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

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

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

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

#include "RuntimeByteCodePch.h"

#if DBG\_DUMP

#if DBG

// Parser Includes

#include "RegexCommon.h"

#include "DebugWriter.h"

#include "RegexPattern.h"

#endif

namespace Js

{

// Pre-order recursive dump, head first, then children.

void ByteCodeDumper::DumpRecursively(FunctionBody\* dumpFunction)

{

dumpFunction->EnsureDeserialized();

ByteCodeDumper::Dump(dumpFunction);

for (uint i = 0; i < dumpFunction->GetNestedCount(); i ++)

{

dumpFunction->GetNestedFunc(i)->EnsureDeserialized();

ByteCodeDumper::DumpRecursively(dumpFunction->GetNestedFunc(i)->GetFunctionBody());

}

}

void ByteCodeDumper::Dump(FunctionBody\* dumpFunction)

{

if (!CONFIG\_FLAG(DumpDbgControllerBytecode) && dumpFunction->GetSourceContextInfo() &&

dumpFunction->GetSourceContextInfo()->url != nullptr &&

\_wcsicmp(dumpFunction->GetSourceContextInfo()->url, L"dbgcontroller.js") == 0)

{

return;

}

ByteCodeReader reader;

reader.Create(dumpFunction);

StatementReader statementReader;

statementReader.Create(dumpFunction);

dumpFunction->DumpFullFunctionName();

Output::Print(L" (");

ArgSlot inParamCount = dumpFunction->GetInParamsCount();

for (ArgSlot paramIndex = 0; paramIndex < inParamCount; paramIndex++)

{

if (paramIndex > 0)

{

Output::Print(L", ");

}

Output::Print(L"In%hu", paramIndex);

}

Output::Print(L") ");

Output::Print(L"(size: %d [%d])\n", dumpFunction->GetByteCodeCount(), dumpFunction->GetByteCodeWithoutLDACount());

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

if (dumpFunction->IsByteCodeDebugMode())

{

Output::Print(L"[Bytecode was generated for debug mode]\n");

}

#endif

#if DBG

if (dumpFunction->IsReparsed())

{

Output::Print(L"[A reparse is being done]\n");

}

#endif

Output::Print(

L" %u locals (%u temps from R%u), %u inline cache\n",

dumpFunction->GetLocalsCount(),

dumpFunction->GetTempCount(),

dumpFunction->GetFirstTmpReg(),

dumpFunction->GetInlineCacheCount());

uint32 statementIndex = 0;

ByteCodeDumper::DumpConstantTable(dumpFunction);

ByteCodeDumper::DumpImplicitArgIns(dumpFunction);

while (true)

{

while (statementReader.AtStatementBoundary(&reader))

{

dumpFunction->PrintStatementSourceLine(statementIndex);

statementIndex = statementReader.MoveNextStatementBoundary();

}

uint byteOffset = reader.GetCurrentOffset();

LayoutSize layoutSize;

OpCode op = reader.ReadOp(layoutSize);

if (op == OpCode::EndOfBlock)

{

Assert(reader.GetCurrentOffset() == dumpFunction->GetByteCode()->GetLength());

break;

}

Output::Print(L" %04x %2s", byteOffset, layoutSize == LargeLayout? L"L-" : layoutSize == MediumLayout? L"M-" : L"");

DumpOp(op, layoutSize, reader, dumpFunction);

if (Js::Configuration::Global.flags.Verbose)

{

int layoutStart = byteOffset + 2; // Account fo the prefix op

int endByteOffset = reader.GetCurrentOffset();

Output::SkipToColumn(70);

if (layoutSize == LargeLayout)

{

Output::Print(L"%02X ",

op > Js::OpCode::MaxByteSizedOpcodes?

Js::OpCode::ExtendedLargeLayoutPrefix : Js::OpCode::LargeLayoutPrefix);

}

else if (layoutSize == MediumLayout)

{

Output::Print(L"%02X ",

op > Js::OpCode::MaxByteSizedOpcodes?

Js::OpCode::ExtendedMediumLayoutPrefix : Js::OpCode::MediumLayoutPrefix);

}

else

{

Assert(layoutSize == SmallLayout);

if (op > Js::OpCode::MaxByteSizedOpcodes)

{

Output::Print(L"%02X ", Js::OpCode::ExtendedOpcodePrefix);

}

else

{

Output::Print(L" ");

layoutStart--; // don't have a prefix

}

}

Output::Print(L"%02x", (byte)op);

for (int i = layoutStart; i < endByteOffset; i++)

{

Output::Print(L" %02x", reader.GetRawByte(i));

}

}

Output::Print(L"\n");

}

if (statementReader.AtStatementBoundary(&reader))

{

dumpFunction->PrintStatementSourceLine(statementIndex);

statementIndex = statementReader.MoveNextStatementBoundary();

}

Output::Print(L"\n");

Output::Flush();

}

void ByteCodeDumper::DumpConstantTable(FunctionBody \*dumpFunction)

{

Output::Print(L" Constant Table:\n ======== =====\n ");

uint count = dumpFunction->GetConstantCount();

for (RegSlot reg = FunctionBody::FirstRegSlot; reg < count; reg++)

{

DumpReg(reg);

Var varConst = dumpFunction->GetConstantVar(reg);

Assert(varConst != nullptr);

if (TaggedInt::Is(varConst))

