Identifying Support for Streaming SIMD Extensions in the Processor and Operating System

Version 1.1

01/99

Order Number: 244413-002

Table of Contents

Revision History

Revision	Revision History	Date
1.1	FCS Update	01/99

References

The following documents are referenced in this application note, and provide background or supporting information for understanding the topics presented in this document.

1. "AP-485, Intel Processor Identification with `CPUID` Instruction", Intel Corporation, Order # 241618-008.

1 Introduction

When developing an application to take advantage of Streaming SIMD Extensions, application developers must determine whether the system on which they are developing, and on which their final product will run, supports Streaming SIMD Extensions. Applications can invoke the appropriate algorithm or dll as needed for the particular system after querying both the processor and the operating as to their support for Streaming SIMD Extensions. There are four steps to doing this:

1. Check that the processor is an Intel processor that has the `CPUID` instruction.
2. Check that the processor supports Streaming SIMD Extensions.
3. Determine whether the operating system supports SIMD floating-point state management.
4. Optionally, determine whether the operating system supports unmasked exceptions, if such support is required by an application.

This application note describes each of these steps and includes sample functions in C++ showing how this detection can be done in an application. The appendix discusses using this code in a simple program.

2 Determining Whether the Processor Supports Streaming SIMD Extensions

An Invalid Opcode exception occurs when an application attempts to use the Streaming SIMD Extensions on an Intel processor that does not support it. To avoid this, an application can check the Intel architecture feature flags to determine whether the processor supports Streaming SIMD Extensions. An application can decide which version of code is most appropriate for the system's processor by inspecting the results of the CPUID instruction, which returns the feature flags in the EDX register.

The processor supports Streaming SIMD Extensions when bit 25 of the feature flags is set to 1. The following code segment loads the feature flags into `EDX` and tests the result for the support. It is recommended that applications first check to make sure that the processor is an Intel processor that has the `CPUID` instruction as described in application note, "AP-485, Intel Processor Identification with `CPUID` Instruction", (Order # 241618-008).

```
mov    eax, 1           ; request the processor feature flags from CPUID
CPUID                   ; CPUID loads the feature flags into edx
test   edx, 02000000h   ; test bit 25 for Streaming SIMD Extensions existence
                        ; if 1, Streaming SIMD Extensions is supported
```

The following code uses this assembly sequence in inlined-assembly as part of a C++ function. It first checks that the `CPUID` instruction is available, by using structured exception handling (a `try/except` clause), and that it is running on an Intel processor. The absence of processor support for the `CPUID` instruction results in an Invalid Opcode exception (a `STATUS_ILLEGAL_INSTRUCTION` exception code on Windows* 95, 98, and NT) that is handled in the `except` clause. If `CPUID` is available, it checks the `CPUID` vendor identification string to see if it is a `"GenuineIntel"` processor prior to

using the `CPUID` instruction to get the processor feature information. Please refer to AP-485 for more details on determining the processor features.

```
bool StreamingSIMDExtensionsHWSupport() {
    bool HWSupport = false;
    char brand[12];
    unsigned *str = (unsigned *) brand;

    // Does the processor have CPUID, and is it "GenuineIntel"?
    __try {
        _asm{
            mov     eax, 0     //First, check to make sure this is an Intel processor
            cpuid              //by getting the processor information string with CPUID
            mov     str, ebx   //  ebx contains "Genu"
            mov     str+4, edx //  edx contains "ineI"
            mov     str+8, ecx  //  ecx contains "ntel"  -- "GenuineIntel"
        }
    }
    __except(EXCEPTION_EXECUTE_HANDLER) {
        if (_exception_code() == STATUS_ILLEGAL_INSTRUCTION) {
            cout << endl << "****CPUID is not enabled****" << endl;
            return (false);
        }
        return (false); // If we get here, an unexpected exception occurred.
    }
    // Now make sure the processor is "GenuineIntel".
    if (!strncmp(brand, "GenuineIntel", 12)) {
        cout << endl << "****This is not an Intel processor!****" << endl;
        return (false);
    }
    // And finally, check the CPUID for Streaming SIMD Extensions support.
    _asm{
        mov     eax, 1         // Put a "1" in eax to tell CPUID to get the feature bits
        cpuid                  // Perform CPUID (puts processor feature info into EDX)
        test    edx, 02000000h // Test bit 25, for Streaming SIMD Extensions existence.
        jz      NotFound       // If not set, jump over the next instruction (No Streaming SIMD
        mov     [HWSupport],1  // Extensions).  Set return value to 1 to indicate,
                               // that the processor does support Streaming SIMD Extensions.
    NotFound:
    }
    return (HWSupport);
}
```

Code Example 1: Using `CPUID` to Detect Streaming SIMD Extensions Support

3 Determining Whether the Operating System Supports Streaming SIMD Extensions

Since Streaming SIMD Extensions introduces a new processor state (specifically, the Streaming SIMD Extensions floating-point, control, and status registers), the operating system in use on a system has to be able to manage saves and restores of this state at various times. In addition, the operating system must also support exception handling for unmasked SIMD floating-point exceptions where needed. This section describes how to detect whether the operating system in use has this support.

