, 0xFFFF, 0x01FD, 0xFFFF, 0x01FE, 0xFFFF, 0x01FF, 0xFFFF, 0x0000, 0x0000, 0x0001, 0x0001, 0x0002, 0x0002, 0x0003, 0x0003, 0x0004, 0x0004, 0x0005, 0x0005, 0x0006, 0x0006, 0x0007, 0x0007, 0x0008, 0x0008, 0x0009, 0x0009, 0x000A, 0x000A, 0x000B, 0x000B, 0x000C, 0x000C, 0x000D, 0x000D, 0x000E, 0x000E, 0x000F, 0x000F, 0x0010, 0x0010, 0x0011, 0x0011, 0x0012, 0x0012, 0x0013, 0x0013, 0x0014, 0x0014, 0x0015, 0x0015, 0x0016, 0x0016, 0x0017, 0x0017, 0x0018, 0x0018, 0x0019, 0x0019, 0x001A, 0x001A, 0x001B, 0x001B, 0x001C, 0x001C, 0x001D, 0x001D, 0x001E, 0x001E, 0x001F, 0x001F, 0x0020, 0x0020, 0x0021, 0x0021, 0x0022, 0x0022, 0x0023, 0x0023, 0x0024, 0x0024, 0x0025, 0x0025, 0x0026, 0x0026, 0x0027, 0x0027, 0x0028, 0x0028, 0x0029, 0x0029, 0x002A, 0x002A, 0x002B, 0x002B, 0x002C, 0x002C, 0x002D, 0x002D, 0x002E, 0x002E, 0x002F, 0x002F, 0x0030, 0x0030, 0x0031, 0x0031, 0x0032, 0x0032, 0x0033, 0x0033, 0x0034, 0x0034, 0x0035, 0x0035, 0x0036, 0x0036, 0x0037, 0x0037, 0x0038, 0x0038, 0x0039, 0x0039, 0x003A, 0x003A, 0x003B, 0x003B, 0x003C, 0x003C, 0x003D, 0x003D, 0x003E, 0x003E, 0x003F, 0x003F, 0x0040, 0x0040, 0x0041, 0x0041, 0x0042, 0x0042, 0x0043, 0x0043, 0x0044, 0x0044, 0x0045, 0x0045, 0x0046, 0x0046, 0x0047, 0x0047, 0x0048, 0x0048, 0x0049, 0x0049, 0x004A, 0x004A, 0x004B, 0x004B, 0x004C, 0x004C, 0x004D, 0x004D, 0x004E, 0x004E, 0x004F, 0x004F, 0x0050, 0x0050, 0x0051, 0x0051, 0x0052, 0x0052, 0x0053, 0x0053, 0x0054, 0x0054, 0x0055, 0x0055, 0x0056, 0x0056, 0x0057, 0x0057, 0x0058, 0x0058, 0x0059, 0x0059, 0x005A, 0x005A, 0x005B, 0x005B, 0x005C, 0x005C, 0x005D, 0x005D, 0x005E, 0x005E, 0x005F, 0x005F, 0x0060, 0x0060, 0x0061, 0x0061, 0x0062, 0x0062, 0x0063, 0x0063, 0x0064, 0x0064, 0x0065, 0x0065, 0x0066, 0x0066, 0x0067, 0x0067, 0x0068, 0x0068, 0x0069, 0x0069, 0x006A, 0x006A, 0x006B, 0x006B, 0x006C, 0x006C, 0x006D, 0x006D, 0x006E, 0x006E, 0x006F, 0x006F, 0x0070, 0x0070, 0x0071, 0x0071, 0x0072, 0x0072, 0x0073, 0x0073, 0x0074, 0x0074, 0x0075, 0x0075, 0x0076, 0x0076, 0x0077, 0x0077, 0x0078, 0x0078, 0x0079, 0x0079, 0x007A, 0x007A, 0x007B, 0x007B, 0x007C, 0x007C, 0x007D, 0x007D, 0x007E, 0x007E, 0x007F, 0x007F, 0x0080, 0x0080, 0x0081, 0x0081, 0x0082, 0x0082, 0x0083, 0x0083, 0x0084, 0x0084, 0x0085, 0x0085, 0x0086, 0x0086, 0x0087, 0x0087, 0x0088, 0x0088, 0x0089, 0x0089, 0x008A, 0x008A, 0x008B, 0x008B, 0x008C, 0x008C, 0x008D, 0x008D, 0x008E, 0x008E, 0x008F, 0x008F, 0x0090, 0x0090, 0x0091, 0x0091, 0x0092, 0x0092, 0x0093, 0x0093, 0x0094, 0x0094, 0x0095, 0x0095, 0x0096, 0x0096, 0x0097, 0x0097, 0x0098, 0x0098, 0x0099, 0x0099, 0x009A, 0x009A, 0x009B, 0x009B, 0x009C, 0x009C, 0x009D, 0x009D, 0x009E, 0x009E, 0x009F, 0x009F, 0x00A0, 0x00A0, 0x00A1, 0x00A1, 0x00A2, 0x00A2, 0x00A3, 0x00A3, 0x00A4, 0x00A4, 0x00A5, 0x00A5, 0x00A6, 0x00A6, 0x00A7, 0x00A7, 0x00A8, 0x00A8, 0x00A9, 0x00A9, 0x00AA, 0x00AA, 0x00AB, 0x00AB, 0x00AC, 0x00AC, 0x00AD, 0x00AD, 0x00AE, 0x00AE, 0x00AF, 0x00AF, 0x00B0, 0x00B0, 0x00B1, 0x00B1, 0x00B2, 0x00B2, 0x00B3, 0x00B3, 0x00B4, 0x00B4, 0x00B5, 0x00B5, 0x00B6, 0x00B6, 0x00B7, 0x00B7, 0x00B8, 0x00B8, 0x00B9, 0x00B9, 0x00BA, 0x00BA, 0x00BB, 0x00BB, 0x00BC, 0x00BC, 0x00BD, 0x00BD, 0x00BE, 0x00BE, 0x00BF, 0x00BF, 0x00C0, 0x00C0, 0x00C1, 0x00C1, 0x00C2, 0x00C2, 0x00C3, 0x00C3, 0x00C4, 0x00C4, 0x00C5, 0x00C5, 0x00C6, 0x00C6, 0x00C7, 0x00C7, 0x00C8, 0x00C8, 0x00C9, 0x00C9, 0x00CA, 0x00CA, 0x00CB, 0x00CB, 0x00CC, 0x00CC, 0x00CD, 0x00CD, 0x00CE, 0x00CE, 0x00CF, 0x00CF, 0x00D0, 0x00D0, 0x00D1, 0x00D1, 0x00D2, 0x00D2, 0x00D3, 0x00D3, 0x00D4, 0x00D4, 0x00D5, 0x00D5, 0x00D6, 0x00D6, 0x00D7, 0x00D7, 0x00D8, 0x00D8, 0x00D9, 0x00D9, 0x00DA, 0x00DA, 0x00DB, 0x00DB, 0x00DC, 0x00DC, 0x00DD, 0x00DD, 0x00DE, 0x00DE, 0x00DF, 0x00DF, 0x00E0, 0x00E0, 0x00E1, 0x00E1, 0x00E2, 0x00E2, 0x00E3, 0x00E3, 0x00E4, 0x00E4, 0x00E5, 0x00E5, 0x00E6, 0x00E6, 0x00E7, 0x00E7, 0x00E8, 0x00E8, 0x00E9, 0x00E9, 0x00EA, 0x00EA, 0x00EB, 0x00EB, 0x00EC, 0x00EC, 0x00ED, 0x00ED, 0x00EE, 0x00EE, 0x00EF, 0x00EF, 0x00F0, 0x00F0, 0x00F1, 0x00F1, 0x00F2, 0x00F2, 0x00F3, 0x00F3, 0x00F4, 0x00F4, 0x00F5, 0x00F5, 0x00F6, 0x00F6, 0x00F7, 0x00F7, 0x00F8, 0x00F8, 0x00F9, 0x00F9, 0x00FA, 0x00FA, 0x00FB, 0x00FB, 0x00FC, 0x00FC, 0x00FD, 0x00FD, 0x00FE, 0x00FE, 0x00FF, 0x00FF, 0x0100, 0x0100, 0x0101, 0x0101, 0x0102, 0x0102, 0x0103, 0x0103, 0x0104, 0x0104, 0x0105, 0x0105, 0x0106, 0x0106, 0x0107, 0x0107, 0x0108, 0x0108, 0x0109, 0x0109, 0x010A, 0x010A, 0x010B, 0x010B, 0x010C, 0x010C, 0x010D, 0x010D, 0x010E, 0x010E, 0x010F, 0x010F, 0x0110, 0x0110, 0x0111, 0x0111, 0x0112, 0x0112, 0x0113, 0x0113, 0x0114, 0x0114, 0x0115, 0x0115, 0x0116, 0x0116, 0x0117, 0x0117, 0x0118, 0x0118, 0x0119, 0x0119, 0x011A, 0x011A, 0x011B, 0x011B, 0x011C, 0x011C, 0x011D, 0x011D, 0x011E, 0x011E, 0x011F, 0x011F, 0x0120, 0x0120, 0x0121, 0x0121, 0x0122, 0x0122, 0x0123, 0x0123, 0x0124, 0x0124, 0x0125, 0x0125, 0x0126, 0x0126, 0x0127, 0x0127, 0x0128, 0x0128, 0x0129, 0x0129, 0x012A, 0x012A, 0x012B, 0x012B, 0x012C, 0x012C, 0x012D, 0x012D, 0x012E, 0x012E, 0x012F, 0x012F, 0x0130, 0x0130, 0x0131, 0x0131, 0x0132, 0x0132, 0x0133, 0x0133, 0x0134, 0x0134, 0x0135, 0x0135, 0x0136, 0x0136, 0x0137, 0x0137, 0x0138, 0x0138, 0x0139, 0x0139, 0x013A, 0x013A, 0x013B, 0x013B, 0x013C, 0x013C, 0x013D, 0x013D, 0x013E, 0x013E, 0x013F, 0x013F, 0x0140, 0x0140, 0x0141, 0x0141, 0x0142, 0x0142, 0x0143, 0x0143, 0x0144, 0x0144, 0x0145, 0x0145, 0x0146, 0x0146, 0x0147, 0x0147, 0x0148, 0x0148, 0x0149, 0x0149, 0x014A, 0x014A, 0x014B, 0x014B, 0x014C, 0x014C, 0x014D, 0x014D, 0x014E, 0x014E, 0x014F, 0x014F, 0x0150, 0x0150, 0x0151, 0x0151, 0x0152, 0x0152, 0x0153, 0x0153, 0x0154, 0x0154, 0x0155, 0x0155, 0x0156, 0x0156, 0x0157, 0x0157, 0x0158, 0x0158, 0x0159, 0x0159, 0x015A, 0x015A, 0x015B, 0x015B, 0x015C, 0x015C, 0x015D, 0x015D, 0x015E, 0x015E, 0x015F, 0x015F, 0x0160, 0x0160, 0x0161, 0x0161, 0x0162, 0x0162, 0x0163, 0x0163, 0x0164, 0x0164, 0x0165, 0x0165, 0x0166, 0x0166, 0x0167, 0x0167, 0x0168, 0x0168, 0x0169, 0x0169, 0x016A, 0x016A, 0x016B, 0x016B, 0x016C, 0x016C, 0x016D, 0x016D, 0x016E, 0x016E, 0x016F, 0x016F, 0x0170, 0x0170, 0x0171, 0x0171, 0x0172, 0x0172, 0x0173, 0x0173, 0x0174, 0x0174, 0x0175, 0x0175, 0x0176, 0x0176, 0x0177, 0x0177, 0x0178, 0x0178, 0x0179, 0x0179, 0x017A, 0x017A, 0x017B, 0x017B, 0x017C, 0x017C, 0x017D, 0x017D, 0x017E, 0x017E, 0x017F, 0x017F, 0x0180, 0x0180, 0x0181, 0x0181, 0x0182, 0x0182, 0x0183, 0x0183, 0x0184, 0x0184, 0x0185, 0x0185, 0x0186, 0x0186, 0x0187, 0x0187, 0x0188, 0x0188, 0x0189, 0x0189, 0x018A, 0x018A, 0x018B, 0x018B, 0x018C, 0x018C, 0x018D, 0x018D, 0x018E, 0x018E, 0x018F, 0x018F, 0x0190, 0x0190, 0x0191, 0x0191, 0x0192, 0x0192, 0x0193, 0x0193, 0x0194, 0x0194, 0x0195, 0x0195, 0x0196, 0x0196, 0x0197, 0x0197, 0x0198, 0x0198, 0x0199, 0x0199, 0x019A, 0x019A, 0x019B, 0x019B, 0x019C, 0x019C, 0x019D, 0x019D, 0x019E, 0x019E, 0x019F, 0x019F, 0x01A0, 0x01A0, 0x01A1, 0x01A1, 0x01A2, 0x01A2, 0x01A3, 0x01A3, 0x01A4, 0x01A4, 0x01A5, 0x01A5, 0x01A6, 0x01A6, 0x01A7, 0x01A7, 0x01A8, 0x01A8, 0x01A9, 0x01A9, 0x01AA, 0x01AA, 0x01AB, 0x01AB, 0x01AC, 0x01AC, 0x01AD, 0x01AD, 0x01AE, 0x01AE, 0x01AF, 0x01AF, 0x01B0, 0x01B0, 0x01B1, 0x01B1, 0x01B2, 0x01B2, 0x01B3, 0x01B3, 0x01B4, 0x01B4, 0x01B5, 0x01B5, 0x01B6, 0x01B6, 0x01B7, 0x01B7, 0x01B8, 0x01B8, 0x01B9, 0x01B9, 0x01BA, 0x01BA, 0x01BB, 0x01BB, 0x01BC, 0x01BC, 0x01BD, 0x01BD, 0x01BE, 0x01BE, 0x01BF, 0x01BF, 0x01C0, 0x01C0, 0x01C1, 0x01C1, 0x01C2, 0x01C2, 0x01C3, 0x01C3, 0x01C4, 0x01C4, 0x01C5, 0x01C5, 0x01C6, 0x01C6, 0x01C7, 0x01C7, 0x01C8, 0x01C8, 0x01C9, 0x01C9, 0x01CA, 0x01CA, 0x01CB, 0x01CB, 0x01CC, 0x01CC, 0x01CD, 0x01CD, 0x01CE, 0x01CE, 0x01CF, 0x01CF, 0x01D0, 0x01D0, 0x01D1, 0x01D1, 0x01D2, 0x01D2, 0x01D3, 0x01D3, 0x01D4, 0x01D4, 0x01D5, 0x01D5, 0x01D6, 0x01D6, 0x01D7, 0x01D7, 0x01D8, 0x01D8, 0x01D9, 0x01D9, 0x01DA, 0x01DA, 0x01DB, 0x01DB, 0x01DC, 0x01DC, 0x01DD, 0x01DD, 0x01DE, 0x01DE, 0x01DF, 0x01DF, 0x01E0, 0x01E0, 0x01E1, 0x01E1, 0x01E2, 0x01E2, 0x01E3, 0x01E3, 0x01E4, 0x01E4, 0x01E5, 0x01E5, 0x01E6, 0x01E6, 0x01E7, 0x01E7, 0x01E8, 0x01E8, 0x01E9, 0x01E9, 0x01EA, 0x01EA, 0x01EB, 0x01EB, 0x01EC, 0x01EC, 0x01ED, 0x01ED, 0x01EE, 0x01EE, 0x01EF, 0x01EF, 0x01F0, 0x01F0, 0x01F1, 0x01F1, 0x01F2, 0x01F2, 0x01F3, 0x01F3, 0x01F4, 0x01F4, 0x01F5, 0x01F5, 0x01F6, 0x01F6, 0x01F7, 0x01F7, 0x01F8, 0x01F8, 0x01F9, 0x01F9, 0x01FA, 0x01FA, 0x01FB, 0x01FB, 0x01FC, 0x01FC, 0x01FD, 0x01FD, 0x01FE, 0x01FE, 0x01FF, 0x01FF