{

#if ENABLE\_NATIVE\_CODEGEN

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::LdC\_A\_I4));

#else

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::Ld\_A));

#endif

DumpI4(TaggedInt::ToInt32(varConst));

}

else if (varConst == (Js::Var)&Js::NullFrameDisplay)

{

#if ENABLE\_NATIVE\_CODEGEN

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::LdNullDisplay));

#else

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::Ld\_A));

Output::Print(L" (NullDisplay)");

#endif

}

else if (varConst == (Js::Var)&Js::StrictNullFrameDisplay)

{

#if ENABLE\_NATIVE\_CODEGEN

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::LdStrictNullDisplay));

#else

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::Ld\_A));

Output::Print(L" (StrictNullDisplay)");

#endif

}

else

{

switch (JavascriptOperators::GetTypeId(varConst))

{

case Js::TypeIds\_Undefined:

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::Ld\_A));

Output::Print(L" (undefined)");

break;

case Js::TypeIds\_Null:

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::Ld\_A));

Output::Print(L" (null)");

break;

case Js::TypeIds\_Boolean:

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(

JavascriptBoolean::FromVar(varConst)->GetValue() ? OpCode::LdTrue : OpCode::LdFalse));

break;

case Js::TypeIds\_Number:

#if ENABLE\_NATIVE\_CODEGEN

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::LdC\_A\_R8));

#else

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::Ld\_A));

#endif

Output::Print(L"%G", JavascriptNumber::GetValue(varConst));

break;

case Js::TypeIds\_String:

#if ENABLE\_NATIVE\_CODEGEN

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::LdStr));

#else

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::Ld\_A));

#endif

Output::Print(L" (\"%s\")", JavascriptString::FromVar(varConst)->GetSz());

break;

case Js::TypeIds\_GlobalObject:

#if ENABLE\_NATIVE\_CODEGEN

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::LdRoot));

#else

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::Ld\_A));

#endif

break;

case Js::TypeIds\_ModuleRoot:

#if ENABLE\_NATIVE\_CODEGEN

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::LdModuleRoot));

#else

Output::Print(L"%-10s", OpCodeUtil::GetOpCodeName(OpCode::Ld\_A));

#endif

DumpI4(dumpFunction->GetModuleID());

break;

case Js::TypeIds\_ES5Array:

// ES5Array objects in the constant table are always string template callsite objects.

// If we later put other ES5Array objects in the constant table, we'll need another way

// to decide the constant type.

Output::Print(L"%-10s", L"LdStringTemplate");

Output::Print(L" (\"%s\")", dumpFunction->GetScriptContext()->GetLibrary()->GetStringTemplateCallsiteObjectKey(varConst));

break;

default:

AssertMsg(UNREACHED, "Unexpected object type in CloneConstantTable");

break;

}

}

Output::Print(L"\n ");

}

Output::Print(L"\n");

}

void ByteCodeDumper::DumpImplicitArgIns(FunctionBody \* dumpFunction)

{

if (dumpFunction->GetInParamsCount() <= 1 || !dumpFunction->GetHasImplicitArgIns())

{

return;

}

Output::Print(L" Implicit Arg Ins:\n ======== === ===\n ");

for (RegSlot reg = 1;

reg < dumpFunction->GetInParamsCount(); reg++)

{

DumpReg((RegSlot)(reg + dumpFunction->GetConstantCount() - 1));

// DisableJIT-TODO: Should this entire function be ifdefed?

#if ENABLE\_NATIVE\_CODEGEN

Output::Print(L"%-11s", OpCodeUtil::GetOpCodeName(Js::OpCode::ArgIn\_A));

#endif

Output::Print(L"In%d\n ", reg);

}

if (dumpFunction->GetHasRestParameter())

{

DumpReg(dumpFunction->GetRestParamRegSlot());

#if ENABLE\_NATIVE\_CODEGEN

Output::Print(L"%-11s", OpCodeUtil::GetOpCodeName(Js::OpCode::ArgIn\_Rest));

#endif

Output::Print(L"In%d\n ", dumpFunction->GetInParamsCount());

}

Output::Print(L"\n");

}

void ByteCodeDumper::DumpU4(uint32 value)

{

Output::Print(L" uint:%u ", value);

}

void ByteCodeDumper::DumpI4(int value)

{

Output::Print(L" int:%d ", value);

}

void ByteCodeDumper::DumpU2(ushort value)

{

Output::Print(L" ushort:%d ", value);

}

void ByteCodeDumper::DumpOffset(int byteOffset, ByteCodeReader const& reader)

{

Output::Print(L" x:%04x (%4d) ", reader.GetCurrentOffset() + byteOffset, byteOffset);

}

void ByteCodeDumper::DumpAddr(void\* addr)

{

Output::Print(L" addr:%04x ", addr);

}

void ByteCodeDumper::DumpOp(OpCode op, LayoutSize layoutSize, ByteCodeReader& reader, FunctionBody\* dumpFunction)

{

Output::Print(L"%-20s", OpCodeUtil::GetOpCodeName(op));

OpLayoutType nType = OpCodeUtil::GetOpCodeLayout(op);

switch (layoutSize \* OpLayoutType::Count + nType)

{

#define LAYOUT\_TYPE(layout) \

case OpLayoutType::layout: \

Assert(layoutSize == SmallLayout); \

Dump##layout(op, reader.layout(), dumpFunction, reader); \

break;