3.1 FXSAVE and FXRSTOR Support

Deschutes-based Pentium® II processors, and Pentium® III, introduce two state management instructions, FXSAVE and FXRSTOR. These instructions save the floating-point and MMX™ technology state, and the SIMD floating-point state to memory. The operating system uses these instructions to ensure that the application's state is properly saved and restored. If the operating system is enabled properly for SIMD floating-point state management, bit 9 of control register 4 (CR4) the OSFXSR bit, will be set to 1. This bit is set by the operating system if the following conditions are met:

- the processor supports FXSAVE/FXRSTOR instructions (the CPUID.FXSR bit is set)
- the operating system supports SIMD floating-point state management by using the FXSAVE/FXRSTOR instructions

Applications can successfully execute Streaming SIMD Extensions that use the extended register set (the new state) when the following two conditions are met:

- Floating point emulation (CR0.EM = 0) is disabled. This is necessary for MMX™ technology as well as Streaming SIMD Extensions.
- The operating system manages state using FXSAVE/FXSRTOR (CR4.OSFXSR = 1).

The easiest way to check for this operating system support is simply by executing an instruction using Streaming SIMD Extensions and catching an Invalid Opcode interrupt using structured exception handling in C++ as in the example below. This indirect method of detection is necessary since access to the control registers is allowed only in privileged ring0 of the operating system, preventing user applications from simply querying the CR0.EM and the CR4.OSFXSR bits. The following example demonstrates how to check for this support using C++ structured exception handling.

```
bool StreamingSIMDExtensionOSSupport()
{
   __try
   {
      _asm  xorps xmm0, xmm0  //Execute an instruction using Streaming
                              //SIMD Extensions to see if support exists.
   }
   //
   // Catch any exception.  If an Invalid Opcode exception (ILLEGAL_INSTRUCTION)
   // occurs, and you have already checked the XMM bit in CPUID, Streaming SIMD
   // Extensions are not supported by the OS.
   //
    __except(EXCEPTION_EXECUTE_HANDLER) {
      if (_exception_code() == STATUS_ILLEGAL_INSTRUCTION) {
         return (false);
      }
      // If we get here, an unknown exception occurred; investigation needed.
      return (false);
   }
   return (true);
}
```

Code Example 2: OS support for Streaming SIMD Extensions

Before checking for such operating system support, applications should first check that the processor supports Streaming SIMD Extensions in the manner described in Code Example 1. If Streaming SIMD Extensions are executed but is not supported by the processor, an Invalid Opcode exception will also be raised. The above example assumes that this has been done, so that any Invalid Opcode exception is due to the fact that the OS support for SIMD floating-point state management is absent.

3.2 Operating System Support For Unmasked Exceptions

This final check for operating system support of Streaming SIMD Extensions is optional. It is only recommended if an application must handle unmasked exceptions. Unmasking exceptions typically is only done in non-production code or when debugging applications to determine when exception conditions occur. When the exceptions occur and are masked, the processor automatically provides a "reasonable" value for the computation based on the IEEE standards.

If the processor detects a condition for an unmasked Streaming SIMD Extensions application exception, a software exception handler is invoked immediately at the end of the excepting instruction. If the exception is unmasked, and Streaming SIMD Extensions unmasked exceptions are not enabled (control register 4, bit 10 - OSXMMEXCPT), an Invalid Opcode fault occurs. If, however, the operating system

supports Streaming SIMD Extensions unmasked exception handling (the `OSXMMEXCEPT` bit is 1), a Streaming SIMD Extensions exception is raised, and the operating system can provide information to help find the cause of the fault. (The actual exception code name for the Streaming SIMD Extensions exception is STATUS_FLOAT_MULTIPLE_TRAPS and STATUS_FLOAT_MULTIPLE_FAULTS, with the `_exception_code()` routine for Windows' C++ structured exception handling also giving the integer value in Windows NT 5.0, beta version 2 and later. At the time of the creation of this application note, only Windows NT 5.0 beta version 2 and later offers this support.)