},

{

0x0000, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0x0066, 0xFFFF, 0xFFFF, 0x0067, 0xFFFF, 0xFFFF, 0x0068, 0xFFFF, 0xFFFF, 0x0069, 0xFFFF, 0xFFFF, 0x006A, 0xFFFF, 0xFFFF, 0x006B, 0xFFFF, 0xFFFF, 0x006C, 0xFFFF, 0xFFFF, 0x006D, 0xFFFF, 0xFFFF, 0x006E, 0xFFFF, 0xFFFF, 0x006F, 0xFFFF, 0xFFFF, 0x0070, 0xFFFF, 0xFFFF, 0x0071, 0xFFFF, 0xFFFF, 0x0072, 0xFFFF, 0xFFFF, 0x0073, 0xFFFF, 0xFFFF, 0x0074, 0xFFFF, 0xFFFF, 0x0075, 0xFFFF, 0xFFFF, 0x0076, 0xFFFF, 0xFFFF, 0x0077, 0xFFFF, 0xFFFF, 0x0078, 0xFFFF, 0xFFFF, 0x0079, 0xFFFF, 0xFFFF, 0x007A, 0xFFFF, 0xFFFF, 0x007B, 0xFFFF, 0xFFFF, 0x007C, 0xFFFF, 0xFFFF, 0x007D, 0xFFFF, 0xFFFF, 0x007E, 0xFFFF, 0xFFFF, 0x007F, 0xFFFF, 0xFFFF, 0x0080, 0xFFFF, 0xFFFF, 0x0081, 0xFFFF, 0xFFFF, 0x0082, 0xFFFF, 0xFFFF, 0x0083, 0xFFFF, 0xFFFF, 0x0084, 0xFFFF, 0xFFFF, 0x0085, 0xFFFF, 0xFFFF, 0x0086, 0xFFFF, 0xFFFF, 0x0087, 0xFFFF, 0xFFFF, 0x0088, 0xFFFF, 0xFFFF, 0x0089, 0xFFFF, 0xFFFF, 0x008A, 0xFFFF, 0xFFFF, 0x008B, 0xFFFF, 0xFFFF, 0x008C, 0xFFFF, 0xFFFF, 0x008D, 0xFFFF, 0xFFFF, 0x008E, 0xFFFF, 0xFFFF, 0x008F, 0xFFFF, 0xFFFF, 0x0090, 0xFFFF, 0xFFFF, 0x0091, 0xFFFF, 0xFFFF, 0x0092, 0xFFFF, 0xFFFF, 0x0093, 0xFFFF, 0xFFFF, 0x0094, 0xFFFF, 0xFFFF, 0x0095, 0xFFFF, 0xFFFF, 0x0096, 0xFFFF, 0xFFFF, 0x0097, 0xFFFF, 0xFFFF, 0x0098, 0xFFFF, 0xFFFF, 0x0099, 0xFFFF, 0xFFFF, 0x009A, 0xFFFF, 0xFFFF, 0x009B, 0xFFFF, 0xFFFF, 0x009C, 0xFFFF, 0xFFFF, 0x009D, 0xFFFF, 0xFFFF, 0x009E, 0xFFFF, 0xFFFF, 0x009F, 0xFFFF, 0xFFFF, 0x00A0, 0xFFFF, 0xFFFF, 0x00A1, 0xFFFF, 0xFFFF, 0x00A2, 0xFFFF, 0xFFFF, 0x00A3, 0xFFFF, 0xFFFF, 0x00A4, 0xFFFF, 0xFFFF, 0x00A5, 0xFFFF, 0xFFFF, 0x00A6, 0xFFFF, 0xFFFF, 0x00A7, 0xFFFF, 0xFFFF, 0x00A8, 0xFFFF, 0xFFFF, 0x00A9, 0xFFFF, 0xFFFF, 0x00AA, 0xFFFF, 0xFFFF, 0x00AB, 0xFFFF, 0xFFFF, 0x00AC, 0xFFFF, 0xFFFF, 0x00AD, 0xFFFF, 0xFFFF, 0x00AE, 0xFFFF, 0xFFFF, 0x00AF, 0xFFFF, 0xFFFF, 0x00B0, 0xFFFF, 0xFFFF, 0x00B1, 0xFFFF, 0xFFFF, 0x00B2, 0xFFFF, 0xFFFF, 0x00B3, 0xFFFF, 0xFFFF, 0x00B4, 0xFFFF, 0xFFFF, 0x00B5, 0xFFFF, 0xFFFF, 0x00B6, 0xFFFF, 0xFFFF, 0x00B7, 0xFFFF, 0xFFFF, 0x00B8, 0xFFFF, 0xFFFF, 0x00B9, 0xFFFF, 0xFFFF, 0x00BA, 0xFFFF, 0xFFFF, 0x00BB, 0xFFFF, 0xFFFF, 0x00BC, 0xFFFF, 0xFFFF, 0x00BD, 0xFFFF, 0xFFFF, 0x00BE, 0xFFFF, 0xFFFF, 0x00BF, 0xFFFF, 0xFFFF, 0x00C0, 0xFFFF, 0xFFFF, 0x00C1, 0xFFFF, 0xFFFF, 0x00C2, 0xFFFF, 0xFFFF, 0x00C3, 0xFFFF, 0xFFFF, 0x00C4, 0xFFFF, 0xFFFF, 0x00C5, 0xFFFF, 0xFFFF, 0x00C6, 0xFFFF, 0xFFFF, 0x00C7, 0xFFFF, 0xFFFF, 0x00C8, 0xFFFF, 0xFFFF, 0x00C9, 0xFFFF, 0xFFFF, 0x00CA, 0xFFFF, 0xFFFF, 0x00CB, 0xFFFF, 0xFFFF, 0x00CC, 0xFFFF, 0xFFFF, 0x00CD, 0xFFFF, 0xFFFF, 0x00CE, 0xFFFF, 0xFFFF, 0x00CF, 0xFFFF, 0xFFFF, 0x00D0, 0xFFFF, 0xFFFF, 0x00D1, 0xFFFF, 0xFFFF, 0x00D2, 0xFFFF, 0xFFFF, 0x00D3, 0xFFFF, 0xFFFF, 0x00D4, 0xFFFF, 0xFFFF, 0x00D5, 0xFFFF, 0xFFFF, 0x00D6, 0xFFFF, 0xFFFF, 0x00D7, 0xFFFF, 0xFFFF, 0x00D8, 0xFFFF, 0xFFFF, 0x00D9, 0xFFFF, 0xFFFF, 0x00DA, 0xFFFF, 0xFFFF, 0x00DB, 0xFFFF, 0xFFFF, 0x00DC, 0xFFFF, 0xFFFF, 0x00DD, 0xFFFF, 0xFFFF, 0x00DE, 0xFFFF, 0xFFFF, 0x00DF, 0xFFFF, 0xFFFF, 0x00E0, 0xFFFF, 0xFFFF, 0x00E1, 0xFFFF, 0xFFFF, 0x00E2, 0xFFFF, 0xFFFF, 0x00E3, 0xFFFF, 0xFFFF, 0x00E4, 0xFFFF, 0xFFFF, 0x00E5, 0xFFFF, 0xFFFF, 0x00E6, 0xFFFF, 0xFFFF, 0x00E7, 0xFFFF, 0xFFFF, 0x00E8, 0xFFFF, 0xFFFF, 0x00E9, 0xFFFF, 0xFFFF, 0x00EA, 0xFFFF, 0xFFFF, 0x00EB, 0xFFFF, 0xFFFF, 0x00EC, 0xFFFF, 0xFFFF, 0x00ED, 0xFFFF, 0xFFFF, 0x00EE, 0xFFFF, 0xFFFF, 0x00EF, 0xFFFF, 0xFFFF, 0x00F0, 0xFFFF, 0xFFFF, 0x00F1, 0xFFFF, 0xFFFF, 0x00F2, 0xFFFF, 0xFFFF, 0x00F3, 0xFFFF, 0xFFFF, 0x00F4, 0xFFFF, 0xFFFF, 0x00F5, 0xFFFF, 0xFFFF, 0x00F6, 0xFFFF, 0xFFFF, 0x00F7, 0xFFFF, 0xFFFF, 0x00F8, 0xFFFF, 0xFFFF, 0x00F9, 0xFFFF, 0xFFFF, 0x00FA, 0xFFFF, 0xFFFF, 0x00FB, 0xFFFF, 0xFFFF, 0x00FC, 0xFFFF, 0xFFFF, 0x00FD, 0xFFFF, 0xFFFF, 0x00FE, 0xFFFF, 0xFFFF, 0x00FF, 0xFFFF, 0xFFFF, 0x0100, 0xFFFF, 0xFFFF, 0x0101, 0xFFFF, 0xFFFF, 0x0102, 0xFFFF, 0xFFFF, 0x0103, 0xFFFF, 0xFFFF, 0x0104, 0xFFFF, 0xFFFF, 0x0105, 0xFFFF, 0xFFFF, 0x0106, 0xFFFF, 0xFFFF, 0x0107, 0xFFFF, 0xFFFF, 0x0108, 0xFFFF, 0xFFFF, 0x0109, 0xFFFF, 0xFFFF, 0x010A, 0xFFFF, 0xFFFF, 0x010B, 0xFFFF, 0xFFFF, 0x010C, 0xFFFF, 0xFFFF, 0x010D, 0xFFFF, 0xFFFF, 0x010E, 0xFFFF, 0xFFFF, 0x010F, 0xFFFF, 0xFFFF, 0x0110, 0xFFFF, 0xFFFF, 0x0111, 0xFFFF, 0xFFFF, 0x0112, 0xFFFF, 0xFFFF, 0x0113, 0xFFFF, 0xFFFF, 0x0114, 0xFFFF, 0xFFFF, 0x0115, 0xFFFF, 0xFFFF, 0x0116, 0xFFFF, 0xFFFF, 0x0117, 0xFFFF, 0xFFFF, 0x0118, 0xFFFF, 0xFFFF, 0x0119, 0xFFFF, 0xFFFF, 0x011A, 0xFFFF, 0xFFFF, 0x011B, 0xFFFF, 0xFFFF, 0x011C, 0xFFFF, 0xFFFF, 0x011D, 0xFFFF, 0xFFFF, 0x011E, 0xFFFF, 0xFFFF, 0x011F, 0xFFFF, 0xFFFF, 0x0120, 0xFFFF, 0xFFFF, 0x0121, 0xFFFF, 0xFFFF, 0x0122, 0xFFFF, 0xFFFF, 0x0123, 0xFFFF, 0xFFFF, 0x0124, 0xFFFF, 0xFFFF, 0x0125, 0xFFFF, 0xFFFF, 0x0126, 0xFFFF, 0xFFFF, 0x0127, 0xFFFF, 0xFFFF, 0x0128, 0xFFFF, 0xFFFF, 0x0129, 0xFFFF, 0xFFFF, 0x012A, 0xFFFF, 0xFFFF, 0x012B, 0xFFFF, 0xFFFF, 0x012C, 0xFFFF, 0xFFFF, 0x012D, 0xFFFF, 0xFFFF, 0x012E, 0xFFFF, 0xFFFF, 0x012F, 0xFFFF, 0xFFFF, 0x0130, 0xFFFF, 0xFFFF, 0x0131, 0xFFFF, 0xFFFF, 0x0132, 0xFFFF, 0xFFFF, 0x0133, 0xFFFF, 0xFFFF, 0x0134, 0xFFFF, 0xFFFF, 0x0135, 0xFFFF, 0xFFFF, 0x0136, 0xFFFF, 0xFFFF, 0x0137, 0xFFFF, 0xFFFF, 0x0138, 0xFFFF, 0xFFFF, 0x0139, 0xFFFF, 0xFFFF, 0x013A, 0xFFFF, 0xFFFF, 0x013B, 0xFFFF, 0xFFFF, 0x013C, 0xFFFF, 0xFFFF, 0x013D, 0xFFFF, 0xFFFF, 0x013E, 0xFFFF, 0xFFFF, 0x013F, 0xFFFF, 0xFFFF, 0x0140, 0xFFFF, 0xFFFF, 0x0141, 0xFFFF, 0xFFFF, 0x0142, 0xFFFF, 0xFFFF, 0x0143, 0xFFFF, 0xFFFF, 0x0144, 0xFFFF, 0xFFFF, 0x0145, 0xFFFF, 0xFFFF, 0x0146, 0xFFFF, 0xFFFF, 0x0147, 0xFFFF, 0xFFFF, 0x0148, 0xFFFF, 0xFFFF, 0x0149, 0xFFFF, 0xFFFF, 0x014A, 0xFFFF, 0xFFFF, 0x014B, 0xFFFF, 0xFFFF, 0x014C, 0xFFFF, 0xFFFF, 0x014D, 0xFFFF, 0xFFFF, 0x014E, 0xFFFF, 0xFFFF, 0x014F, 0xFFFF, 0xFFFF, 0x0150, 0xFFFF, 0xFFFF, 0x0151, 0xFFFF, 0xFFFF, 0x0152, 0xFFFF, 0xFFFF, 0x0153, 0xFFFF, 0xFFFF, 0x0154, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0066, 0x0066, 0x0066, 0x0067, 0x0067, 0x0067, 0x0068, 0x0068, 0x0068, 0x0069, 0x0069, 0x0069, 0x006A, 0x006A, 0x006A, 0x006B, 0x006B, 0x006B, 0x006C, 0x006C, 0x006C, 0x006D, 0x006D, 0x006D, 0x006E, 0x006E, 0x006E, 0x006F, 0x006F, 0x006F, 0x0070, 0x0070, 0x0070, 0x0071, 0x0071, 0x0071, 0x0072, 0x0072, 0x0072, 0x0073, 0x0073, 0x0073, 0x0074, 0x0074, 0x0074, 0x0075, 0x0075, 0x0075, 0x0076, 0x0076, 0x0076, 0x0077, 0x0077, 0x0077, 0x0078, 0x0078, 0x0078, 0x0079, 0x0079, 0x0079, 0x007A, 0x007A, 0x007A, 0x007B, 0x007B, 0x007B, 0x007C, 0x007C, 0x007C, 0x007D, 0x007D, 0x007D, 0x007E, 0x007E, 0x007E, 0x007F, 0x007F, 0x007F, 0x0080, 0x0080, 0x0080, 0x0081, 0x0081, 0x0081, 0x0082, 0x0082, 0x0082, 0x0083, 0x0083, 0x0083, 0x0084, 0x0084, 0x0084, 0x0085, 0x0085, 0x0085, 0x0086, 0x0086, 0x0086, 0x0087, 0x0087, 0x0087, 0x0088, 0x0088, 0x0088, 0x0089, 0x0089, 0x0089, 0x008A, 0x008A, 0x008A, 0x008B, 0x008B, 0x008B, 0x008C, 0x008C, 0x008C, 0x008D, 0x008D, 0x008D, 0x008E, 0x008E, 0x008E, 0x008F, 0x008F, 0x008F, 0x0090, 0x0090, 0x0090, 0x0091, 0x0091, 0x0091, 0x0092, 0x0092, 0x0092, 0x0093, 0x0093, 0x0093, 0x0094, 0x0094, 0x0094, 0x0095, 0x0095, 0x0095, 0x0096, 0x0096, 0x0096, 0x0097, 0x0097, 0x0097, 0x0098, 0x0098, 0x0098, 0x0099, 0x0099, 0x0099, 0x009A, 0x009A, 0x009A, 0x009B, 0x009B, 0x009B, 0x009C, 0x009C, 0x009C, 0x009D, 0x009D, 0x009D, 0x009E, 0x009E, 0x009E, 0x009F, 0x009F, 0x009F, 0x00A0, 0x00A0, 0x00A0, 0x00A1, 0x00A1, 0x00A1, 0x00A2, 0x00A2, 0x00A2, 0x00A3, 0x00A3, 0x00A3, 0x00A4, 0x00A4, 0x00A4, 0x00A5, 0x00A5, 0x00A5, 0x00A6, 0x00A6, 0x00A6, 0x00A7, 0x00A7, 0x00A7, 0x00A8, 0x00A8, 0x00A8, 0x00A9, 0x00A9, 0x00A9, 0x00AA, 0x00AA, 0x00AA, 0x00AB, 0x00AB, 0x00AB, 0x00AC, 0x00AC, 0x00AC, 0x00AD, 0x00AD, 0x00AD, 0x00AE, 0x00AE, 0x00AE, 0x00AF, 0x00AF, 0x00AF, 0x00B0, 0x00B0, 0x00B0, 0x00B1, 0x00B1, 0x00B1, 0x00B2, 0x00B2, 0x00B2, 0x00B3, 0x00B3, 0x00B3, 0x00B4, 0x00B4, 0x00B4, 0x00B5, 0x00B5, 0x00B5, 0x00B6, 0x00B6, 0x00B6, 0x00B7, 0x00B7, 0x00B7, 0x00B8, 0x00B8, 0x00B8, 0x00B9, 0x00B9, 0x00B9, 0x00BA, 0x00BA, 0x00BA, 0x00BB, 0x00BB, 0x00BB, 0x00BC, 0x00BC, 0x00BC, 0x00BD, 0x00BD, 0x00BD, 0x00BE, 0x00BE, 0x00BE, 0x00BF, 0x00BF, 0x00BF, 0x00C0, 0x00C0, 0x00C0, 0x00C1, 0x00C1, 0x00C1, 0x00C2, 0x00C2, 0x00C2, 0x00C3, 0x00C3, 0x00C3, 0x00C4, 0x00C4, 0x00C4, 0x00C5, 0x00C5, 0x00C5, 0x00C6, 0x00C6, 0x00C6, 0x00C7, 0x00C7, 0x00C7, 0x00C8, 0x00C8, 0x00C8, 0x00C9, 0x00C9, 0x00C9, 0x00CA, 0x00CA, 0x00CA, 0x00CB, 0x00CB, 0x00CB, 0x00CC, 0x00CC, 0x00CC, 0x00CD, 0x00CD, 0x00CD, 0x00CE, 0x00CE, 0x00CE, 0x00CF, 0x00CF, 0x00CF, 0x00D0, 0x00D0, 0x00D0, 0x00D1, 0x00D1, 0x00D1, 0x00D2, 0x00D2, 0x00D2, 0x00D3, 0x00D3, 0x00D3, 0x00D4, 0x00D4, 0x00D4, 0x00D5, 0x00D5, 0x00D5, 0x00D6, 0x00D6, 0x00D6, 0x00D7, 0x00D7, 0x00D7, 0x00D8, 0x00D8, 0x00D8, 0x00D9, 0x00D9, 0x00D9, 0x00DA, 0x00DA, 0x00DA, 0x00DB, 0x00DB, 0x00DB, 0x00DC, 0x00DC, 0x00DC, 0x00DD, 0x00DD, 0x00DD, 0x00DE, 0x00DE, 0x00DE, 0x00DF, 0x00DF, 0x00DF, 0x00E0, 0x00E0, 0x00E0, 0x00E1, 0x00E1, 0x00E1, 0x00E2, 0x00E2, 0x00E2, 0x00E3, 0x00E3, 0x00E3, 0x00E4, 0x00E4, 0x00E4, 0x00E5, 0x00E5, 0x00E5, 0x00E6, 0x00E6, 0x00E6, 0x00E7, 0x00E7, 0x00E7, 0x00E8, 0x00E8, 0x00E8, 0x00E9, 0x00E9, 0x00E9, 0x00EA, 0x00EA, 0x00EA, 0x00EB, 0x00EB, 0x00EB, 0x00EC, 0x00EC, 0x00EC, 0x00ED, 0x00ED, 0x00ED, 0x00EE, 0x00EE, 0x00EE, 0x00EF, 0x00EF, 0x00EF, 0x00F0, 0x00F0, 0x00F0, 0x00F1, 0x00F1, 0x00F1, 0x00F2, 0x00F2, 0x00F2, 0x00F3, 0x00F3, 0x00F3, 0x00F4, 0x00F4, 0x00F4, 0x00F5, 0x00F5, 0x00F5, 0x00F6, 0x00F6, 0x00F6, 0x00F7, 0x00F7, 0x00F7, 0x00F8, 0x00F8, 0x00F8, 0x00F9, 0x00F9, 0x00F9, 0x00FA, 0x00FA, 0x00FA, 0x00FB, 0x00FB, 0x00FB, 0x00FC, 0x00FC, 0x00FC, 0x00FD, 0x00FD, 0x00FD, 0x00FE, 0x00FE, 0x00FE, 0x00FF, 0x00FF, 0x00FF, 0x0100, 0x0100, 0x0100, 0x0101, 0x0101, 0x0101, 0x0102, 0x0102, 0x0102, 0x0103, 0x0103, 0x0103, 0x0104, 0x0104, 0x0104, 0x0105, 0x0105, 0x0105, 0x0106, 0x0106, 0x0106, 0x0107, 0x0107, 0x0107, 0x0108, 0x0108, 0x0108, 0x0109, 0x0109, 0x0109, 0x010A, 0x010A, 0x010A, 0x010B, 0x010B, 0x010B, 0x010C, 0x010C, 0x010C, 0x010D, 0x010D, 0x010D, 0x010E, 0x010E, 0x010E, 0x010F, 0x010F, 0x010F, 0x0110, 0x0110, 0x0110, 0x0111, 0x0111, 0x0111, 0x0112, 0x0112, 0x0112, 0x0113, 0x0113, 0x0113, 0x0114, 0x0114, 0x0114, 0x0115, 0x0115, 0x0115, 0x0116, 0x0116, 0x0116, 0x0117, 0x0117, 0x0117, 0x0118, 0x0118, 0x0118, 0x0119, 0x0119, 0x0119, 0x011A, 0x011A, 0x011A, 0x011B, 0x011B, 0x011B, 0x011C, 0x011C, 0x011C, 0x011D, 0x011D, 0x011D, 0x011E, 0x011E, 0x011E, 0x011F, 0x011F, 0x011F, 0x0120, 0x0120, 0x0120, 0x0121, 0x0121, 0x0121, 0x0122, 0x0122, 0x0122, 0x0123, 0x0123, 0x0123, 0x0124, 0x0124, 0x0124, 0x0125, 0x0125, 0x0125, 0x0126, 0x0126, 0x0126, 0x0127, 0x0127, 0x0127, 0x0128, 0x0128, 0x0128, 0x0129, 0x0129, 0x0129, 0x012A, 0x012A, 0x012A, 0x012B, 0x012B, 0x012B, 0x012C, 0x012C, 0x012C, 0x012D, 0x012D, 0x012D, 0x012E, 0x012E, 0x012E, 0x012F, 0x012F, 0x012F, 0x0130, 0x0130, 0x0130, 0x0131, 0x0131, 0x0131, 0x0132, 0x0132, 0x0132, 0x0133, 0x0133, 0x0133, 0x0134, 0x0134, 0x0134, 0x0135, 0x0135, 0x0135, 0x0136, 0x0136, 0x0136, 0x0137, 0x0137, 0x0137, 0x0138, 0x0138, 0x0138, 0x0139, 0x0139, 0x0139, 0x013A, 0x013A, 0x013A, 0x013B, 0x013B, 0x013B, 0x013C, 0x013C, 0x013C, 0x013D, 0x013D, 0x013D, 0x013E, 0x013E, 0x013E, 0x013F, 0x013F, 0x013F, 0x0140, 0x0140, 0x0140, 0x0141, 0x0141, 0x0141, 0x0142, 0x0142, 0x0142, 0x0143, 0x0143, 0x0143, 0x0144, 0x0144, 0x0144, 0x0145, 0x0145, 0x0145, 0x0146, 0x0146, 0x0146, 0x0147, 0x0147, 0x0147, 0x0148, 0x0148, 0x0148, 0x0149, 0x0149, 0x0149, 0x014A, 0x014A, 0x014A, 0x014B, 0x014B, 0x014B, 0x014C, 0x014C, 0x014C, 0x014D, 0x014D, 0x014D, 0x014E, 0x014E, 0x014E, 0x014F, 0x014F, 0x014F, 0x0150, 0x0150, 0x0150, 0x0151, 0x0151, 0x0151, 0x0152, 0x0152, 0x0152, 0x0153, 0x0153, 0x0153, 0x0154, 0x0154, 0x0154, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0x0066, 0xFFFF, 0xFFFF, 0xFFFF, 0x0067, 0xFFFF, 0xFFFF, 0xFFFF, 0x0068, 0xFFFF, 0xFFFF, 0xFFFF, 0x0069, 0xFFFF, 0xFFFF, 0xFFFF, 0x006A, 0xFFFF, 0xFFFF, 0xFFFF, 0x006B, 0xFFFF, 0xFFFF, 0xFFFF, 0x006C, 0xFFFF, 0xFFFF, 0xFFFF, 0x006D, 0xFFFF, 0xFFFF, 0xFFFF, 0x006E, 0xFFFF, 0xFFFF, 0xFFFF, 0x006F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0070, 0xFFFF, 0xFFFF, 0xFFFF, 0x0071, 0xFFFF, 0xFFFF, 0xFFFF, 0x0072, 0xFFFF, 0xFFFF, 0xFFFF, 0x0073, 0xFFFF, 0xFFFF, 0xFFFF, 0x0074, 0xFFFF, 0xFFFF, 0xFFFF, 0x0075, 0xFFFF, 0xFFFF, 0xFFFF, 0x0076, 0xFFFF, 0xFFFF, 0xFFFF, 0x0077, 0xFFFF, 0xFFFF, 0xFFFF, 0x0078, 0xFFFF, 0xFFFF, 0xFFFF, 0x0079, 0xFFFF, 0xFFFF, 0xFFFF, 0x007A, 0xFFFF, 0xFFFF, 0xFFFF, 0x007B, 0xFFFF, 0xFFFF, 0xFFFF, 0x007C, 0xFFFF, 0xFFFF, 0xFFFF, 0x007D, 0xFFFF, 0xFFFF, 0xFFFF, 0x007E, 0xFFFF, 0xFFFF, 0xFFFF, 0x007F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0080, 0xFFFF, 0xFFFF, 0xFFFF, 0x0081, 0xFFFF, 0xFFFF, 0xFFFF, 0x0082, 0xFFFF, 0xFFFF, 0xFFFF, 0x0083, 0xFFFF, 0xFFFF, 0xFFFF, 0x0084, 0xFFFF, 0xFFFF, 0xFFFF, 0x0085, 0xFFFF, 0xFFFF, 0xFFFF, 0x0086, 0xFFFF, 0xFFFF, 0xFFFF, 0x0087, 0xFFFF, 0xFFFF, 0xFFFF, 0x0088, 0xFFFF, 0xFFFF, 0xFFFF, 0x0089, 0xFFFF, 0xFFFF, 0xFFFF, 0x008A, 0xFFFF, 0xFFFF, 0xFFFF, 0x008B, 0xFFFF, 0xFFFF, 0xFFFF, 0x008C, 0xFFFF, 0xFFFF, 0xFFFF, 0x008D, 0xFFFF, 0xFFFF, 0xFFFF, 0x008E, 0xFFFF, 0xFFFF, 0xFFFF, 0x008F, 0xFFFF, 0xFFFF, 0xFFFF, 0x0090, 0xFFFF, 0xFFFF, 0xFFFF, 0x0091, 0xFFFF, 0xFFFF, 0xFFFF, 0x0092, 0xFFFF, 0xFFFF, 0xFFFF, 0x0093, 0xFFFF, 0xFFFF, 0xFFFF, 0x0094, 0xFFFF, 0xFFFF, 0xFFFF, 0x0095, 0xFFFF, 0xFFFF, 0xFFFF, 0x0096, 0xFFFF, 0xFFFF, 0xFFFF, 0x0097, 0xFFFF, 0xFFFF, 0xFFFF, 0x0098, 0xFFFF, 0xFFFF, 0xFFFF, 0x0099, 0xFFFF, 0xFFFF, 0xFFFF, 0x009A, 0xFFFF, 0xFFFF, 0xFFFF, 0x009B, 0xFFFF, 0xFFFF, 0xFFFF, 0x009C, 0xFFFF, 0xFFFF, 0xFFFF, 0x009D, 0xFFFF, 0xFFFF, 0xFFFF, 0x009E, 0xFFFF, 0xFFFF, 0xFFFF, 0x009F, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A0, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A1, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A2, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A3, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A4, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A5, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A6, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A7, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A8, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A9, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AA, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AB, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AC, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AD, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AE, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B0, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B1, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B2, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B3, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B4, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B5, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B6, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B7, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B8, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B9, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BA, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BB, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BC, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BD, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BE, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C0, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C1, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C2, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C3, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C4, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C5, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C6, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C7, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C8, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C9, 