<https://docs.microsoft.com/ru-ru/cpp/intrinsics/cpuid-cpuidex?view=msvc-160>

/ InstructionSet.cpp

// Compile by using: cl /EHsc /W4 InstructionSet.cpp

// processor: x86, x64

// Uses the \_\_cpuid intrinsic to get information about

// CPU extended instruction set support.

#include <iostream>

#include <vector>

#include <bitset>

#include <array>

#include <string>

#include <intrin.h>

class InstructionSet

{

// forward declarations

class InstructionSet\_Internal;

public:

// getters

static std::string Vendor(void) { return CPU\_Rep.vendor\_; }

static std::string Brand(void) { return CPU\_Rep.brand\_; }

static bool SSE3(void) { return CPU\_Rep.f\_1\_ECX\_[0]; }

static bool PCLMULQDQ(void) { return CPU\_Rep.f\_1\_ECX\_[1]; }

static bool MONITOR(void) { return CPU\_Rep.f\_1\_ECX\_[3]; }

static bool SSSE3(void) { return CPU\_Rep.f\_1\_ECX\_[9]; }

static bool FMA(void) { return CPU\_Rep.f\_1\_ECX\_[12]; }

static bool CMPXCHG16B(void) { return CPU\_Rep.f\_1\_ECX\_[13]; }

static bool SSE41(void) { return CPU\_Rep.f\_1\_ECX\_[19]; }

static bool SSE42(void) { return CPU\_Rep.f\_1\_ECX\_[20]; }

static bool MOVBE(void) { return CPU\_Rep.f\_1\_ECX\_[22]; }

static bool POPCNT(void) { return CPU\_Rep.f\_1\_ECX\_[23]; }

static bool AES(void) { return CPU\_Rep.f\_1\_ECX\_[25]; }

static bool XSAVE(void) { return CPU\_Rep.f\_1\_ECX\_[26]; }

static bool OSXSAVE(void) { return CPU\_Rep.f\_1\_ECX\_[27]; }

static bool AVX(void) { return CPU\_Rep.f\_1\_ECX\_[28]; }

static bool F16C(void) { return CPU\_Rep.f\_1\_ECX\_[29]; }

static bool RDRAND(void) { return CPU\_Rep.f\_1\_ECX\_[30]; }

static bool MSR(void) { return CPU\_Rep.f\_1\_EDX\_[5]; }

static bool CX8(void) { return CPU\_Rep.f\_1\_EDX\_[8]; }

static bool SEP(void) { return CPU\_Rep.f\_1\_EDX\_[11]; }

static bool CMOV(void) { return CPU\_Rep.f\_1\_EDX\_[15]; }

static bool CLFSH(void) { return CPU\_Rep.f\_1\_EDX\_[19]; }

static bool MMX(void) { return CPU\_Rep.f\_1\_EDX\_[23]; }

static bool FXSR(void) { return CPU\_Rep.f\_1\_EDX\_[24]; }

static bool SSE(void) { return CPU\_Rep.f\_1\_EDX\_[25]; }

static bool SSE2(void) { return CPU\_Rep.f\_1\_EDX\_[26]; }

static bool FSGSBASE(void) { return CPU\_Rep.f\_7\_EBX\_[0]; }

static bool BMI1(void) { return CPU\_Rep.f\_7\_EBX\_[3]; }

static bool HLE(void) { return CPU\_Rep.isIntel\_ && CPU\_Rep.f\_7\_EBX\_[4]; }

static bool AVX2(void) { return CPU\_Rep.f\_7\_EBX\_[5]; }

static bool BMI2(void) { return CPU\_Rep.f\_7\_EBX\_[8]; }

static bool ERMS(void) { return CPU\_Rep.f\_7\_EBX\_[9]; }

static bool INVPCID(void) { return CPU\_Rep.f\_7\_EBX\_[10]; }

static bool RTM(void) { return CPU\_Rep.isIntel\_ && CPU\_Rep.f\_7\_EBX\_[11]; }

static bool AVX512F(void) { return CPU\_Rep.f\_7\_EBX\_[16]; }

static bool RDSEED(void) { return CPU\_Rep.f\_7\_EBX\_[18]; }

static bool ADX(void) { return CPU\_Rep.f\_7\_EBX\_[19]; }

static bool AVX512PF(void) { return CPU\_Rep.f\_7\_EBX\_[26]; }

static bool AVX512ER(void) { return CPU\_Rep.f\_7\_EBX\_[27]; }

static bool AVX512CD(void) { return CPU\_Rep.f\_7\_EBX\_[28]; }

static bool SHA(void) { return CPU\_Rep.f\_7\_EBX\_[29]; }

static bool PREFETCHWT1(void) { return CPU\_Rep.f\_7\_ECX\_[0]; }

static bool LAHF(void) { return CPU\_Rep.f\_81\_ECX\_[0]; }

static bool LZCNT(void) { return CPU\_Rep.isIntel\_ && CPU\_Rep.f\_81\_ECX\_[5]; }