#define LAYOUT\_TYPE\_WMS(layout) \

case SmallLayout \* OpLayoutType::Count + OpLayoutType::layout: \

Dump##layout(op, reader.layout##\_Small(), dumpFunction, reader); \

break; \

case MediumLayout \* OpLayoutType::Count + OpLayoutType::layout: \

Dump##layout(op, reader.layout##\_Medium(), dumpFunction, reader); \

break; \

case LargeLayout \* OpLayoutType::Count + OpLayoutType::layout: \

Dump##layout(op, reader.layout##\_Large(), dumpFunction, reader); \

break;

#include "LayoutTypes.h"

default:

{

AssertMsg(false, "Unknown OpLayout");

break;

}

}

}

void ByteCodeDumper::DumpR8(double value)

{

Output::Print(L" double:%g ", value);

}

void ByteCodeDumper::DumpReg(RegSlot registerID)

{

Output::Print(L" R%d ", (int) registerID);

}

void ByteCodeDumper::DumpReg(RegSlot\_TwoByte registerID)

{

Output::Print(L" R%d ", (int) registerID);

}

void ByteCodeDumper::DumpReg(RegSlot\_OneByte registerID)

{

Output::Print(L" R%d ", (int) registerID);

}

void ByteCodeDumper::DumpProfileId(uint id)

{

Output::Print(L" <%d> ", id);

}

void ByteCodeDumper::DumpEmpty(OpCode op, const unaligned OpLayoutEmpty \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

}

template <class T>

void ByteCodeDumper::DumpCallI(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

if (data->Return != Constants::NoRegister)

{

DumpReg((RegSlot)data->Return);

Output::Print(L"=");

}

Output::Print(L" R%d(ArgCount: %d)", data->Function, data->ArgCount);

}

template <class T>

void ByteCodeDumper::DumpCallIExtended(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpCallI(op, data, dumpFunction, reader);

if (data->Options & Js::CallIExtended\_SpreadArgs)

{

const Js::AuxArray<uint32> \*arr = reader.ReadAuxArray<uint32>(data->SpreadAuxOffset, dumpFunction);

Output::Print(L" spreadArgs [", arr->count);

for (uint i = 0; i < arr->count; i++)

{

if (i > 10)

{

Output::Print(L", ...");

break;

}

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%u", arr->elements[i]);

}

Output::Print(L"]");

}

}

template <class T>

void ByteCodeDumper::DumpCallIFlags(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpCallI(op, data, dumpFunction, reader);

Output::Print(L" <%04x> ", data->callFlags);

}

template <class T>

void ByteCodeDumper::DumpCallIExtendedFlags(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpCallIFlags(op, data, dumpFunction, reader);

if (data->Options & Js::CallIExtended\_SpreadArgs)

{

const Js::AuxArray<uint32> \*arr = reader.ReadAuxArray<uint32>(data->SpreadAuxOffset, dumpFunction);

Output::Print(L" spreadArgs [", arr->count);

for (uint i = 0; i < arr->count; i++)

{

if (i > 10)

{

Output::Print(L", ...");

break;

}

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%u", arr->elements[i]);

}

Output::Print(L"]");

}

}

template <class T>

void ByteCodeDumper::DumpCallIExtendedFlagsWithICIndex(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpCallIFlags(op, data, dumpFunction, reader);

DumpCallIWithICIndex(op, data, dumpFunction, reader);

if (data->Options & Js::CallIExtended\_SpreadArgs)

{

const Js::AuxArray<uint32> \*arr = reader.ReadAuxArray<uint32>(data->SpreadAuxOffset, dumpFunction);

Output::Print(L" spreadArgs [", arr->count);

for (uint i = 0; i < arr->count; i++)

{

if (i > 10)

{

Output::Print(L", ...");

break;

}

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%u", arr->elements[i]);

}

Output::Print(L"]");

}

}

template <class T>

void ByteCodeDumper::DumpCallIWithICIndex(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpCallI(op, data, dumpFunction, reader);

Output::Print(L" <%d> ", data->inlineCacheIndex);

}

template <class T>

void ByteCodeDumper::DumpCallIFlagsWithICIndex(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpCallI(op, data, dumpFunction, reader);

Output::Print(L" <%d> ", data->inlineCacheIndex);

Output::Print(L" <%d> ", data->callFlags);

}

template <class T>

void ByteCodeDumper::DumpCallIExtendedWithICIndex(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpCallIWithICIndex(op, data, dumpFunction, reader);

if (data->Options & Js::CallIExtended\_SpreadArgs)

{

const Js::AuxArray<uint32> \*arr = reader.ReadAuxArray<uint32>(data->SpreadAuxOffset, dumpFunction);

Output::Print(L" spreadArgs [", arr->count);

for (uint i=0; i < arr->count; i++)

{

if (i > 10)

{

Output::Print(L", ...");

break;

}

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%u", arr->elements[i]);

}

Output::Print(L"]");

}

}

template <class T>

void ByteCodeDumper::DumpElementI(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::ProfiledLdElemI\_A:

case OpCode::LdElemI\_A:

case OpCode::LdMethodElem:

case OpCode::TypeofElem:

{

Output::Print(L" R%d = R%d[R%d]", data->Value, data->Instance, data->Element);

break;