0xFFFF, 0xFFFF, 0xFFFF, 0x00CA, 0xFFFF, 0xFFFF, 0xFFFF, 0x00CB, 0xFFFF, 0xFFFF, 0xFFFF, 0x00CC, 0xFFFF, 0xFFFF, 0xFFFF, 0x00CD, 0xFFFF, 0xFFFF, 0xFFFF, 0x00CE, 0xFFFF, 0xFFFF, 0xFFFF, 0x00CF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D0, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D1, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D2, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D3, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D4, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D5, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D6, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D7, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D8, 0xFFFF, 0xFFFF, 0xFFFF, 0x00D9, 0xFFFF, 0xFFFF, 0xFFFF, 0x00DA, 0xFFFF, 0xFFFF, 0xFFFF, 0x00DB, 0xFFFF, 0xFFFF, 0xFFFF, 0x00DC, 0xFFFF, 0xFFFF, 0xFFFF, 0x00DD, 0xFFFF, 0xFFFF, 0xFFFF, 0x00DE, 0xFFFF, 0xFFFF, 0xFFFF, 0x00DF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E0, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E1, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E2, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E3, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E4, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E5, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E6, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E7, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E8, 0xFFFF, 0xFFFF, 0xFFFF, 0x00E9, 0xFFFF, 0xFFFF, 0xFFFF, 0x00EA, 0xFFFF, 0xFFFF, 0xFFFF, 0x00EB, 0xFFFF, 0xFFFF, 0xFFFF, 0x00EC, 0xFFFF, 0xFFFF, 0xFFFF, 0x00ED, 0xFFFF, 0xFFFF, 0xFFFF, 0x00EE, 0xFFFF, 0xFFFF, 0xFFFF, 0x00EF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F0, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F1, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F2, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F3, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F4, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F5, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F6, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F7, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F8, 0xFFFF, 0xFFFF, 0xFFFF, 0x00F9, 0xFFFF, 0xFFFF, 0xFFFF, 0x00FA, 0xFFFF, 0xFFFF, 0xFFFF, 0x00FB, 0xFFFF, 0xFFFF, 0xFFFF, 0x00FC, 0xFFFF, 0xFFFF, 0xFFFF, 0x00FD, 0xFFFF, 0xFFFF, 0xFFFF, 0x00FE, 0xFFFF, 0xFFFF, 0xFFFF, 0x00FF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0065, 0x0066, 0x0066, 0x0066, 0x0066, 0x0067, 0x0067, 0x0067, 0x0067, 0x0068, 0x0068, 0x0068, 0x0068, 0x0069, 0x0069, 0x0069, 0x0069, 0x006A, 0x006A, 0x006A, 0x006A, 0x006B, 0x006B, 0x006B, 0x006B, 0x006C, 0x006C, 0x006C, 0x006C, 0x006D, 0x006D, 0x006D, 0x006D, 0x006E, 0x006E, 0x006E, 0x006E, 0x006F, 0x006F, 0x006F, 0x006F, 0x0070, 0x0070, 0x0070, 0x0070, 0x0071, 0x0071, 0x0071, 0x0071, 0x0072, 0x0072, 0x0072, 0x0072, 0x0073, 0x0073, 0x0073, 0x0073, 0x0074, 0x0074, 0x0074, 0x0074, 0x0075, 0x0075, 0x0075, 0x0075, 0x0076, 0x0076, 0x0076, 0x0076, 0x0077, 0x0077, 0x0077, 0x0077, 0x0078, 0x0078, 0x0078, 0x0078, 0x0079, 0x0079, 0x0079, 0x0079, 0x007A, 0x007A, 0x007A, 0x007A, 0x007B, 0x007B, 0x007B, 0x007B, 0x007C, 0x007C, 0x007C, 0x007C, 0x007D, 0x007D, 0x007D, 0x007D, 0x007E, 0x007E, 0x007E, 0x007E, 0x007F, 0x007F, 0x007F, 0x007F, 0x0080, 0x0080, 0x0080, 0x0080, 0x0081, 0x0081, 0x0081, 0x0081, 0x0082, 0x0082, 0x0082, 0x0082, 0x0083, 0x0083, 0x0083, 0x0083, 0x0084, 0x0084, 0x0084, 0x0084, 0x0085, 0x0085, 0x0085, 0x0085, 0x0086, 0x0086, 0x0086, 0x0086, 0x0087, 0x0087, 0x0087, 0x0087, 0x0088, 0x0088, 0x0088, 0x0088, 0x0089, 0x0089, 0x0089, 0x0089, 0x008A, 0x008A, 0x008A, 0x008A, 0x008B, 0x008B, 0x008B, 0x008B, 0x008C, 0x008C, 0x008C, 0x008C, 0x008D, 0x008D, 0x008D, 0x008D, 0x008E, 0x008E, 0x008E, 0x008E, 0x008F, 0x008F, 0x008F, 0x008F, 0x0090, 0x0090, 0x0090, 0x0090, 0x0091, 0x0091, 0x0091, 0x0091, 0x0092, 0x0092, 0x0092, 0x0092, 0x0093, 0x0093, 0x0093, 0x0093, 0x0094, 0x0094, 0x0094, 0x0094, 0x0095, 0x0095, 0x0095, 0x0095, 0x0096, 0x0096, 0x0096, 0x0096, 0x0097, 0x0097, 0x0097, 0x0097, 0x0098, 0x0098, 0x0098, 0x0098, 0x0099, 0x0099, 0x0099, 0x0099, 0x009A, 0x009A, 0x009A, 0x009A, 0x009B, 0x009B, 0x009B, 0x009B, 0x009C, 0x009C, 0x009C, 0x009C, 0x009D, 0x009D, 0x009D, 0x009D, 0x009E, 0x009E, 0x009E, 0x009E, 0x009F, 0x009F, 0x009F, 0x009F, 0x00A0, 0x00A0, 0x00A0, 0x00A0, 0x00A1, 0x00A1, 0x00A1, 0x00A1, 0x00A2, 0x00A2, 0x00A2, 0x00A2, 0x00A3, 0x00A3, 0x00A3, 0x00A3, 0x00A4, 0x00A4, 0x00A4, 0x00A4, 0x00A5, 0x00A5, 0x00A5, 0x00A5, 0x00A6, 0x00A6, 0x00A6, 0x00A6, 0x00A7, 0x00A7, 0x00A7, 0x00A7, 0x00A8, 0x00A8, 0x00A8, 0x00A8, 0x00A9, 0x00A9, 0x00A9, 0x00A9, 0x00AA, 0x00AA, 0x00AA, 0x00AA, 0x00AB, 0x00AB, 0x00AB, 0x00AB, 0x00AC, 0x00AC, 0x00AC, 0x00AC, 0x00AD, 0x00AD, 0x00AD, 0x00AD, 0x00AE, 0x00AE, 0x00AE, 0x00AE, 0x00AF, 0x00AF, 0x00AF, 0x00AF, 0x00B0, 0x00B0, 0x00B0, 0x00B0, 0x00B1, 0x00B1, 0x00B1, 0x00B1, 0x00B2, 0x00B2, 0x00B2, 0x00B2, 0x00B3, 0x00B3, 0x00B3, 0x00B3, 0x00B4, 0x00B4, 0x00B4, 0x00B4, 0x00B5, 0x00B5, 0x00B5, 0x00B5, 0x00B6, 0x00B6, 0x00B6, 0x00B6, 0x00B7, 0x00B7, 0x00B7, 0x00B7, 0x00B8, 0x00B8, 0x00B8, 0x00B8, 0x00B9, 0x00B9, 0x00B9, 0x00B9, 0x00BA, 0x00BA, 0x00BA, 0x00BA, 0x00BB, 0x00BB, 0x00BB, 0x00BB, 0x00BC, 0x00BC, 0x00BC, 0x00BC, 0x00BD, 0x00BD, 0x00BD, 0x00BD, 0x00BE, 0x00BE, 0x00BE, 0x00BE, 0x00BF, 0x00BF, 0x00BF, 0x00BF, 0x00C0, 0x00C0, 0x00C0, 0x00C0, 0x00C1, 0x00C1, 0x00C1, 0x00C1, 0x00C2, 0x00C2, 0x00C2, 0x00C2, 0x00C3, 0x00C3, 0x00C3, 0x00C3, 0x00C4, 0x00C4, 0x00C4, 0x00C4, 0x00C5, 0x00C5, 0x00C5, 0x00C5, 0x00C6, 0x00C6, 0x00C6, 0x00C6, 0x00C7, 0x00C7, 0x00C7, 0x00C7, 0x00C8, 0x00C8, 0x00C8, 0x00C8, 0x00C9, 0x00C9, 0x00C9, 0x00C9, 0x00CA, 0x00CA, 0x00CA, 0x00CA, 0x00CB, 0x00CB, 0x00CB, 0x00CB, 0x00CC, 0x00CC, 0x00CC, 0x00CC, 0x00CD, 0x00CD, 0x00CD, 0x00CD, 0x00CE, 0x00CE, 0x00CE, 0x00CE, 0x00CF, 0x00CF, 0x00CF, 0x00CF, 0x00D0, 0x00D0, 0x00D0, 0x00D0, 0x00D1, 0x00D1, 0x00D1, 0x00D1, 0x00D2, 0x00D2, 0x00D2, 0x00D2, 0x00D3, 0x00D3, 0x00D3, 0x00D3, 0x00D4, 0x00D4, 0x00D4, 0x00D4, 0x00D5, 0x00D5, 0x00D5, 0x00D5, 0x00D6, 0x00D6, 0x00D6, 0x00D6, 0x00D7, 0x00D7, 0x00D7, 0x00D7, 0x00D8, 0x00D8, 0x00D8, 0x00D8, 0x00D9, 0x00D9, 0x00D9, 0x00D9, 0x00DA, 0x00DA, 0x00DA, 0x00DA, 0x00DB, 0x00DB, 0x00DB, 0x00DB, 0x00DC, 0x00DC, 0x00DC, 0x00DC, 0x00DD, 0x00DD, 0x00DD, 0x00DD, 0x00DE, 0x00DE, 0x00DE, 0x00DE, 0x00DF, 0x00DF, 0x00DF, 0x00DF, 0x00E0, 0x00E0, 0x00E0, 0x00E0, 0x00E1, 0x00E1, 0x00E1, 0x00E1, 0x00E2, 0x00E2, 0x00E2, 0x00E2, 0x00E3, 0x00E3, 0x00E3, 0x00E3, 0x00E4, 0x00E4, 0x00E4, 0x00E4, 0x00E5, 0x00E5, 0x00E5, 0x00E5, 0x00E6, 0x00E6, 0x00E6, 0x00E6, 0x00E7, 0x00E7, 0x00E7, 0x00E7, 0x00E8, 0x00E8, 0x00E8, 0x00E8, 0x00E9, 0x00E9, 0x00E9, 0x00E9, 0x00EA, 0x00EA, 0x00EA, 0x00EA, 0x00EB, 0x00EB, 0x00EB, 0x00EB, 0x00EC, 0x00EC, 0x00EC, 0x00EC, 0x00ED, 0x00ED, 0x00ED, 0x00ED, 0x00EE, 0x00EE, 0x00EE, 0x00EE, 0x00EF, 0x00EF, 0x00EF, 0x00EF, 0x00F0, 0x00F0, 0x00F0, 0x00F0, 0x00F1, 0x00F1, 0x00F1, 0x00F1, 0x00F2, 0x00F2, 0x00F2, 0x00F2, 0x00F3, 0x00F3, 0x00F3, 0x00F3, 0x00F4, 0x00F4, 0x00F4, 0x00F4, 0x00F5, 0x00F5, 0x00F5, 0x00F5, 0x00F6, 0x00F6, 0x00F6, 0x00F6, 0x00F7, 0x00F7, 0x00F7, 0x00F7, 0x00F8, 0x00F8, 0x00F8, 0x00F8, 0x00F9, 0x00F9, 0x00F9, 0x00F9, 0x00FA, 0x00FA, 0x00FA, 0x00FA, 0x00FB, 0x00FB, 0x00FB, 0x00FB, 0x00FC, 0x00FC, 0x00FC, 0x00FC, 0x00FD, 0x00FD, 0x00FD, 0x00FD, 0x00FE, 0x00FE, 0x00FE, 0x00FE, 0x00FF, 0x00FF, 0x00FF, 0x00FF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0066, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0067, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0068, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0069, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0070, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0071, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0072, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0073, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0074, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0075, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0076, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0077, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0078, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0079, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0080, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0081, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0082, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0083, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0084, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0085, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0086, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0087, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0088, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0089, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0090, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0091, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0092, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0093, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0094, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0095, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0096, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0097, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0098, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0099, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A0, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A1, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A2, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A3, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A4, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A5, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A6, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A7, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A8, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A9, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AA, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AB, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AC, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AD, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AE, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00AF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B0, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B1, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B2, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B3, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B4, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B5, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B6, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B7, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B8, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00B9, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BA, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BB, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BC, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BD, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BE, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00BF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C0, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C1, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C2, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C3, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C4, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C5, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C6, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C7, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C8, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00C9, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00CA, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00CB, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F, 0x0080, 0x0080, 0x0080, 0x0080, 0x0080, 0x0081, 0x0081, 0x0081, 0x0081, 0x0081, 0x0082, 0x0082, 0x0082, 0x0082, 0x0082, 0x0083, 0x0083, 0x0083, 0x0083, 0x0083, 0x0084, 0x0084, 0x0084, 0x0084, 0x0084, 0x0085, 0x0085, 0x0085, 0x0085, 0x0085, 0x0086, 0x0086, 0x0086, 0x0086, 0x0086, 0x0087, 0x0087, 0x0087, 0x0087, 0x0087, 0x0088, 0x0088, 0x0088, 0x0088, 0x0088, 0x0089, 0x0089, 0x0089, 0x0089, 0x0089, 0x008A, 0x008A, 0x008A, 0x008A, 0x008A, 0x008B, 0x008B, 0x008B, 0x008B, 0x008B, 0x008C, 0x008C, 0x008C, 0x008C, 0x008C, 0x008D, 0x008D, 0x008D, 0x008D, 0x008D, 0x008E, 0x008E, 0x008E, 0x008E, 0x008E, 0x008F, 0x008F, 0x008F, 0x008F, 0x008F, 0x0090, 0x0090, 0x0090, 0x0090, 0x0090, 0x0091, 0x0091, 0x0091, 0x0091, 0x0091, 0x0092, 0x0092, 0x0092, 0x0092, 0x0092, 0x0093, 0x0093, 0x0093, 0x0093, 0x0093, 0x0094, 0x0094, 0x0094, 0x0094, 0x0094, 0x0095, 0x0095, 0x0095, 0x0095, 0x0095, 0x0096, 0x0096, 0x0096, 0x0096, 0x0096, 0x0097, 0x0097, 0x0097, 0x0097, 0x0097, 0x0098, 0x0098, 0x0098, 0x0098, 0x0098, 0x0099, 0x0099, 0x0099, 0x0099, 0x0099, 0x009A, 0x009A, 0x009A, 0x009A, 0x009A, 0x009B, 0x009B, 0x009B, 0x009B, 0x009B, 0x009C, 0x009C, 0x009C, 0x009C, 0x009C, 0x009D, 0x009D, 0x009D, 0x009D, 0x009D, 0x009E, 0x009E, 0x009E, 0x009E, 0x009E, 0x009F, 0x009F, 0x009F, 0x009F, 0x009F, 0x00A0, 0x00A0, 0x00A0, 0x00A0, 0x00A0, 0x00A1, 0x00A1, 0x00A1, 0x00A1, 0x00A1, 0x00A2, 0x00A2, 0x00A2, 0x00A2, 0x00A2, 0x00A3, 0x00A3, 0x00A3, 0x00A3, 0x00A3, 0x00A4, 0x00A4, 0x00A4, 0x00A4, 0x00A4, 0x00A5, 0x00A5, 0x00A5, 0x00A5, 0x00A5, 0x00A6, 0x00A6, 0x00A6, 0x00A6, 0x00A6, 0x00A7, 0x00A7, 0x00A7, 0x00A7, 0x00A7, 0x00A8, 0x00A8, 0x00A8, 0x00A8, 0x00A8, 0x00A9, 0x00A9, 0x00A9, 0x00A9, 0x00A9, 0x00AA, 0x00AA, 0x00AA, 0x00AA, 0x00AA, 0x00AB, 0x00AB, 0x00AB, 0x00AB, 0x00AB, 0x00AC, 0x00AC, 0x00AC, 0x00AC, 0x00AC, 0x00AD, 0x00AD, 0x00AD, 0x00AD, 0x00AD, 0x00AE, 0x00AE, 0x00AE, 0x00AE, 0x00AE, 0x00AF, 0x00AF, 0x00AF, 0x00AF, 0x00AF, 0x00B0, 0x00B0, 0x00B0, 0x00B0, 0x00B0, 0x00B1, 0x00B1, 0x00B1, 0x00B1, 0x00B1, 0x00B2, 0x00B2, 0x00B2, 0x00B2, 0x00B2, 0x00B3, 0x00B3, 0x00B3, 0x00B3, 0x00B3, 0x00B4, 0x00B4, 0x00B4, 0x00B4, 0x00B4, 0x00B5, 0x00B5, 0x00B5, 0x00B5, 0x00B5, 0x00B6, 0x00B6, 0x00B6, 0x00B6, 0x00B6, 0x00B7, 0x00B7, 0x00B7, 0x00B7, 0x00B7, 0x00B8, 0x00B8, 0x00B8, 0x00B8, 0x00B8, 0x00B9, 0x00B9, 0x00B9, 0x00B9, 0x00B9, 0x00BA, 0x00BA, 0x00BA, 0x00BA, 0x00BA, 0x00BB, 0x00BB, 0x00BB, 0x00BB, 0x00BB, 0x00BC, 0x00BC, 0x00BC, 0x00BC, 0x00BC, 0x00BD, 0x00BD, 0x00BD, 0x00BD, 0x00BD, 0x00BE, 0x00BE, 0x00BE, 0x00BE, 0x00BE, 0x00BF, 0x00BF, 0x00BF, 0x00BF, 0x00BF, 0x00C0, 0x00C0, 0x00C0, 0x00C0, 0x00C0, 0x00C1, 0x00C1, 0x00C1, 0x00C1, 0x00C1, 0x00C2, 0x00C2, 0x00C2, 0x00C2, 0x00C2, 0x00C3, 0x00C3, 0x00C3, 0x00C3, 0x00C3, 0x00C4, 0x00C4, 0x00C4, 0x00C4, 0x00C4, 0x00C5, 0x00C5, 0x00C5, 0x00C5, 0x00C5, 0x00C6, 0x00C6, 0x00C6, 0x00C6, 0x00C6, 0x00C7, 0x00C7, 0x00C7, 0x00C7, 0x00C7, 0x00C8, 0x00C8, 0x00C8, 0x00C8, 0x00C8, 0x00C9, 0x00C9, 0x00C9, 0x00C9, 0x00C9, 0x00CA, 0x00CA, 0x00CA, 0x00CA, 0x00CA, 0x00CB, 0x00CB, 0x00CB, 0x00CB, 