static bool ABM(void) { return CPU\_Rep.isAMD\_ && CPU\_Rep.f\_81\_ECX\_[5]; }

static bool SSE4a(void) { return CPU\_Rep.isAMD\_ && CPU\_Rep.f\_81\_ECX\_[6]; }

static bool XOP(void) { return CPU\_Rep.isAMD\_ && CPU\_Rep.f\_81\_ECX\_[11]; }

static bool TBM(void) { return CPU\_Rep.isAMD\_ && CPU\_Rep.f\_81\_ECX\_[21]; }

static bool SYSCALL(void) { return CPU\_Rep.isIntel\_ && CPU\_Rep.f\_81\_EDX\_[11]; }

static bool MMXEXT(void) { return CPU\_Rep.isAMD\_ && CPU\_Rep.f\_81\_EDX\_[22]; }

static bool RDTSCP(void) { return CPU\_Rep.isIntel\_ && CPU\_Rep.f\_81\_EDX\_[27]; }

static bool \_3DNOWEXT(void) { return CPU\_Rep.isAMD\_ && CPU\_Rep.f\_81\_EDX\_[30]; }

static bool \_3DNOW(void) { return CPU\_Rep.isAMD\_ && CPU\_Rep.f\_81\_EDX\_[31]; }

private:

static const InstructionSet\_Internal CPU\_Rep;

class InstructionSet\_Internal

{

public:

InstructionSet\_Internal()

: nIds\_{ 0 },

nExIds\_{ 0 },

isIntel\_{ false },

isAMD\_{ false },

f\_1\_ECX\_{ 0 },

f\_1\_EDX\_{ 0 },

f\_7\_EBX\_{ 0 },

f\_7\_ECX\_{ 0 },

f\_81\_ECX\_{ 0 },

f\_81\_EDX\_{ 0 },

data\_{},

extdata\_{}

{

//int cpuInfo[4] = {-1};

std::array<int, 4> cpui;

// Calling \_\_cpuid with 0x0 as the function\_id argument

// gets the number of the highest valid function ID.

\_\_cpuid(cpui.data(), 0);

nIds\_ = cpui[0];

for (int i = 0; i <= nIds\_; ++i)

{

\_\_cpuidex(cpui.data(), i, 0);

data\_.push\_back(cpui);

}

// Capture vendor string

char vendor[0x20];

memset(vendor, 0, sizeof(vendor));

\*reinterpret\_cast<int\*>(vendor) = data\_[0][1];

\*reinterpret\_cast<int\*>(vendor + 4) = data\_[0][3];

\*reinterpret\_cast<int\*>(vendor + 8) = data\_[0][2];

vendor\_ = vendor;

if (vendor\_ == "GenuineIntel")

{

isIntel\_ = true;

}

else if (vendor\_ == "AuthenticAMD")

{

isAMD\_ = true;

}

// load bitset with flags for function 0x00000001

if (nIds\_ >= 1)

{

f\_1\_ECX\_ = data\_[1][2];

f\_1\_EDX\_ = data\_[1][3];

}

// load bitset with flags for function 0x00000007

if (nIds\_ >= 7)

{

f\_7\_EBX\_ = data\_[7][1];

f\_7\_ECX\_ = data\_[7][2];

}

// Calling \_\_cpuid with 0x80000000 as the function\_id argument

// gets the number of the highest valid extended ID.