}

case OpCode::ProfiledStElemI\_A:

case OpCode::ProfiledStElemI\_A\_Strict:

case OpCode::StElemI\_A:

case OpCode::StElemI\_A\_Strict:

case OpCode::InitSetElemI:

case OpCode::InitGetElemI:

case OpCode::InitComputedProperty:

case OpCode::InitClassMemberComputedName:

case OpCode::InitClassMemberGetComputedName:

case OpCode::InitClassMemberSetComputedName:

{

Output::Print(L" R%d[R%d] = R%d", data->Instance, data->Element, data->Value);

break;

}

case OpCode::DeleteElemI\_A:

case OpCode::DeleteElemIStrict\_A:

{

Output::Print(L" R%d[R%d]", data->Instance, data->Element);

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementI");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpReg2Int1(OpCode op, const unaligned T\* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::LdThis:

case OpCode::ProfiledLdThis:

Output::Print(L" R%d = R%d, %d", data->R0, data->R1, data->C1);

break;

case OpCode::LdIndexedFrameDisplay:

Output::Print(L" R%d = [%d], R%d ", data->R0, data->C1, data->R1);

break;

case OpCode::GetCachedFunc:

DumpReg(data->R0);

Output::Print(L"= func(");

DumpReg(data->R1);

Output::Print(L",");

DumpI4(data->C1);

Output::Print(L")");

break;

default:

AssertMsg(false, "Unknown OpCode for OpLayoutReg2Int1");

break;

}

}

template <class T>

void ByteCodeDumper::DumpElementScopedU(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

switch (op)

{

case OpCode::LdElemUndefScoped:

{

Output::Print(L" %s = undefined, R%d", pPropertyName->GetBuffer(), Js::FunctionBody::RootObjectRegSlot);

break;

}

case OpCode::InitUndeclConsoleLetFld:

case OpCode::InitUndeclConsoleConstFld:

{

Output::Print(L" %s = undefined", pPropertyName->GetBuffer());

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for ElementScopedU");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementU(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

switch (op)

{

case OpCode::LdElemUndef:

{

Output::Print(L" R%d.%s = undefined", data->Instance, pPropertyName->GetBuffer());

break;

}

// TODO: Change InitUndeclLetFld and InitUndeclConstFld to ElementU layout

// case OpCode::InitUndeclLetFld:

// case OpCode::InitUndeclConstFld:

// {

// PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(data->PropertyIndex);

// Output::Print(L" R%d.%s", data->Instance, pPropertyName->GetBuffer());

// break;

// }

case OpCode::ClearAttributes:

{

Output::Print(L" R%d.%s.writable/enumerable/configurable = 0", data->Instance, pPropertyName->GetBuffer());

break;

}

case OpCode::DeleteLocalFld:

Output::Print(L" R%d = %s ", data->Instance, pPropertyName->GetBuffer());

break;

case OpCode::StLocalFuncExpr:

Output::Print(L" %s = R%d", pPropertyName->GetBuffer(), data->Instance);

break;

default:

{

AssertMsg(false, "Unknown OpCode for ElementU");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementRootU(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

switch (op)

{

case OpCode::InitUndeclRootLetFld:

case OpCode::InitUndeclRootConstFld:

case OpCode::EnsureNoRootFld:

case OpCode::EnsureNoRootRedeclFld:

{

Output::Print(L" root.%s", pPropertyName->GetBuffer());

break;

}

case OpCode::LdLocalElemUndef:

{

Output::Print(L" %s = undefined", pPropertyName->GetBuffer());

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for ElementRootU");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementScopedC(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

switch (op)

{

case OpCode::ScopedEnsureNoRedeclFld:

case OpCode::ScopedDeleteFld:

case OpCode::ScopedDeleteFldStrict:

{

Output::Print(L" %s, R%d", pPropertyName->GetBuffer(), data->Value);

break;

}

case OpCode::ScopedInitFunc:

{

Output::Print(L" %s = R%d, R%d", pPropertyName->GetBuffer(), data->Value,

Js::FunctionBody::RootObjectRegSlot);

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementScopedC");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementC(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

switch (op)

{

case OpCode::DeleteFld:

case OpCode::DeleteRootFld:

case OpCode::DeleteFldStrict:

case OpCode::DeleteRootFldStrict:

{

Output::Print(L" R%d.%s", data->Instance, pPropertyName->GetBuffer());

break;

}

case OpCode::InitSetFld:

case OpCode::InitGetFld:

case OpCode::InitClassMemberGet:

case OpCode::InitClassMemberSet:

{

Output::Print(L" R%d.%s = (Set/Get) R%d", data->Instance, pPropertyName->GetBuffer(),

data->Value);

break;

}

case OpCode::StFuncExpr:

case OpCode::InitProto:

{

Output::Print(L" R%d.%s = R%d", data->Instance, pPropertyName->GetBuffer(),

data->Value);

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementC");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementScopedC2(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

switch (op)

{

case OpCode::ScopedLdInst:

{

Output::Print(L" R%d, R%d = %s", data->Value, data->Value2, pPropertyName->GetBuffer());

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementScopedC2");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementC2(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

switch (op)

{

case OpCode::LdSuperFld:

{

Output::Print(L" R%d = R%d(this=R%d).%s #%d", data->Value, data->Instance, data->Value2,

pPropertyName->GetBuffer(), data->PropertyIdIndex);

break;