0x00CB, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0066, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0067, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0068, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0069, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0070, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0071, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0072, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0073, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0074, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0075, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0076, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0077, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0078, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0079, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0080, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0081, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0082, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0083, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0084, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0085, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0086, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0087, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0088, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0089, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0090, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0091, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0092, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0093, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0094, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0095, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0096, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0097, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0098, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0099, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x009F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A0, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A1, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A2, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A3, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A4, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A5, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A6, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A7, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A8, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x00A9, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F, 0x0080, 0x0080, 0x0080, 0x0080, 0x0080, 0x0080, 0x0081, 0x0081, 0x0081, 0x0081, 0x0081, 0x0081, 0x0082, 0x0082, 0x0082, 0x0082, 0x0082, 0x0082, 0x0083, 0x0083, 0x0083, 0x0083, 0x0083, 0x0083, 0x0084, 0x0084, 0x0084, 0x0084, 0x0084, 0x0084, 0x0085, 0x0085, 0x0085, 0x0085, 0x0085, 0x0085, 0x0086, 0x0086, 0x0086, 0x0086, 0x0086, 0x0086, 0x0087, 0x0087, 0x0087, 0x0087, 0x0087, 0x0087, 0x0088, 0x0088, 0x0088, 0x0088, 0x0088, 0x0088, 0x0089, 0x0089, 0x0089, 0x0089, 0x0089, 0x0089, 0x008A, 0x008A, 0x008A, 0x008A, 0x008A, 0x008A, 0x008B, 0x008B, 0x008B, 0x008B, 0x008B, 0x008B, 0x008C, 0x008C, 0x008C, 0x008C, 0x008C, 0x008C, 0x008D, 0x008D, 0x008D, 0x008D, 0x008D, 0x008D, 0x008E, 0x008E, 0x008E, 0x008E, 0x008E, 0x008E, 0x008F, 0x008F, 0x008F, 0x008F, 0x008F, 0x008F, 0x0090, 0x0090, 0x0090, 0x0090, 0x0090, 0x0090, 0x0091, 0x0091, 0x0091, 0x0091, 0x0091, 0x0091, 0x0092, 0x0092, 0x0092, 0x0092, 0x0092, 0x0092, 0x0093, 0x0093, 0x0093, 0x0093, 0x0093, 0x0093, 0x0094, 0x0094, 0x0094, 0x0094, 0x0094, 0x0094, 0x0095, 0x0095, 0x0095, 0x0095, 0x0095, 0x0095, 0x0096, 0x0096, 0x0096, 0x0096, 0x0096, 0x0096, 0x0097, 0x0097, 0x0097, 0x0097, 0x0097, 0x0097, 0x0098, 0x0098, 0x0098, 0x0098, 0x0098, 0x0098, 0x0099, 0x0099, 0x0099, 0x0099, 0x0099, 0x0099, 0x009A, 0x009A, 0x009A, 0x009A, 0x009A, 0x009A, 0x009B, 0x009B, 0x009B, 0x009B, 0x009B, 0x009B, 0x009C, 0x009C, 0x009C, 0x009C, 0x009C, 0x009C, 0x009D, 0x009D, 0x009D, 0x009D, 0x009D, 0x009D, 0x009E, 0x009E, 0x009E, 0x009E, 0x009E, 0x009E, 0x009F, 0x009F, 0x009F, 0x009F, 0x009F, 0x009F, 0x00A0, 0x00A0, 0x00A0, 0x00A0, 0x00A0, 0x00A0, 0x00A1, 0x00A1, 0x00A1, 0x00A1, 0x00A1, 0x00A1, 0x00A2, 0x00A2, 0x00A2, 0x00A2, 0x00A2, 0x00A2, 0x00A3, 0x00A3, 0x00A3, 0x00A3, 0x00A3, 0x00A3, 0x00A4, 0x00A4, 0x00A4, 0x00A4, 0x00A4, 0x00A4, 0x00A5, 0x00A5, 0x00A5, 0x00A5, 0x00A5, 0x00A5, 0x00A6, 0x00A6, 0x00A6, 0x00A6, 0x00A6, 0x00A6, 0x00A7, 0x00A7, 0x00A7, 0x00A7, 0x00A7, 0x00A7, 0x00A8, 0x00A8, 0x00A8, 0x00A8, 0x00A8, 0x00A8, 0x00A9, 0x00A9, 0x00A9, 0x00A9, 0x00A9, 0x00A9, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0066, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0067, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0068, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0069, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0070, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0071, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0072, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0073, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0074, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0075, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0076, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0077, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0078, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0079, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0080, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0081, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0082, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0083, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0084, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0085, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0086, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0087, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0088, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0089, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x008F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0090, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0091, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F, 0x0080, 0x0080, 0x0080, 0x0080, 0x0080, 0x0080, 0x0080, 0x0081, 0x0081, 0x0081, 0x0081, 0x0081, 0x0081, 0x0081, 0x0082, 0x0082, 0x0082, 0x0082, 0x0082, 0x0082, 0x0082, 0x0083, 0x0083, 0x0083, 0x0083, 0x0083, 0x0083, 0x0083, 0x0084, 0x0084, 0x0084, 0x0084, 0x0084, 0x0084, 0x0084, 0x0085, 0x0085, 0x0085, 0x0085, 0x0085, 0x0085, 0x0085, 0x0086, 0x0086, 0x0086, 0x0086, 0x0086, 0x0086, 0x0086, 0x0087, 0x0087, 0x0087, 0x0087, 0x0087, 0x0087, 0x0087, 0x0088, 0x0088, 0x0088, 0x0088, 0x0088, 0x0088, 0x0088, 0x0089, 0x0089, 0x0089, 0x0089, 0x0089, 0x0089, 0x0089, 0x008A, 0x008A, 0x008A, 0x008A, 0x008A, 0x008A, 0x008A, 0x008B, 0x008B, 0x008B, 0x008B, 0x008B, 0x008B, 0x008B, 0x008C, 0x008C, 0x008C, 0x008C, 0x008C, 0x008C, 0x008C, 0x008D, 0x008D, 0x008D, 0x008D, 0x008D, 0x008D, 0x008D, 0x008E, 0x008E, 0x008E, 0x008E, 0x008E, 0x008E, 0x008E, 0x008F, 0x008F, 0x008F, 0x008F, 0x008F, 0x008F, 0x008F, 0x0090, 0x0090, 0x0090, 0x0090, 0x0090, 0x0090, 0x0090, 0x0091, 0x0091, 0x0091, 0x0091, 0x0091, 0x0091, 0x0091, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0066, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0067, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0068, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0069, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0070, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0071, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0072, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0073, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0074, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0075, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0076, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0077, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0078, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0079, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x007F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0071, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0072, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0073, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0074, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0075, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0076, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0077, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0078, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x0079, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007A, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007B, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007C, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007D, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007E, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F, 0x007F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0066, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0067, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0068, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0069, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x006F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0070, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0066, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0067, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0068, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x0069, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006A, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006B, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006C, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006D, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006E, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x006F, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0x0070, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0060, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0061, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0062, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0063, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0064, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005D, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005E, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x005F, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0060, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0061, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0062, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0063, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0064, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0x0065, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0055, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0056, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0057, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0058, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0059, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x005C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0055, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0056, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0057, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0058, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x0059, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005A, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005B, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0x005C, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0050, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0051, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0052, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0053, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004E, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x004F, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0050, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0051, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0052, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0053, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0x0054, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0049, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x0049, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004A, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004B, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004C, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0x004D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0044, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0045, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0046, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0047, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0048, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0044, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0045, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0046, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0047, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0x0048, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0040, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0041, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0042, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0040, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0041, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0042, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0x0043, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003C, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003D, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003E, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F, 0x003F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0038, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0039, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0038, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x0039, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003A, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0x003B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0035, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0036, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0035, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0036, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0x0037, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0033, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0033, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0x0034, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0030, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0031, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0030, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0031, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0x0032, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002E, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0x002F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002C, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0x002D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002A, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0x002B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0028, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0028, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0x0029, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0x0027, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0025, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0025, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0x0026, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0x0024, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0x0023, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0x0022, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0x0021, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0020, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0x0020, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