\_\_cpuid(cpui.data(), 0x80000000);

nExIds\_ = cpui[0];

char brand[0x40];

memset(brand, 0, sizeof(brand));

for (int i = 0x80000000; i <= nExIds\_; ++i)

{

\_\_cpuidex(cpui.data(), i, 0);

extdata\_.push\_back(cpui);

}

// load bitset with flags for function 0x80000001

if (nExIds\_ >= 0x80000001)

{

f\_81\_ECX\_ = extdata\_[1][2];

f\_81\_EDX\_ = extdata\_[1][3];

}

// Interpret CPU brand string if reported

if (nExIds\_ >= 0x80000004)

{

memcpy(brand, extdata\_[2].data(), sizeof(cpui));

memcpy(brand + 16, extdata\_[3].data(), sizeof(cpui));

memcpy(brand + 32, extdata\_[4].data(), sizeof(cpui));

brand\_ = brand;

}

};

int nIds\_;

int nExIds\_;

std::string vendor\_;

std::string brand\_;

bool isIntel\_;

bool isAMD\_;

std::bitset<32> f\_1\_ECX\_;

std::bitset<32> f\_1\_EDX\_;

std::bitset<32> f\_7\_EBX\_;

std::bitset<32> f\_7\_ECX\_;

std::bitset<32> f\_81\_ECX\_;

std::bitset<32> f\_81\_EDX\_;

std::vector<std::array<int, 4>> data\_;

std::vector<std::array<int, 4>> extdata\_;

};

};

// Initialize static member data

const InstructionSet::InstructionSet\_Internal InstructionSet::CPU\_Rep;

// Print out supported instruction set extensions

int main()

{

auto& outstream = std::cout;

auto support\_message = [&outstream](std::string isa\_feature, bool is\_supported) {

outstream << isa\_feature << (is\_supported ? " supported" : " not supported") << std::endl;

};

std::cout << InstructionSet::Vendor() << std::endl;

std::cout << InstructionSet::Brand() << std::endl;

support\_message("3DNOW", InstructionSet::\_3DNOW());

support\_message("3DNOWEXT", InstructionSet::\_3DNOWEXT());

support\_message("ABM", InstructionSet::ABM());

support\_message("ADX", InstructionSet::ADX());

support\_message("AES", InstructionSet::AES());

support\_message("AVX", InstructionSet::AVX());

support\_message("AVX2", InstructionSet::AVX2());

support\_message("AVX512CD", InstructionSet::AVX512CD());

support\_message("AVX512ER", InstructionSet::AVX512ER());

support\_message("AVX512F", InstructionSet::AVX512F());

support\_message("AVX512PF", InstructionSet::AVX512PF());

support\_message("BMI1", InstructionSet::BMI1());

support\_message("BMI2", InstructionSet::BMI2());

support\_message("CLFSH", InstructionSet::CLFSH());

support\_message("CMPXCHG16B", InstructionSet::CMPXCHG16B());

support\_message("CX8", InstructionSet::CX8());

support\_message("ERMS", InstructionSet::ERMS());

support\_message("F16C", InstructionSet::F16C());

support\_message("FMA", InstructionSet::FMA());

support\_message("FSGSBASE", InstructionSet::FSGSBASE());

support\_message("FXSR", InstructionSet::FXSR());

support\_message("HLE", InstructionSet::HLE());

support\_message("INVPCID", InstructionSet::INVPCID());

support\_message("LAHF", InstructionSet::LAHF());

support\_message("LZCNT", InstructionSet::LZCNT());

support\_message("MMX", InstructionSet::MMX());

support\_message("MMXEXT", InstructionSet::MMXEXT());

support\_message("MONITOR", InstructionSet::MONITOR());

support\_message("MOVBE", InstructionSet::MOVBE());

support\_message("MSR", InstructionSet::MSR());

support\_message("OSXSAVE", InstructionSet::OSXSAVE());

support\_message("PCLMULQDQ", InstructionSet::PCLMULQDQ());

support\_message("POPCNT", InstructionSet::POPCNT());

support\_message("PREFETCHWT1", InstructionSet::PREFETCHWT1());

support\_message("RDRAND", InstructionSet::RDRAND());

support\_message("RDSEED", InstructionSet::RDSEED());

support\_message("RDTSCP", InstructionSet::RDTSCP());

support\_message("RTM", InstructionSet::RTM());

support\_message("SEP", InstructionSet::SEP());

support\_message("SHA", InstructionSet::SHA());

support\_message("SSE", InstructionSet::SSE());

support\_message("SSE2", InstructionSet::SSE2());

support\_message("SSE3", InstructionSet::SSE3());

support\_message("SSE4.1", InstructionSet::SSE41());

support\_message("SSE4.2", InstructionSet::SSE42());

support\_message("SSE4a", InstructionSet::SSE4a());

support\_message("SSSE3", InstructionSet::SSSE3());

support\_message("SYSCALL", InstructionSet::SYSCALL());

support\_message("TBM", InstructionSet::TBM());

support\_message("XOP", InstructionSet::XOP());

support\_message("XSAVE", InstructionSet::XSAVE());

}