}

case OpCode::ProfiledLdSuperFld:

{

Output::Print(L" R%d = R%d(this=R%d).%s #%d", data->Value, data->Instance, data->Value2,

pPropertyName->GetBuffer(), data->PropertyIdIndex);

DumpProfileId(data->PropertyIdIndex);

break;

}

case OpCode::StSuperFld:

{

Output::Print(L" R%d.%s(this=R%d) = R%d #%d", data->Instance, pPropertyName->GetBuffer(),

data->Value2, data->Value, data->PropertyIdIndex);

break;

}

case OpCode::ProfiledStSuperFld:

{

Output::Print(L" R%d.%s(this=R%d) = R%d #%d", data->Instance, pPropertyName->GetBuffer(),

data->Value2, data->Value, data->PropertyIdIndex);

DumpProfileId(data->PropertyIdIndex);

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementC2");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpReg1Unsigned1(OpCode op, const unaligned T\* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::InvalCachedScope:

#if ENABLE\_NATIVE\_CODEGEN

case OpCode::NewScopeSlots:

#endif

Output::Print(L" R%u[%u]", data->R0, data->C1);

break;

case OpCode::NewRegEx:

{

DumpReg(data->R0);

#if DBG

Output::Print(L"=");

UnifiedRegex::DebugWriter w;

dumpFunction->GetLiteralRegex(data->C1)->Print(&w);

#else

Output::Print(L"=<regex>");

#endif

break;

}

default:

DumpReg(data->R0);

Output::Print(L"=");

DumpU4(data->C1);

break;

};

}

template <class T>

void ByteCodeDumper::DumpElementSlot(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::NewInnerStackScFunc:

case OpCode::NewInnerScFunc:

case OpCode::NewInnerScGenFunc:

{

FunctionProxy\* pfuncActual = dumpFunction->GetNestedFunc((uint)data->SlotIndex)->GetFunctionProxy();

Output::Print(L" R%d = env:R%d, %s()", data->Value, data->Instance,

pfuncActual->EnsureDeserialized()->GetDisplayName());

break;

}

#if ENABLE\_NATIVE\_CODEGEN

case OpCode::StSlot:

case OpCode::StSlotChkUndecl:

#endif

case OpCode::StObjSlot:

case OpCode::StObjSlotChkUndecl:

Output::Print(L" R%d[%d] = R%d ",data->Instance,data->SlotIndex,data->Value);

break;

case OpCode::LdSlot:

#if ENABLE\_NATIVE\_CODEGEN

case OpCode::LdSlotArr:

#endif

case OpCode::LdObjSlot:

Output::Print(L" R%d = R%d[%d] ",data->Value,data->Instance,data->SlotIndex);

break;

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementSlot");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementSlotI1(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::StLocalSlot:

case OpCode::StLocalObjSlot:

case OpCode::StLocalSlotChkUndecl:

case OpCode::StLocalObjSlotChkUndecl:

Output::Print(L" [%d] = R%d ",data->SlotIndex, data->Value);

break;

case OpCode::LdLocalSlot:

case OpCode::LdEnvObj:

case OpCode::LdLocalObjSlot:

Output::Print(L" R%d = [%d] ",data->Value, data->SlotIndex);

break;

case OpCode::NewScFunc:

case OpCode::NewStackScFunc:

case OpCode::NewScGenFunc:

{

FunctionProxy\* pfuncActual = dumpFunction->GetNestedFunc((uint)data->SlotIndex)->GetFunctionProxy();

Output::Print(L" R%d = %s()", data->Value,

pfuncActual->EnsureDeserialized()->GetDisplayName());

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementSlotI1");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementSlotI2(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::StInnerSlot:

case OpCode::StInnerObjSlot:

case OpCode::StEnvSlot:

case OpCode::StEnvObjSlot:

case OpCode::StEnvSlotChkUndecl:

case OpCode::StEnvObjSlotChkUndecl:

Output::Print(L" [%d][%d] = R%d ",data->SlotIndex1, data->SlotIndex2, data->Value);

break;

case OpCode::LdInnerSlot:

case OpCode::LdInnerObjSlot:

case OpCode::LdEnvSlot:

case OpCode::LdEnvObjSlot:

Output::Print(L" R%d = [%d][%d] ",data->Value, data->SlotIndex1, data->SlotIndex2);

break;

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementSlotI2");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementP(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyId propertyId = dumpFunction->GetPropertyIdFromCacheId(data->inlineCacheIndex);

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(propertyId);

switch (op)

{

case OpCode::ScopedLdFldForTypeOf:

case OpCode::ScopedLdFld:

Output::Print(L" R%d = %s, R%d #%d", data->Value, pPropertyName->GetBuffer(),

Js::FunctionBody::RootObjectRegSlot, data->inlineCacheIndex);

break;

case OpCode::ScopedStFld:

case OpCode::ConsoleScopedStFld:

case OpCode::ScopedStFldStrict:

Output::Print(L" %s = R%d, R%d #%d", pPropertyName->GetBuffer(), data->Value,

Js::FunctionBody::RootObjectRegSlot, data->inlineCacheIndex);

break;

case OpCode::LdLocalFld:

Output::Print(L" R%d = %s #%d", data->Value, pPropertyName->GetBuffer(), data->inlineCacheIndex);

break;

case OpCode::ProfiledLdLocalFld:

Output::Print(L" R%d = %s #%d", data->Value, pPropertyName->GetBuffer(), data->inlineCacheIndex);

DumpProfileId(data->inlineCacheIndex);

break;

case OpCode::StLocalFld:

case OpCode::InitLocalFld:

case OpCode::InitLocalLetFld:

case OpCode::InitUndeclLocalLetFld:

case OpCode::InitUndeclLocalConstFld:

Output::Print(L" %s = R%d #%d", pPropertyName->GetBuffer(), data->Value, data->inlineCacheIndex);

break;

case OpCode::ProfiledStLocalFld:

case OpCode::ProfiledInitLocalFld:

Output::Print(L" %s = R%d #%d", pPropertyName->GetBuffer(), data->Value, data->inlineCacheIndex);

DumpProfileId(data->inlineCacheIndex);

break;

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementP");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementPIndexed(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyId propertyId = dumpFunction->GetPropertyIdFromCacheId(data->inlineCacheIndex);

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(propertyId);

switch (op)

{

case OpCode::InitInnerFld:

case OpCode::InitInnerLetFld:

case OpCode::InitUndeclLetFld:

case OpCode::InitUndeclConstFld:

Output::Print(L" [%d].%s = R%d #%d", data->scopeIndex, pPropertyName->GetBuffer(), data->Value, data->inlineCacheIndex);

break;

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementPIndexed");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementCP(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyId propertyId = dumpFunction->GetPropertyIdFromCacheId(data->inlineCacheIndex);

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(propertyId);

switch (op)

{

case OpCode::LdFldForTypeOf:

case OpCode::LdFld:

case OpCode::LdFldForCallApplyTarget:

case OpCode::LdMethodFld:

case OpCode::ScopedLdMethodFld:

{

Output::Print(L" R%d = R%d.%s #%d", data->Value, data->Instance,

pPropertyName->GetBuffer(), data->inlineCacheIndex);

break;

}

case OpCode::InitFld:

case OpCode::InitLetFld:

case OpCode::InitConstFld:

case OpCode::StFld:

case OpCode::StFldStrict:

case OpCode::InitClassMember:

{

Output::Print(L" R%d.%s = R%d #%d", data->Instance, pPropertyName->GetBuffer(),

data->Value, data->inlineCacheIndex);

break;

}

case OpCode::ProfiledLdFldForTypeOf:

case OpCode::ProfiledLdFld:

case OpCode::ProfiledLdFldForCallApplyTarget:

case OpCode::ProfiledLdMethodFld:

{

Output::Print(L" R%d = R%d.%s #%d", data->Value, data->Instance,

pPropertyName->GetBuffer(), data->inlineCacheIndex);

DumpProfileId(data->inlineCacheIndex);

break;

}

case OpCode::ProfiledInitFld:

case OpCode::ProfiledStFld:

case OpCode::ProfiledStFldStrict:

{

Output::Print(L" R%d.%s = R%d #%d", data->Instance, pPropertyName->GetBuffer(),

data->Value, data->inlineCacheIndex);

DumpProfileId(data->inlineCacheIndex);

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementCP");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementRootCP(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyId propertyId = dumpFunction->GetPropertyIdFromCacheId(data->inlineCacheIndex);

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(propertyId);

switch (op)

{

case OpCode::LdRootFld:

case OpCode::LdRootMethodFld:

case OpCode::LdRootFldForTypeOf:

{

Output::Print(L" R%d = root.%s #%d", data->Value,

pPropertyName->GetBuffer(), data->inlineCacheIndex);

break;

}

case OpCode::InitRootFld:

case OpCode::InitRootLetFld:

case OpCode::InitRootConstFld:

case OpCode::StRootFld:

case OpCode::StRootFldStrict:

{

Output::Print(L" root.%s = R%d #%d", pPropertyName->GetBuffer(),

data->Value, data->inlineCacheIndex);

break;

}

case OpCode::ProfiledLdRootFld:

case OpCode::ProfiledLdRootFldForTypeOf:

case OpCode::ProfiledLdRootMethodFld:

{

Output::Print(L" R%d = root.%s #%d", data->Value,

pPropertyName->GetBuffer(), data->inlineCacheIndex);

DumpProfileId(data->inlineCacheIndex);

break;

}

case OpCode::ProfiledInitRootFld:

case OpCode::ProfiledStRootFld:

case OpCode::ProfiledStRootFldStrict:

{

Output::Print(L" root.%s = R%d #%d", pPropertyName->GetBuffer(),

data->Value, data->inlineCacheIndex);

DumpProfileId(data->inlineCacheIndex);

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutElementRootCP");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpElementUnsigned1(OpCode op, const unaligned T \* data, Js::FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::StArrItemC\_CI4:

case OpCode::StArrItemI\_CI4:

case OpCode::StArrSegItem\_CI4:

case OpCode::StArrInlineItem\_CI4:

Output::Print(L" R%d[", data->Instance);

DumpI4(data->Element);

Output::Print(L"] = R%d", data->Value);

break;

default:

AssertMsg(false, "Unknown OpCode for OpLayoutElementUnsigned1");

break;