}

};

const BVUnit HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::invalidBitsData[HeapConstants::BucketCount][SmallHeapBlockT<SmallAllocationBlockAttributes>::SmallHeapBlockBitVector::wordCount] = {

{

0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000

},

{

0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA, 0xAAAAAAAAAAAAAAAA

},

{

0x6DB6DB6DB6DB6DB6, 0xB6DB6DB6DB6DB6DB, 0xDB6DB6DB6DB6DB6D, 0x6DB6DB6DB6DB6DB6, 0xB6DB6DB6DB6DB6DB, 0xDB6DB6DB6DB6DB6D, 0x6DB6DB6DB6DB6DB6, 0xB6DB6DB6DB6DB6DB, 0xDB6DB6DB6DB6DB6D, 0x6DB6DB6DB6DB6DB6, 0xB6DB6DB6DB6DB6DB, 0xDB6DB6DB6DB6DB6D, 0x6DB6DB6DB6DB6DB6, 0xB6DB6DB6DB6DB6DB, 0xDB6DB6DB6DB6DB6D, 0xEDB6DB6DB6DB6DB6

},

{

0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE, 0xEEEEEEEEEEEEEEEE

},

{

0xEF7BDEF7BDEF7BDE, 0xDEF7BDEF7BDEF7BD, 0xBDEF7BDEF7BDEF7B, 0x7BDEF7BDEF7BDEF7, 0xF7BDEF7BDEF7BDEF, 0xEF7BDEF7BDEF7BDE, 0xDEF7BDEF7BDEF7BD, 0xBDEF7BDEF7BDEF7B, 0x7BDEF7BDEF7BDEF7, 0xF7BDEF7BDEF7BDEF, 0xEF7BDEF7BDEF7BDE, 0xDEF7BDEF7BDEF7BD, 0xBDEF7BDEF7BDEF7B, 0x7BDEF7BDEF7BDEF7, 0xF7BDEF7BDEF7BDEF, 0xFF7BDEF7BDEF7BDE

},

{

0xEFBEFBEFBEFBEFBE, 0xBEFBEFBEFBEFBEFB, 0xFBEFBEFBEFBEFBEF, 0xEFBEFBEFBEFBEFBE, 0xBEFBEFBEFBEFBEFB, 0xFBEFBEFBEFBEFBEF, 0xEFBEFBEFBEFBEFBE, 0xBEFBEFBEFBEFBEFB, 0xFBEFBEFBEFBEFBEF, 0xEFBEFBEFBEFBEFBE, 0xBEFBEFBEFBEFBEFB, 0xFBEFBEFBEFBEFBEF, 0xEFBEFBEFBEFBEFBE, 0xBEFBEFBEFBEFBEFB, 0xFBEFBEFBEFBEFBEF, 0xFFBEFBEFBEFBEFBE

},

{

0x7EFDFBF7EFDFBF7E, 0xBF7EFDFBF7EFDFBF, 0xDFBF7EFDFBF7EFDF, 0xEFDFBF7EFDFBF7EF, 0xF7EFDFBF7EFDFBF7, 0xFBF7EFDFBF7EFDFB, 0xFDFBF7EFDFBF7EFD, 0x7EFDFBF7EFDFBF7E, 0xBF7EFDFBF7EFDFBF, 0xDFBF7EFDFBF7EFDF, 0xEFDFBF7EFDFBF7EF, 0xF7EFDFBF7EFDFBF7, 0xFBF7EFDFBF7EFDFB, 0xFDFBF7EFDFBF7EFD, 0x7EFDFBF7EFDFBF7E, 0xFF7EFDFBF7EFDFBF