The operating system support for Streaming SIMD Extensions exception handling can be detected in the same manner as described in the code for Example 2. That is, execute a code sequence that should raise an exception and catch it. The difference here, though, is that since the processor and operating system mask the numeric exceptions by default, you must first unmask the one(s) in which you are interested prior to generating the exception.

In the example below, a Divide-by-Zero exception is raised as a result of using a Streaming SIMD Extensions packed divide with a zero denominator. If unmasked exceptions are not enabled by the operating system, an Invalid Opcode exception is raised. Otherwise, a Streaming SIMD Extensions exception is raised. In this latter case, using the `_exception_info()` routine supported by Windows® in the `except` clause can provide information to investigate the situation and take appropriate action. (Use of this routine is not shown here, though. Further information is available through the Microsoft Developer's Studio help information on structured exception handling.)

```
bool StreamingSIMDExtensionOSException()
{
    bool StreamingSIMDExtensionOSExcept = true;
    unsigned long csr;
    float x[4] = {1.0f, 2.0f, 3.0f, 4.0f};

    __try
    {
        _asm{
            stmxcsr [csr]      //Get the MXCSR (Streaming SIMD Extensions control register)
            and     [csr], 0FFFFFDFFh // Set the divide-by-zero mask bit (bit 9) to 0 to
            ldmxcsr [csr]      // unmask this exception, then reload the MXCSR.

            xorps   xmm0, xmm0 //Fill the denominator register with 0's
            movups  xmm1,[x]   //Do a divide by zero using Streaming SIMD Extensions
            divps   xmm1, xmm0 //  packed divide.  The interrupt handler should be invoked.
        }
    }
    //
    // Catch any exception that occurs. The exception handler here needs to take different
    // actions depending on the OS.  If you have determined that your processor and OS
    // indeed support Streaming SIMD Extensions, an ILLEGAL_INSTRUCTION exception
    // indicates that unmasked SIMD floating-point exceptions are not supported.
    //
    __except(EXCEPTION_EXECUTE_HANDLER) {
        unsigned long code = _exception_code();
        if (code == STATUS_ILLEGAL_INSTRUCTION) {
            cout << endl << "****OS did not handle the exception!****" << endl;
            StreamingSIMDExtensionOSExcept = false;
        }
        else {
            cout << endl << "****OS handled the exception.****" << endl;
            StreamingSIMDExtensionOSExcept = true;
            // But you would need to have special handling code here to decipher what
            // exception really occurred and why!
        }
    }
    return (StreamingSIMDExtensionOSExcept);
}
```

Code Example 3: Determining Support for Un-masked SIMD floating-point Exceptions

4 Summary

To check for complete support for Streaming SIMD Extensions on the system on which an application is running, both the processor and the operating system features must be queried. There are four steps to do this:

1) Check whether the processor is an Intel processor using `CPUID`.
2) Check the `CPUID` feature flags for Streaming SIMD Extensions support.
3) Determine whether the operating system support SIMD floating-point state management.
4) Determine whether the operating system supports unmasked SIMD floating-point exceptions, if such support is required by an application.

If all of these conditions are met, Streaming SIMD Extensions is fully enabled on your system.

Appendix: CpuOSId program

The sample program in Samples/CpuOSId/CpuOSId.cpp on this CD looks for the four conditions described above: a genuine Intel processor, the processor supporting Streaming SIMD Extensions, and the operating system supporting state management and unmaskable exceptions for Streaming SIMD Extensions.

The example program, has been tested on the following systems:

- Pentium® Pro and Pentium II processors with Windows NT 4.0 Service Pack 3
- Pentium® processor with Windows 95
- Processor with support for Streaming SIMD Extensions with Windows 98
- Processor with support for Streaming SIMD Extensions with Windows NT 5.0 (beta version 2)
- Processor with support for Streaming SIMD Extensions with Windows NT 4.0, Service Pack 4
- Processor with support for Streaming SIMD Extensions with Windows 95

The program uses the example functions described in this application note to answer the three questions:

```
"Does your processor support Streaming SIMD Extensions? "
"Does the OS support save/restore for SIMD floating-point state? "
"Does the OS support SIMD floating-point exceptions? "
```

and answers each with "YES!!!" or "No.".

Only systems with Streaming SIMD Extensions support running NT 5.0, beta version 2 and higher, will answer "YES!!!" to all three questions. The processor with Streaming SIMD Extensions support and Windows 98 lacks only the unmasked exception support.