}

}

template <class T>

void ByteCodeDumper::DumpArg(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::ProfiledArgOut\_A:

case OpCode::ArgOut\_A:

case OpCode::ArgOut\_ANonVar:

{

Output::Print(L" Out%d =", (int) data->Arg);

DumpReg(data->Reg);

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutArg");

break;

}

}

}

template <class T>

void ByteCodeDumper::DumpArgNoSrc(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case Js::OpCode::ArgOut\_Env:

{

Output::Print(L" Out%d ", (int) data->Arg);

break;

}

default:

{

AssertMsg(false, "Unknown OpCode for OpLayoutArgNoSrc");

break;

}

}

}

void

ByteCodeDumper::DumpStartCall(OpCode op, const unaligned OpLayoutStartCall \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

Assert(op == OpCode::StartCall );

Output::Print(L" ArgCount: %d", data->ArgCount);

}

template <class T> void

ByteCodeDumper::DumpUnsigned1(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpU4(data->C1);

}

template <class T> void

ByteCodeDumper::DumpReg1(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::ObjectFreeze:

Output::Print(L" R%d.freeze()", data->R0);

break;

default:

DumpReg(data->R0);

break;

}

}

template <class T> void

ByteCodeDumper::DumpReg2(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpReg(data->R0);

DumpReg(data->R1);

}

template <class T> void

ByteCodeDumper::DumpReg2WithICIndex(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpReg2(op, data, dumpFunction, reader);

Output::Print(L" <%d> ", data->inlineCacheIndex);

}

template <class T> void

ByteCodeDumper::DumpReg3(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::NewInnerScopeSlots:

Output::Print(L" [%d], %d, %d ", data->R0, data->R1, data->R2);

break;

default:

DumpReg(data->R0);

DumpReg(data->R1);

DumpReg(data->R2);

break;

}

}

template <class T> void

ByteCodeDumper::DumpReg3C(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::IsInst:

Output::Print(L"R%d = R%d instanceof R%d #%d",

data->R0, data->R1, data->R2, data->inlineCacheIndex);

break;

default:

AssertMsg(false, "Unknown OpCode for OpLayoutReg3C");

}

}

template <class T> void

ByteCodeDumper::DumpReg4(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpReg(data->R0);

DumpReg(data->R1);

DumpReg(data->R2);

DumpReg(data->R3);

}

template <class T> void

ByteCodeDumper::DumpReg2B1(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpReg(data->R0);

DumpReg(data->R1);

DumpI4(data->B2);

}

template <class T> void

ByteCodeDumper::DumpReg3B1(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpReg(data->R0);

DumpReg(data->R1);

DumpReg(data->R2);

DumpI4(data->B3);

}

template <class T> void

ByteCodeDumper::DumpReg5(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpReg(data->R0);

DumpReg(data->R1);

DumpReg(data->R2);

DumpReg(data->R3);

DumpReg(data->R4);

}

void

ByteCodeDumper::DumpW1(OpCode op, const unaligned OpLayoutW1 \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpU2(data->C1);

}

void

ByteCodeDumper::DumpReg1Int2(OpCode op, const unaligned OpLayoutReg1Int2 \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpReg(data->R0);

Output::Print(L"=");

DumpI4(data->C1);

DumpI4(data->C2);

}

void

ByteCodeDumper::DumpAuxNoReg(OpCode op, const unaligned OpLayoutAuxNoReg \* playout, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::CommitScope:

{

const Js::PropertyIdArray \*propIds = reader.ReadPropertyIdArray(playout->Offset, dumpFunction);

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

Output::Print(L" %d [", propIds->count);

for (uint i=0; i < propIds->count && i < 3; i++)

{

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(propIds->elements[i]);

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%s", pPropertyName->GetBuffer());

}

Output::Print(L"]");

break;

}

case Js::OpCode::InitCachedFuncs:

{

const Js::FuncInfoArray \*arr = reader.ReadAuxArray<FuncInfoEntry>(playout->Offset, dumpFunction);

Output::Print(L" %d [", arr->count);

for (uint i = 0; i < arr->count && i < 3; i++)

{

Js::ParseableFunctionInfo \*info = dumpFunction->GetNestedFunctionForExecution(arr->elements[i].nestedIndex);

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%s", info->GetDisplayName());

}

Output::Print(L"]");

break;

}

default:

AssertMsg(false, "Unknown OpCode for OpLayoutType::AuxNoReg");

break;

}

}

void

ByteCodeDumper::DumpAuxiliary(OpCode op, const unaligned OpLayoutAuxiliary \* playout, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case OpCode::NewScObjectLiteral:

case OpCode::LdPropIds:

{

const Js::PropertyIdArray \*propIds = reader.ReadPropertyIdArray(playout->Offset, dumpFunction);

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

DumpReg(playout->R0);

Output::Print(L"= %d [", propIds->count);

for (uint i=0; i< propIds->count && i < 3; i++)

{

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(propIds->elements[i]);

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%s", pPropertyName->GetBuffer());

}

if (propIds->count >= 3)

{

Output::Print(L", ...");

}

Output::Print(L"], LiteralId %d", playout->C1);

break;

}

case OpCode::StArrSegItem\_A:

{

const Js::VarArray \*vars = reader.ReadAuxArray<Var>(playout->Offset, dumpFunction);

DumpReg(playout->R0);

Output::Print(L"= %d [", vars->count);

uint i=0;

for (; i<vars->count && i < 3; i++)