},

{

0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE, 0xFEFEFEFEFEFEFEFE

},

{

0x7FBFDFEFF7FBFDFE, 0xBFDFEFF7FBFDFEFF, 0xDFEFF7FBFDFEFF7F, 0xEFF7FBFDFEFF7FBF, 0xF7FBFDFEFF7FBFDF, 0xFBFDFEFF7FBFDFEF, 0xFDFEFF7FBFDFEFF7, 0xFEFF7FBFDFEFF7FB, 0xFF7FBFDFEFF7FBFD, 0x7FBFDFEFF7FBFDFE, 0xBFDFEFF7FBFDFEFF, 0xDFEFF7FBFDFEFF7F, 0xEFF7FBFDFEFF7FBF, 0xF7FBFDFEFF7FBFDF, 0xFBFDFEFF7FBFDFEF, 0xFFFEFF7FBFDFEFF7

},

{

0xEFFBFEFFBFEFFBFE, 0xFEFFBFEFFBFEFFBF, 0xBFEFFBFEFFBFEFFB, 0xFBFEFFBFEFFBFEFF, 0xFFBFEFFBFEFFBFEF, 0xEFFBFEFFBFEFFBFE, 0xFEFFBFEFFBFEFFBF, 0xBFEFFBFEFFBFEFFB, 0xFBFEFFBFEFFBFEFF, 0xFFBFEFFBFEFFBFEF, 0xEFFBFEFFBFEFFBFE, 0xFEFFBFEFFBFEFFBF, 0xBFEFFBFEFFBFEFFB, 0xFBFEFFBFEFFBFEFF, 0xFFBFEFFBFEFFBFEF, 0xFFFBFEFFBFEFFBFE

},

{

0xFF7FEFFDFFBFF7FE, 0xFDFFBFF7FEFFDFFB, 0xF7FEFFDFFBFF7FEF, 0xDFFBFF7FEFFDFFBF, 0x7FEFFDFFBFF7FEFF, 0xFFBFF7FEFFDFFBFF, 0xFEFFDFFBFF7FEFFD, 0xFBFF7FEFFDFFBFF7, 0xEFFDFFBFF7FEFFDF, 0xBFF7FEFFDFFBFF7F, 0xFFDFFBFF7FEFFDFF, 0xFF7FEFFDFFBFF7FE, 0xFDFFBFF7FEFFDFFB, 0xF7FEFFDFFBFF7FEF, 0xDFFBFF7FEFFDFFBF, 0xFFEFFDFFBFF7FEFF

},

{

0xEFFEFFEFFEFFEFFE, 0xFEFFEFFEFFEFFEFF, 0xFFEFFEFFEFFEFFEF, 0xEFFEFFEFFEFFEFFE, 0xFEFFEFFEFFEFFEFF, 0xFFEFFEFFEFFEFFEF, 0xEFFEFFEFFEFFEFFE, 0xFEFFEFFEFFEFFEFF, 0xFFEFFEFFEFFEFFEF, 0xEFFEFFEFFEFFEFFE, 0xFEFFEFFEFFEFFEFF, 0xFFEFFEFFEFFEFFEF, 0xEFFEFFEFFEFFEFFE, 0xFEFFEFFEFFEFFEFF, 0xFFEFFEFFEFFEFFEF, 0xFFFEFFEFFEFFEFFE

},

{

0xFFEFFF7FFBFFDFFE, 0xFFDFFEFFF7FFBFFD, 0xFFBFFDFFEFFF7FFB, 0xFF7FFBFFDFFEFFF7, 0xFEFFF7FFBFFDFFEF, 0xFDFFEFFF7FFBFFDF, 0xFBFFDFFEFFF7FFBF, 0xF7FFBFFDFFEFFF7F, 0xEFFF7FFBFFDFFEFF, 0xDFFEFFF7FFBFFDFF, 0xBFFDFFEFFF7FFBFF, 0x7FFBFFDFFEFFF7FF, 0xFFF7FFBFFDFFEFFF, 0xFFEFFF7FFBFFDFFE, 0xFFDFFEFFF7FFBFFD, 0xFFFFFDFFEFFF7FFB

},

{

0xFEFFFBFFEFFFBFFE, 0xBFFEFFFBFFEFFFBF, 0xFFBFFEFFFBFFEFFF, 0xEFFFBFFEFFFBFFEF, 0xFFEFFFBFFEFFFBFF, 0xFBFFEFFFBFFEFFFB, 0xFFFBFFEFFFBFFEFF, 0xFEFFFBFFEFFFBFFE, 0xBFFEFFFBFFEFFFBF, 0xFFBFFEFFFBFFEFFF, 0xEFFFBFFEFFFBFFEF, 0xFFEFFFBFFEFFFBFF, 0xFBFFEFFFBFFEFFFB, 0xFFFBFFEFFFBFFEFF, 0xFEFFFBFFEFFFBFFE, 0xFFFEFFFBFFEFFFBF

},

{

0xEFFFDFFFBFFF7FFE, 0xFEFFFDFFFBFFF7FF, 0xFFEFFFDFFFBFFF7F, 0x7FFEFFFDFFFBFFF7, 0xF7FFEFFFDFFFBFFF, 0xFF7FFEFFFDFFFBFF, 0xFFF7FFEFFFDFFFBF, 0xBFFF7FFEFFFDFFFB, 0xFBFFF7FFEFFFDFFF, 0xFFBFFF7FFEFFFDFF, 0xFFFBFFF7FFEFFFDF, 0xDFFFBFFF7FFEFFFD, 0xFDFFFBFFF7FFEFFF, 0xFFDFFFBFFF7FFEFF, 0xFFFDFFFBFFF7FFEF, 0xFFFFDFFFBFFF7FFE

},

{

0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE, 0xFFFEFFFEFFFEFFFE

},

{

0xFFF7FFFBFFFDFFFE, 0xFF7FFFBFFFDFFFEF, 0xF7FFFBFFFDFFFEFF, 0x7FFFBFFFDFFFEFFF, 0xFFFBFFFDFFFEFFFF, 0xFFBFFFDFFFEFFFF7, 0xFBFFFDFFFEFFFF7F, 0xBFFFDFFFEFFFF7FF, 0xFFFDFFFEFFFF7FFF, 0xFFDFFFEFFFF7FFFB, 0xFDFFFEFFFF7FFFBF, 0xDFFFEFFFF7FFFBFF, 0xFFFEFFFF7FFFBFFF, 0xFFEFFFF7FFFBFFFD, 0xFEFFFF7FFFBFFFDF, 0xFFFFF7FFFBFFFDFF

},

{

0xFFBFFFEFFFFBFFFE, 0xBFFFEFFFFBFFFEFF, 0xFFEFFFFBFFFEFFFF, 0xEFFFFBFFFEFFFFBF, 0xFFFBFFFEFFFFBFFF, 0xFBFFFEFFFFBFFFEF, 0xFFFEFFFFBFFFEFFF, 0xFEFFFFBFFFEFFFFB, 0xFFFFBFFFEFFFFBFF, 0xFFBFFFEFFFFBFFFE, 0xBFFFEFFFFBFFFEFF, 0xFFEFFFFBFFFEFFFF, 0xEFFFFBFFFEFFFFBF, 0xFFFBFFFEFFFFBFFF, 0xFBFFFEFFFFBFFFEF, 0xFFFFFFFFBFFFEFFF

},

{

0xFDFFFFBFFFF7FFFE, 0xFFFBFFFF7FFFEFFF, 0xBFFFF7FFFEFFFFDF, 0xFF7FFFEFFFFDFFFF, 0xFFFEFFFFDFFFFBFF, 0xEFFFFDFFFFBFFFF7, 0xFFDFFFFBFFFF7FFF, 0xFFFFBFFFF7FFFEFF, 0xFBFFFF7FFFEFFFFD, 0xFFF7FFFEFFFFDFFF, 0x7FFFEFFFFDFFFFBF, 0xFEFFFFDFFFFBFFFF, 0xFFFDFFFFBFFFF7FF, 0xDFFFFBFFFF7FFFEF, 0xFFBFFFF7FFFEFFFF, 0xFFFFFFFFEFFFFDFF

},

{

0xEFFFFEFFFFEFFFFE, 0xFEFFFFEFFFFEFFFF, 0xFFEFFFFEFFFFEFFF, 0xFFFEFFFFEFFFFEFF, 0xFFFFEFFFFEFFFFEF, 0xEFFFFEFFFFEFFFFE, 0xFEFFFFEFFFFEFFFF, 0xFFEFFFFEFFFFEFFF, 0xFFFEFFFFEFFFFEFF, 0xFFFFEFFFFEFFFFEF, 0xEFFFFEFFFFEFFFFE, 0xFEFFFFEFFFFEFFFF, 0xFFEFFFFEFFFFEFFF, 0xFFFEFFFFEFFFFEFF, 0xFFFFEFFFFEFFFFEF, 0xFFFFFEFFFFEFFFFE

},

{

0x7FFFFBFFFFDFFFFE, 0xBFFFFDFFFFEFFFFF, 0xDFFFFEFFFFF7FFFF, 0xEFFFFF7FFFFBFFFF, 0xF7FFFFBFFFFDFFFF, 0xFBFFFFDFFFFEFFFF, 0xFDFFFFEFFFFF7FFF, 0xFEFFFFF7FFFFBFFF, 0xFF7FFFFBFFFFDFFF, 0xFFBFFFFDFFFFEFFF, 0xFFDFFFFEFFFFF7FF, 0xFFEFFFFF7FFFFBFF, 0xFFF7FFFFBFFFFDFF, 0xFFFBFFFFDFFFFEFF, 0xFFFDFFFFEFFFFF7F, 0xFFFFFFFFF7FFFFBF

},

{

0xFFFFEFFFFFBFFFFE, 0xFFFFBFFFFEFFFFFB, 0xFFFEFFFFFBFFFFEF, 0xFFFBFFFFEFFFFFBF, 0xFFEFFFFFBFFFFEFF, 0xFFBFFFFEFFFFFBFF, 0xFEFFFFFBFFFFEFFF, 0xFBFFFFEFFFFFBFFF, 0xEFFFFFBFFFFEFFFF, 0xBFFFFEFFFFFBFFFF, 0xFFFFFBFFFFEFFFFF, 0xFFFFEFFFFFBFFFFE, 0xFFFFBFFFFEFFFFFB, 0xFFFEFFFFFBFFFFEF, 0xFFFBFFFFEFFFFFBF, 0xFFFFFFFFBFFFFEFF

},

{

0xFFFFBFFFFF7FFFFE, 0xFFF7FFFFEFFFFFDF, 0xFEFFFFFDFFFFFBFF, 0xDFFFFFBFFFFF7FFF, 0xFFFFF7FFFFEFFFFF, 0xFFFEFFFFFDFFFFFB, 0xFFDFFFFFBFFFFF7F, 0xFBFFFFF7FFFFEFFF, 0x7FFFFEFFFFFDFFFF, 0xFFFFDFFFFFBFFFFF, 0xFFFBFFFFF7FFFFEF, 0xFF7FFFFEFFFFFDFF, 0xEFFFFFDFFFFFBFFF, 0xFFFFFBFFFFF7FFFF, 0xFFFF7FFFFEFFFFFD, 0xFFFFFFFFDFFFFFBF

},

{

0xFFFEFFFFFEFFFFFE, 0xFEFFFFFEFFFFFEFF, 0xFFFFFEFFFFFEFFFF, 0xFFFEFFFFFEFFFFFE, 0xFEFFFFFEFFFFFEFF, 0xFFFFFEFFFFFEFFFF, 0xFFFEFFFFFEFFFFFE, 0xFEFFFFFEFFFFFEFF, 0xFFFFFEFFFFFEFFFF, 0xFFFEFFFFFEFFFFFE, 0xFEFFFFFEFFFFFEFF, 0xFFFFFEFFFFFEFFFF, 0xFFFEFFFFFEFFFFFE, 0xFEFFFFFEFFFFFEFF, 0xFFFFFEFFFFFEFFFF, 0xFFFFFFFFFEFFFFFE

},

{

0xFFFBFFFFFDFFFFFE, 0xDFFFFFEFFFFFF7FF, 0xFFFF7FFFFFBFFFFF, 0xFBFFFFFDFFFFFEFF, 0xFFFFEFFFFFF7FFFF, 0xFF7FFFFFBFFFFFDF, 0xFFFFFDFFFFFEFFFF, 0xFFEFFFFFF7FFFFFB, 0x7FFFFFBFFFFFDFFF, 0xFFFDFFFFFEFFFFFF, 0xEFFFFFF7FFFFFBFF, 0xFFFFBFFFFFDFFFFF, 0xFDFFFFFEFFFFFF7F, 0xFFFFF7FFFFFBFFFF, 0xFFBFFFFFDFFFFFEF, 0xFFFFFFFFFFFF7FFF

},

{

0xFFEFFFFFFBFFFFFE, 0xFFFFFEFFFFFFBFFF, 0xFFBFFFFFEFFFFFFB, 0xFFFFFBFFFFFEFFFF, 0xFEFFFFFFBFFFFFEF, 0xFFFFEFFFFFFBFFFF, 0xFBFFFFFEFFFFFFBF, 0xFFFFBFFFFFEFFFFF, 0xEFFFFFFBFFFFFEFF, 0xFFFEFFFFFFBFFFFF, 0xBFFFFFEFFFFFFBFF, 0xFFFBFFFFFEFFFFFF, 0xFFFFFFBFFFFFEFFF, 0xFFEFFFFFFBFFFFFE, 0xFFFFFEFFFFFFBFFF, 0xFFFFFFFFEFFFFFFB

},

{

0xFFBFFFFFF7FFFFFE, 0xFFFFEFFFFFFDFFFF, 0xDFFFFFFBFFFFFF7F, 0xFFF7FFFFFEFFFFFF, 0xFFFFFDFFFFFFBFFF, 0xFBFFFFFF7FFFFFEF, 0xFFFEFFFFFFDFFFFF, 0xFFFFFFBFFFFFF7FF, 0xFF7FFFFFEFFFFFFD, 0xFFFFDFFFFFFBFFFF, 0xBFFFFFF7FFFFFEFF, 0xFFEFFFFFFDFFFFFF, 0xFFFFFBFFFFFF7FFF, 0xF7FFFFFEFFFFFFDF, 0xFFFDFFFFFFBFFFFF, 0xFFFFFFFFFFFFEFFF

},

{

0xFEFFFFFFEFFFFFFE, 0xFFFEFFFFFFEFFFFF, 0xFFFFFEFFFFFFEFFF, 0xEFFFFFFEFFFFFFEF, 0xFFEFFFFFFEFFFFFF, 0xFFFFEFFFFFFEFFFF, 0xFFFFFFEFFFFFFEFF, 0xFEFFFFFFEFFFFFFE, 0xFFFEFFFFFFEFFFFF, 0xFFFFFEFFFFFFEFFF, 0xEFFFFFFEFFFFFFEF, 0xFFEFFFFFFEFFFFFF, 0xFFFFEFFFFFFEFFFF, 0xFFFFFFEFFFFFFEFF, 0xFEFFFFFFEFFFFFFE, 0xFFFFFFFFFFEFFFFF

},

{

0xFBFFFFFFDFFFFFFE, 0xFFEFFFFFFF7FFFFF, 0xFFFFBFFFFFFDFFFF, 0xFFFFFEFFFFFFF7FF, 0x7FFFFFFBFFFFFFDF, 0xFDFFFFFFEFFFFFFF, 0xFFF7FFFFFFBFFFFF, 0xFFFFDFFFFFFEFFFF, 0xFFFFFF7FFFFFFBFF, 0xBFFFFFFDFFFFFFEF, 0xFEFFFFFFF7FFFFFF, 0xFFFBFFFFFFDFFFFF, 0xFFFFEFFFFFFF7FFF, 0xFFFFFFBFFFFFFDFF, 0xDFFFFFFEFFFFFFF7, 0xFFFFFFFFFBFFFFFF

},

{

0xEFFFFFFFBFFFFFFE, 0xFEFFFFFFFBFFFFFF, 0xFFEFFFFFFFBFFFFF, 0xFFFEFFFFFFFBFFFF, 0xFFFFEFFFFFFFBFFF, 0xFFFFFEFFFFFFFBFF, 0xFFFFFFEFFFFFFFBF, 0xBFFFFFFEFFFFFFFB, 0xFBFFFFFFEFFFFFFF, 0xFFBFFFFFFEFFFFFF, 0xFFFBFFFFFFEFFFFF, 0xFFFFBFFFFFFEFFFF, 0xFFFFFBFFFFFFEFFF, 0xFFFFFFBFFFFFFEFF, 0xFFFFFFFBFFFFFFEF, 0xFFFFFFFFBFFFFFFE

},

{

0xBFFFFFFF7FFFFFFE, 0xEFFFFFFFDFFFFFFF, 0xFBFFFFFFF7FFFFFF, 0xFEFFFFFFFDFFFFFF, 0xFFBFFFFFFF7FFFFF, 0xFFEFFFFFFFDFFFFF, 0xFFFBFFFFFFF7FFFF, 0xFFFEFFFFFFFDFFFF, 0xFFFFBFFFFFFF7FFF, 0xFFFFEFFFFFFFDFFF, 0xFFFFFBFFFFFFF7FF, 0xFFFFFEFFFFFFFDFF, 0xFFFFFFBFFFFFFF7F, 0xFFFFFFEFFFFFFFDF, 0xFFFFFFFBFFFFFFF7, 0xFFFFFFFEFFFFFFFD

},

{

0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE, 0xFFFFFFFEFFFFFFFE

},

{

0xFFFFFFFDFFFFFFFE, 0xFFFFFFF7FFFFFFFB, 0xFFFFFFDFFFFFFFEF, 0xFFFFFF7FFFFFFFBF, 0xFFFFFDFFFFFFFEFF, 0xFFFFF7FFFFFFFBFF, 0xFFFFDFFFFFFFEFFF, 0xFFFF7FFFFFFFBFFF, 0xFFFDFFFFFFFEFFFF, 0xFFF7FFFFFFFBFFFF, 0xFFDFFFFFFFEFFFFF, 0xFF7FFFFFFFBFFFFF, 0xFDFFFFFFFEFFFFFF, 0xF7FFFFFFFBFFFFFF, 0xDFFFFFFFEFFFFFFF, 0xFFFFFFFFBFFFFFFF

},

{

0xFFFFFFFBFFFFFFFE, 0xFFFFFFBFFFFFFFEF, 0xFFFFFBFFFFFFFEFF, 0xFFFFBFFFFFFFEFFF, 0xFFFBFFFFFFFEFFFF, 0xFFBFFFFFFFEFFFFF, 0xFBFFFFFFFEFFFFFF, 0xBFFFFFFFEFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFEFFFFFFFFB, 0xFFFFFEFFFFFFFFBF, 0xFFFFEFFFFFFFFBFF, 0xFFFEFFFFFFFFBFFF, 0xFFEFFFFFFFFBFFFF, 0xFEFFFFFFFFBFFFFF, 0xFFFFFFFFFBFFFFFF

},

{

0xFFFFFFF7FFFFFFFE, 0xFFFFFDFFFFFFFFBF, 0xFFFF7FFFFFFFEFFF, 0xFFDFFFFFFFFBFFFF, 0xF7FFFFFFFEFFFFFF, 0xFFFFFFFFBFFFFFFF, 0xFFFFFFEFFFFFFFFD, 0xFFFFFBFFFFFFFF7F, 0xFFFEFFFFFFFFDFFF, 0xFFBFFFFFFFF7FFFF, 0xEFFFFFFFFDFFFFFF, 0xFFFFFFFF7FFFFFFF, 0xFFFFFFDFFFFFFFFB, 0xFFFFF7FFFFFFFEFF, 0xFFFDFFFFFFFFBFFF, 0xFFFFFFFFFFEFFFFF

},

{

0xFFFFFFEFFFFFFFFE, 0xFFFFEFFFFFFFFEFF, 0xFFEFFFFFFFFEFFFF, 0xEFFFFFFFFEFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFEFFFFFFFFEF, 0xFFFEFFFFFFFFEFFF, 0xFEFFFFFFFFEFFFFF, 0xFFFFFFFFEFFFFFFF, 0xFFFFFFEFFFFFFFFE, 0xFFFFEFFFFFFFFEFF, 0xFFEFFFFFFFFEFFFF, 0xEFFFFFFFFEFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFEFFFFFFFFEF, 0xFFFFFFFFFFFFEFFF

},

{

0xFFFFFFDFFFFFFFFE, 0xFFFF7FFFFFFFFBFF, 0xFDFFFFFFFFEFFFFF, 0xFFFFFFFFBFFFFFFF, 0xFFFFFEFFFFFFFFF7, 0xFFFBFFFFFFFFDFFF, 0xEFFFFFFFFF7FFFFF, 0xFFFFFFFDFFFFFFFF, 0xFFFFF7FFFFFFFFBF, 0xFFDFFFFFFFFEFFFF, 0x7FFFFFFFFBFFFFFF, 0xFFFFFFEFFFFFFFFF, 0xFFFFBFFFFFFFFDFF, 0xFEFFFFFFFFF7FFFF, 0xFFFFFFFFDFFFFFFF, 0xFFFFFFFFFFFFFFFB

},

{

0xFFFFFFBFFFFFFFFE, 0xFFFBFFFFFFFFEFFF, 0xBFFFFFFFFEFFFFFF, 0xFFFFFFEFFFFFFFFF, 0xFFFEFFFFFFFFFBFF, 0xEFFFFFFFFFBFFFFF, 0xFFFFFFFBFFFFFFFF, 0xFFFFBFFFFFFFFEFF, 0xFBFFFFFFFFEFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFEFFFFFFFFFBF, 0xFEFFFFFFFFFBFFFF, 0xFFFFFFFFBFFFFFFF, 0xFFFFFBFFFFFFFFEF, 0xFFBFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFF7FFFFFFFFE, 0xFFDFFFFFFFFFBFFF, 0xFFFFFFFFEFFFFFFF, 0xFFFFFBFFFFFFFFF7, 0xFEFFFFFFFFFDFFFF, 0xFFFFFFFF7FFFFFFF, 0xFFFFDFFFFFFFFFBF, 0xF7FFFFFFFFEFFFFF, 0xFFFFFFFBFFFFFFFF, 0xFFFEFFFFFFFFFDFF, 0xBFFFFFFFFF7FFFFF, 0xFFFFFFDFFFFFFFFF, 0xFFF7FFFFFFFFEFFF, 0xFFFFFFFFFBFFFFFF, 0xFFFFFEFFFFFFFFFD, 0xFFFFFFFFFFFF7FFF

},

{

0xFFFFFEFFFFFFFFFE, 0xFEFFFFFFFFFEFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFEFFFFFFFFFEFF, 0xFFFFFFFFFEFFFFFF, 0xFFFFFEFFFFFFFFFE, 0xFEFFFFFFFFFEFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFEFFFFFFFFFEFF, 0xFFFFFFFFFEFFFFFF, 0xFFFFFEFFFFFFFFFE, 0xFEFFFFFFFFFEFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFEFFFFFFFFFEFF, 0xFFFFFFFFFEFFFFFF, 0xFFFFFFFFFFFFFFFE

},

{

0xFFFFFDFFFFFFFFFE, 0xF7FFFFFFFFFBFFFF, 0xFFFFFFEFFFFFFFFF, 0xFFBFFFFFFFFFDFFF, 0xFFFFFFFF7FFFFFFF, 0xFFFDFFFFFFFFFEFF, 0xFFFFFFFFFBFFFFFF, 0xFFFFEFFFFFFFFFF7, 0xBFFFFFFFFFDFFFFF, 0xFFFFFF7FFFFFFFFF, 0xFDFFFFFFFFFEFFFF, 0xFFFFFFFBFFFFFFFF, 0xFFEFFFFFFFFFF7FF, 0xFFFFFFFFDFFFFFFF, 0xFFFF7FFFFFFFFFBF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFBFFFFFFFFFE, 0xBFFFFFFFFFEFFFFF, 0xFFFFFEFFFFFFFFFF, 0xEFFFFFFFFFFBFFFF, 0xFFFFFFBFFFFFFFFF, 0xFBFFFFFFFFFEFFFF, 0xFFFFFFEFFFFFFFFF, 0xFEFFFFFFFFFFBFFF, 0xFFFFFFFBFFFFFFFF, 0xFFBFFFFFFFFFEFFF, 0xFFFFFFFEFFFFFFFF, 0xFFEFFFFFFFFFFBFF, 0xFFFFFFFFBFFFFFFF, 0xFFFBFFFFFFFFFEFF, 0xFFFFFFFFEFFFFFFF, 0xFFFFFFFFFFFFFFBF

},

{

0xFFFFF7FFFFFFFFFE, 0xFFFFFFFFFFBFFFFF, 0xFFFFEFFFFFFFFFFD, 0xFFFFFFFFFF7FFFFF, 0xFFFFDFFFFFFFFFFB, 0xFFFFFFFFFEFFFFFF, 0xFFFFBFFFFFFFFFF7, 0xFFFFFFFFFDFFFFFF, 0xFFFF7FFFFFFFFFEF, 0xFFFFFFFFFBFFFFFF, 0xFFFEFFFFFFFFFFDF, 0xFFFFFFFFF7FFFFFF, 0xFFFDFFFFFFFFFFBF, 0xFFFFFFFFEFFFFFFF, 0xFFFBFFFFFFFFFF7F, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFEFFFFFFFFFFE, 0xFFFFFFFFFEFFFFFF, 0xFFFEFFFFFFFFFFEF, 0xFFFFFFFFEFFFFFFF, 0xFFEFFFFFFFFFFEFF, 0xFFFFFFFEFFFFFFFF, 0xFEFFFFFFFFFFEFFF, 0xFFFFFFEFFFFFFFFF, 0xEFFFFFFFFFFEFFFF, 0xFFFFFEFFFFFFFFFF, 0xFFFFFFFFFFEFFFFF, 0xFFFFEFFFFFFFFFFE, 0xFFFFFFFFFEFFFFFF, 0xFFFEFFFFFFFFFFEF, 0xFFFFFFFFEFFFFFFF, 0xFFFFFFFFFFFFFEFF

},

{

0xFFFFDFFFFFFFFFFE, 0xFFFFFFFFFBFFFFFF, 0xFFEFFFFFFFFFFF7F, 0xFFFFFFFDFFFFFFFF, 0xF7FFFFFFFFFFBFFF, 0xFFFFFEFFFFFFFFFF, 0xFFFFFFFFFFDFFFFF, 0xFFFF7FFFFFFFFFFB, 0xFFFFFFFFEFFFFFFF, 0xFFBFFFFFFFFFFDFF, 0xFFFFFFF7FFFFFFFF, 0xDFFFFFFFFFFEFFFF, 0xFFFFFBFFFFFFFFFF, 0xFFFFFFFFFF7FFFFF, 0xFFFDFFFFFFFFFFEF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFBFFFFFFFFFFE, 0xFFFFFFFFEFFFFFFF, 0xFEFFFFFFFFFFFBFF, 0xFFFFFFBFFFFFFFFF, 0xFFFFFFFFFFEFFFFF, 0xFFFEFFFFFFFFFFFB, 0xFFFFFFFFBFFFFFFF, 0xFBFFFFFFFFFFEFFF, 0xFFFFFEFFFFFFFFFF, 0xFFFFFFFFFFBFFFFF, 0xFFFBFFFFFFFFFFEF, 0xFFFFFFFEFFFFFFFF, 0xEFFFFFFFFFFFBFFF, 0xFFFFFBFFFFFFFFFF, 0xFFFFFFFFFEFFFFFF, 0xFFFFFFFFFFFFFFBF

},

{

0xFFFF7FFFFFFFFFFE, 0xFFFFFFFFBFFFFFFF, 0xEFFFFFFFFFFFDFFF, 0xFFFFF7FFFFFFFFFF, 0xFFFFFFFFFBFFFFFF, 0xFEFFFFFFFFFFFDFF, 0xFFFFFF7FFFFFFFFF, 0xFFFFFFFFFFBFFFFF, 0xFFEFFFFFFFFFFFDF, 0xFFFFFFF7FFFFFFFF, 0xFFFFFFFFFFFBFFFF, 0xFFFEFFFFFFFFFFFD, 0xFFFFFFFF7FFFFFFF, 0xDFFFFFFFFFFFBFFF, 0xFFFFEFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFEFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFEFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFEFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFEFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFEFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFE

}

};

// The following is used to construct the InvalidBitsTable statically without forcing BVStatic to be an aggregate

const HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::InvalidBitsTable \* const HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::invalidBitsBuffers =

reinterpret\_cast<const HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::InvalidBitsTable \*>(&HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::invalidBitsData);

const SmallHeapBlockT<SmallAllocationBlockAttributes>::BlockInfo HeapInfo::ValidPointersMap<SmallAllocationBlockAttributes>::blockInfoBuffer[SmallAllocationBlockAttributes::BucketCount][SmallAllocationBlockAttributes::PageCount] = {