{

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%d", vars->elements[i]);

}

if (i != vars->count)

{

Output::Print(L", ...");

}

Output::Print(L"]");

break;

}

case OpCode::NewScIntArray:

{

const Js::AuxArray<int32> \*intArray = reader.ReadAuxArray<int32>(playout->Offset, dumpFunction);

Output::Print(L" R%d = %d [", playout->R0, intArray->count);

uint i;

for (i = 0; i<intArray->count && i < 3; i++)

{

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%d", intArray->elements[i]);

}

if (i != intArray->count)

{

Output::Print(L", ...");

}

Output::Print(L"]");

break;

}

case OpCode::NewScFltArray:

{

const Js::AuxArray<double> \*dblArray = reader.ReadAuxArray<double>(playout->Offset, dumpFunction);

Output::Print(L" R%d = %d [", playout->R0, dblArray->count);

uint i;

for (i = 0; i<dblArray->count && i < 3; i++)

{

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%f", dblArray->elements[i]);

}

if (i != dblArray->count)

{

Output::Print(L", ...");

}

Output::Print(L"]");

break;

}

case OpCode::NewScObject\_A:

{

const Js::VarArrayVarCount \*vars = reader.ReadVarArrayVarCount(playout->Offset, dumpFunction);

DumpReg(playout->R0);

int count = Js::TaggedInt::ToInt32(vars->count);

Output::Print(L"= %d [", count);

int i=0;

for (; i<count && i < 3; i++)

{

if (i != 0)

{

Output::Print(L", ");

}

if (TaggedInt::Is(vars->elements[i]))

{

Output::Print(L"%d", TaggedInt::ToInt32(vars->elements[i]));

}

else if (JavascriptNumber::Is(vars->elements[i]))

{

Output::Print(L"%g", JavascriptNumber::GetValue(vars->elements[i]));

}

else

{

Assert(false);

}

}

if (i != count)

{

Output::Print(L", ...");

}

Output::Print(L"]");

break;

}

default:

AssertMsg(false, "Unknown OpCode for OpLayoutType::Auxiliary");

break;

}

}

void

ByteCodeDumper::DumpReg2Aux(OpCode op, const unaligned OpLayoutReg2Aux \* playout, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

switch (op)

{

case Js::OpCode::SpreadArrayLiteral:

{

const Js::AuxArray<uint32> \*arr = reader.ReadAuxArray<uint32>(playout->Offset, dumpFunction);

Output::Print(L" R%u <- R%u, %u spreadArgs [", playout->R0, playout->R1, arr->count);

for (uint i = 0; i < arr->count; i++)

{

if (i > 10)

{

Output::Print(L", ...");

break;

}

if (i != 0)

{

Output::Print(L", ");

}

Output::Print(L"%u", arr->elements[i]);

}

Output::Print(L"]");

break;

}

default:

AssertMsg(false, "Unknown OpCode for OpLayoutType::Reg2Aux");

break;

}

}

template <class T>

void ByteCodeDumper::DumpClass(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpReg(data->Constructor);

if (data->Extends != Js::Constants::NoRegister)

{

Output::Print(L"extends");

DumpReg((RegSlot)data->Extends);

}

}

#ifdef BYTECODE\_BRANCH\_ISLAND

void ByteCodeDumper::DumpBrLong(OpCode op, const unaligned OpLayoutBrLong\* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpOffset(data->RelativeJumpOffset, reader);

}

#endif

void ByteCodeDumper::DumpBr(OpCode op, const unaligned OpLayoutBr \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpOffset(data->RelativeJumpOffset, reader);

}

void ByteCodeDumper::DumpBrS(OpCode op, const unaligned OpLayoutBrS \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpOffset(data->RelativeJumpOffset, reader);

DumpI4(data->val);

}

template <class T>

void ByteCodeDumper::DumpBrReg1(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpOffset(data->RelativeJumpOffset, reader);

DumpReg(data->R1);

}

template <class T>

void ByteCodeDumper::DumpBrReg2(OpCode op, const unaligned T \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpOffset(data->RelativeJumpOffset, reader);

DumpReg(data->R1);

DumpReg(data->R2);

}

void ByteCodeDumper::DumpBrProperty(OpCode op, const unaligned OpLayoutBrProperty \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpOffset(data->RelativeJumpOffset, reader);

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

Output::Print(L"R%d.%s", data->Instance, pPropertyName->GetBuffer());

}

void ByteCodeDumper::DumpBrLocalProperty(OpCode op, const unaligned OpLayoutBrLocalProperty \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpOffset(data->RelativeJumpOffset, reader);

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

Output::Print(L"%s", pPropertyName->GetBuffer());

}

void ByteCodeDumper::DumpBrEnvProperty(OpCode op, const unaligned OpLayoutBrEnvProperty \* data, FunctionBody \* dumpFunction, ByteCodeReader& reader)

{

DumpOffset(data->RelativeJumpOffset, reader);

ScriptContext\* scriptContext = dumpFunction->GetScriptContext();

PropertyRecord const \* pPropertyName = scriptContext->GetPropertyName(

dumpFunction->GetReferencedPropertyId(data->PropertyIdIndex));

Output::Print(L"[%d].%s", data->SlotIndex, pPropertyName->GetBuffer());

}

} // namespace Js

#endif