// Bucket: 0, Size: 16

{

{ 0x00FF, 0x0100 },

{ 0x01FF, 0x0100 },

{ 0x02FF, 0x0100 },

{ 0x03FF, 0x0100 }

},

// Bucket: 1, Size: 32

{

{ 0x007F, 0x0080 },

{ 0x00FF, 0x0080 },

{ 0x017F, 0x0080 },

{ 0x01FF, 0x0080 }

},

// Bucket: 2, Size: 48

{

{ 0x0055, 0x0056 },

{ 0x00AA, 0x0055 },

{ 0x00FF, 0x0055 },

{ 0x0154, 0x0055 }

},

// Bucket: 3, Size: 64

{

{ 0x003F, 0x0040 },

{ 0x007F, 0x0040 },

{ 0x00BF, 0x0040 },

{ 0x00FF, 0x0040 }

},

// Bucket: 4, Size: 80

{

{ 0x0033, 0x0034 },

{ 0x0066, 0x0033 },

{ 0x0099, 0x0033 },

{ 0x00CB, 0x0032 }

},

// Bucket: 5, Size: 96

{

{ 0x002A, 0x002B },

{ 0x0055, 0x002B },

{ 0x007F, 0x002A },

{ 0x00A9, 0x002A }

},

// Bucket: 6, Size: 112

{

{ 0x0024, 0x0025 },

{ 0x0049, 0x0025 },

{ 0x006D, 0x0024 },

{ 0x0091, 0x0024 }

},

// Bucket: 7, Size: 128

{

{ 0x001F, 0x0020 },

{ 0x003F, 0x0020 },

{ 0x005F, 0x0020 },

{ 0x007F, 0x0020 }

},

// Bucket: 8, Size: 144

{

{ 0x001C, 0x001D },

{ 0x0038, 0x001C },

{ 0x0055, 0x001D },

{ 0x0070, 0x001B }

},

// Bucket: 9, Size: 160

{

{ 0x0019, 0x001A },

{ 0x0033, 0x001A },

{ 0x004C, 0x0019 },

{ 0x0065, 0x0019 }

},

// Bucket: 10, Size: 176

{

{ 0x0017, 0x0018 },

{ 0x002E, 0x0017 },

{ 0x0045, 0x0017 },

{ 0x005C, 0x0017 }

},

// Bucket: 11, Size: 192

{

{ 0x0015, 0x0016 },

{ 0x002A, 0x0015 },

{ 0x003F, 0x0015 },

{ 0x0054, 0x0015 }

},

// Bucket: 12, Size: 208

{

{ 0x0013, 0x0014 },

{ 0x0027, 0x0014 },

{ 0x003B, 0x0014 },

{ 0x004D, 0x0012 }

},

// Bucket: 13, Size: 224

{

{ 0x0012, 0x0013 },

{ 0x0024, 0x0012 },

{ 0x0036, 0x0012 },

{ 0x0048, 0x0012 }

},

// Bucket: 14, Size: 240

{

{ 0x0011, 0x0012 },

{ 0x0022, 0x0011 },

{ 0x0033, 0x0011 },

{ 0x0043, 0x0010 }

},

// Bucket: 15, Size: 256

{

{ 0x000F, 0x0010 },

{ 0x001F, 0x0010 },

{ 0x002F, 0x0010 },

{ 0x003F, 0x0010 }

},

// Bucket: 16, Size: 272

{

{ 0x000F, 0x0010 },

{ 0x001E, 0x000F },

{ 0x002D, 0x000F },

{ 0x003B, 0x000E }

},

// Bucket: 17, Size: 288

{

{ 0x000E, 0x000F },

{ 0x001C, 0x000E },

{ 0x002A, 0x000E },

{ 0x0037, 0x000D }

},

// Bucket: 18, Size: 304

{

{ 0x000D, 0x000E },

{ 0x001A, 0x000D },

{ 0x0028, 0x000E },

{ 0x0034, 0x000C }

},

// Bucket: 19, Size: 320

{

{ 0x000C, 0x000D },

{ 0x0019, 0x000D },

{ 0x0026, 0x000D },

{ 0x0032, 0x000C }

},

// Bucket: 20, Size: 336

{

{ 0x000C, 0x000D },

{ 0x0018, 0x000C },

{ 0x0024, 0x000C },

{ 0x002F, 0x000B }

},

// Bucket: 21, Size: 352

{

{ 0x000B, 0x000C },

{ 0x0017, 0x000C },

{ 0x0022, 0x000B },

{ 0x002D, 0x000B }

},

// Bucket: 22, Size: 368

{

{ 0x000B, 0x000C },

{ 0x0016, 0x000B },

{ 0x0021, 0x000B },

{ 0x002B, 0x000A }

},

// Bucket: 23, Size: 384

{

{ 0x000A, 0x000B },

{ 0x0015, 0x000B },

{ 0x001F, 0x000A },

{ 0x0029, 0x000A }

},

// Bucket: 24, Size: 400

{

{ 0x000A, 0x000B },

{ 0x0014, 0x000A },

{ 0x001E, 0x000A },

{ 0x0027, 0x0009 }

},

// Bucket: 25, Size: 416

{

{ 0x0009, 0x000A },

{ 0x0013, 0x000A },

{ 0x001D, 0x000A },

{ 0x0026, 0x0009 }

},

// Bucket: 26, Size: 432

{

{ 0x0009, 0x000A },

{ 0x0012, 0x0009 },

{ 0x001C, 0x000A },

{ 0x0024, 0x0008 }

},

// Bucket: 27, Size: 448

{

{ 0x0009, 0x000A },

{ 0x0012, 0x0009 },

{ 0x001B, 0x0009 },

{ 0x0023, 0x0008 }

},

// Bucket: 28, Size: 464

{

{ 0x0008, 0x0009 },

{ 0x0011, 0x0009 },

{ 0x001A, 0x0009 },

{ 0x0022, 0x0008 }

},

// Bucket: 29, Size: 480

{

{ 0x0008, 0x0009 },

{ 0x0011, 0x0009 },

{ 0x0019, 0x0008 },

{ 0x0021, 0x0008 }

},

// Bucket: 30, Size: 496

{

{ 0x0008, 0x0009 },

{ 0x0010, 0x0008 },

{ 0x0018, 0x0008 },

{ 0x0020, 0x0008 }

},

// Bucket: 31, Size: 512

{

{ 0x0007, 0x0008 },

{ 0x000F, 0x0008 },

{ 0x0017, 0x0008 },

{ 0x001F, 0x0008 }

},

// Bucket: 32, Size: 528

{

{ 0x0007, 0x0008 },

{ 0x000F, 0x0008 },

{ 0x0017, 0x0008 },

{ 0x001E, 0x0007 }

},

// Bucket: 33, Size: 544

{

{ 0x0007, 0x0008 },

{ 0x000F, 0x0008 },

{ 0x0016, 0x0007 },

{ 0x001D, 0x0007 }

},

// Bucket: 34, Size: 560

{

{ 0x0007, 0x0008 },

{ 0x000E, 0x0007 },

{ 0x0015, 0x0007 },

{ 0x001C, 0x0007 }

},

// Bucket: 35, Size: 576

{

{ 0x0007, 0x0008 },

{ 0x000E, 0x0007 },

{ 0x0015, 0x0007 },

{ 0x001B, 0x0006 }

},

// Bucket: 36, Size: 592

{

{ 0x0006, 0x0007 },

{ 0x000D, 0x0007 },

{ 0x0014, 0x0007 },

{ 0x001A, 0x0006 }

},

// Bucket: 37, Size: 608

{

{ 0x0006, 0x0007 },

{ 0x000D, 0x0007 },

{ 0x0014, 0x0007 },

{ 0x0019, 0x0005 }

},

// Bucket: 38, Size: 624

{

{ 0x0006, 0x0007 },

{ 0x000D, 0x0007 },

{ 0x0013, 0x0006 },

{ 0x0019, 0x0006 }

},

// Bucket: 39, Size: 640

{

{ 0x0006, 0x0007 },

{ 0x000C, 0x0006 },

{ 0x0013, 0x0007 },

{ 0x0018, 0x0005 }

},

// Bucket: 40, Size: 656

{

{ 0x0006, 0x0007 },

{ 0x000C, 0x0006 },

{ 0x0012, 0x0006 },

{ 0x0017, 0x0005 }

},

// Bucket: 41, Size: 672

{

{ 0x0006, 0x0007 },

{ 0x000C, 0x0006 },

{ 0x0012, 0x0006 },

{ 0x0017, 0x0005 }

},

// Bucket: 42, Size: 688

{

{ 0x0005, 0x0006 },

{ 0x000B, 0x0006 },

{ 0x0011, 0x0006 },

{ 0x0016, 0x0005 }

},

// Bucket: 43, Size: 704

{

{ 0x0005, 0x0006 },

{ 0x000B, 0x0006 },

{ 0x0011, 0x0006 },

{ 0x0016, 0x0005 }

},

// Bucket: 44, Size: 720

{

{ 0x0005, 0x0006 },

{ 0x000B, 0x0006 },

{ 0x0011, 0x0006 },

{ 0x0015, 0x0004 }

},

// Bucket: 45, Size: 736

{

{ 0x0005, 0x0006 },

{ 0x000B, 0x0006 },

{ 0x0010, 0x0005 },

{ 0x0015, 0x0005 }

},

// Bucket: 46, Size: 752

{

{ 0x0005, 0x0006 },

{ 0x000A, 0x0005 },

{ 0x0010, 0x0006 },

{ 0x0014, 0x0004 }

},

// Bucket: 47, Size: 768

{

{ 0x0005, 0x0006 },

{ 0x000A, 0x0005 },

{ 0x000F, 0x0005 },

{ 0x0014, 0x0005 }

}

};

const ushort HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::validPointersBuffer[MediumAllocationBlockAttributes::BucketCount][HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::rowSize] =

{

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0019, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x001F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x0019, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001A, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001B, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001C, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001D, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001E, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F, 0x001F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0015, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0016, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0017, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0015, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0016, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0017, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0x0018, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0012, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0013, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0012, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0013, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0x0014, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0010, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0010, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0x0011, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000E, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000F, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000E, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F, 0x000F

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000C, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000C, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0x000D, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0x000B, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0x000A, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0x0009, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0x0008, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0007, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007, 0x0007

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0x0006, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0x0005, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0x0004, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF

},

{

0x0000, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0001, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0002, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0003, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003

}

};

const BVUnit HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::invalidBitsData[MediumAllocationBlockAttributes::BucketCount][SmallHeapBlockT<MediumAllocationBlockAttributes>::SmallHeapBlockBitVector::wordCount] = {

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFE

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFEFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFEFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFEFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

},

{

0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFE, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF, 0xFFFFFFFFFFFFFFFF

}

};

// The following is used to construct the InvalidBitsTable statically without forcing BVStatic to be an aggregate

const HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::InvalidBitsTable \* const HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::invalidBitsBuffers =

reinterpret\_cast<const HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::InvalidBitsTable \*>(&HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::invalidBitsData);

const SmallHeapBlockT<MediumAllocationBlockAttributes>::BlockInfo HeapInfo::ValidPointersMap<MediumAllocationBlockAttributes>::blockInfoBuffer[MediumAllocationBlockAttributes::BucketCount][MediumAllocationBlockAttributes::PageCount] = {

// Bucket: 0, Size: 1024

{

{ 0x0003, 0x0004 },

{ 0x0007, 0x0004 },

{ 0x000B, 0x0004 },

{ 0x000F, 0x0004 },

{ 0x0013, 0x0004 },

{ 0x0017, 0x0004 },

{ 0x001B, 0x0004 },

{ 0x001F, 0x0004 }

},

// Bucket: 1, Size: 1280

{

{ 0x0003, 0x0004 },

{ 0x0006, 0x0003 },

{ 0x0009, 0x0003 },

{ 0x000C, 0x0003 },

{ 0x000F, 0x0003 },

{ 0x0013, 0x0004 },

{ 0x0016, 0x0003 },

{ 0x0018, 0x0002 }

},

// Bucket: 2, Size: 1536

{

{ 0x0002, 0x0003 },

{ 0x0005, 0x0003 },

{ 0x0007, 0x0002 },

{ 0x000A, 0x0003 },

{ 0x000D, 0x0003 },

{ 0x000F, 0x0002 },

{ 0x0012, 0x0003 },

{ 0x0014, 0x0002 }

},

// Bucket: 3, Size: 1792

{

{ 0x0002, 0x0003 },

{ 0x0004, 0x0002 },

{ 0x0006, 0x0002 },

{ 0x0009, 0x0003 },

{ 0x000B, 0x0002 },

{ 0x000D, 0x0002 },

{ 0x000F, 0x0002 },

{ 0x0011, 0x0002 }

},

// Bucket: 4, Size: 2048

{

{ 0x0001, 0x0002 },

{ 0x0003, 0x0002 },

{ 0x0005, 0x0002 },

{ 0x0007, 0x0002 },

{ 0x0009, 0x0002 },

{ 0x000B, 0x0002 },

{ 0x000D, 0x0002 },

{ 0x000F, 0x0002 }

},

// Bucket: 5, Size: 2304

{

{ 0x0001, 0x0002 },

{ 0x0003, 0x0002 },

{ 0x0005, 0x0002 },

{ 0x0007, 0x0002 },

{ 0x0008, 0x0001 },

{ 0x000A, 0x0002 },

{ 0x000C, 0x0002 },

{ 0x000D, 0x0001 }

},

// Bucket: 6, Size: 2560

{

{ 0x0001, 0x0002 },

{ 0x0003, 0x0002 },

{ 0x0004, 0x0001 },

{ 0x0006, 0x0002 },

{ 0x0007, 0x0001 },

{ 0x0009, 0x0002 },

{ 0x000B, 0x0002 },

{ 0x0000, 0x0000 }

},

// Bucket: 7, Size: 2816

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0004, 0x0002 },

{ 0x0005, 0x0001 },

{ 0x0007, 0x0002 },

{ 0x0008, 0x0001 },

{ 0x000A, 0x0002 },

{ 0x0000, 0x0000 }

},

// Bucket: 8, Size: 3072

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0005, 0x0002 },

{ 0x0006, 0x0001 },

{ 0x0007, 0x0001 },

{ 0x0009, 0x0002 },

{ 0x0000, 0x0000 }

},

// Bucket: 9, Size: 3328

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0006, 0x0002 },

{ 0x0007, 0x0001 },

{ 0x0008, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 10, Size: 3584

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0007, 0x0001 },

{ 0x0008, 0x0001 }

},

// Bucket: 11, Size: 3840

{

{ 0x0001, 0x0002 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0007, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 12, Size: 4096

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0007, 0x0001 }

},

// Bucket: 13, Size: 4352

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 14, Size: 4608

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0006, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 15, Size: 4864

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 16, Size: 5120

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 17, Size: 5376

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0004, 0x0001 },

{ 0x0005, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 18, Size: 5632

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 19, Size: 5888

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0004, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 20, Size: 6144

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0004, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 21, Size: 6400

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0004, 0x0001 },

{ 0x0000, 0x0000 }

},

// Bucket: 22, Size: 6656

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 23, Size: 6912

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 24, Size: 7168

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 25, Size: 7424

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 26, Size: 7680

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 27, Size: 7936

{

{ 0x0000, 0x0001 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0000, 0x0000 }

},

// Bucket: 28, Size: 8192

{

{ 0x0000, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0001, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0002, 0x0001 },

{ 0x0000, 0x0000 },

{ 0x0003, 0x0001 },

{ 0x0000, 0x0000 }

}

};

#endif // USE\_STATIC\_VPM