Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

November 2007

Notice: The Intel® 64 and IA-32 architectures may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Current characterized errata are documented in the specification updates.

Document Number: 252046-021

Contents

Revision History

Revision	Description	Date
-001	• Initial release	November 2002
-002	• Added 1-10 Documentation Changes. • Removed old Documentation Changes items that already have been incorporated in the published Software Developer's manual	December 2002
-003	• Added 9 -17 Documentation Changes. • Removed Documentation Change #6 - References to bits Gen and Len Deleted. • Removed Documentation Change #4 - VIF Information Added to CLI Discussion	February 2003
-004	• Removed Documentation changes 1-17. • Added Documentation changes 1-24.	June 2003
-005	• Removed Documentation Changes 1-24. • Added Documentation Changes 1-15.	September 2003
-006	• Added Documentation Changes 16- 34.	November 2003
-007	• Updated Documentation changes 14, 16, 17, and 28. • Added Documentation Changes 35-45.	January 2004
-008	• Removed Documentation Changes 1-45. • Added Documentation Changes 1-5.	March 2004
-009	• Added Documentation Changes 7-27.	May 2004
-010	• Removed Documentation Changes 1-27. • Added Documentation Changes 1.	August 2004
-011	• Added Documentation Changes 2-28.	November 2004
-012	• Removed Documentation Changes 1-28. • Added Documentation Changes 1-16.	March 2005
-013	• Updated title. • There are no Documentation Changes for this revision of the document.	July 2005
-014	• Added Documentation Changes 1-21.	September 2005
-015	• Removed Documentation Changes 1-21. • Added Documentation Changes 1-20.	March 9, 2006
-016	• Added Documentation changes 21-23.	March 27, 2006
-017	• Removed Documentation Changes 1-23. • Added Documentation Changes 1-36.	September 2006
-018	• Added Documentation Changes 37-42.	October 2006
-019	• Removed Documentation Changes 1-42. • Added Documentation Changes 1-19.	March 2007
-020	• Added Documentation Changes 20-27.	May 2007
-021	• Removed Documentation Changes 1-27. • Added Documentation Changes 1-6	November 2007

Preface

This document is an update to the specifications contained in the Affected Documents table below. This document is a compilation of device and documentation errata, specification clarifications and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

Affected Documents

Document Title	Document Number/Location
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture	253665
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M	253666
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z	253667
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide.	253668
Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide	253669

Nomenclature

Documentation Changes include typos, errors, or omissions from the current published specifications. These will be incorporated in any new release of the specification.

Summary Tables of Changes

The following table indicates documentation changes which apply to the Intel® 64 and IA-32 architectures. This table uses the following notations:

Codes Used in Summary Tables

Change bar to left of table row indicates this erratum is either new or modified from the previous version of the document.

Documentation Changes

No.	DOCUMENTATION CHANGES
1	System Architecture Overview (Chapter 2, Vol 3A)
2	Interrupt and Exception Handling (Chapter 5, Vol 3A)
3	System Programming For Streaming SIMD Instruction Sets (Chapter 12, Vol 3A)
4	Power and Thermal Management (Chapter 13, Vol 3A)
5	Machine-Check Architecture (Chapter 14, Vol 3A)
6	Debugging and Performance Monitoring (Chapter 18, Vol 3B)

Documentation Changes

1. System Architecture Overview (Chapter 2, Vol 3A)

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide*.

2.5 CONTROL REGISTERS

Control registers (CR0, CR1, CR2, CR3, and CR4; see Figure 2-6) determine operating mode of the processor and the characteristics of the currently executing task. These registers are 32 bits in all 32-bit modes and compatibility mode.

In 64-bit mode, control registers are expanded to 64 bits. The MOV CRn instructions are used to manipulate the register bits. Operand-size prefixes for these instructions are ignored. The following is also true:

- Bits 63:32 of CR0 and CR4 are reserved and must be written with zeros. Writing a nonzero value to any of the upper 32 bits results in a general-protection exception, #GP(0).
- All 64 bits of CR2 are writable by software.
- Bits 51:40 of CR3 are reserved and must be 0.
- The MOV CRn instructions do not check that addresses written to CR2 and CR3 are within the linear-address or physical-address limitations of the implementation.
- Register CR8 is available in 64-bit mode only.

The control registers are summarized below, and each architecturally defined control field in these control registers are described individually. In Figure 2-6, the width of the register in 64-bit mode is indicated in parenthesis (except for CR0).

- **CR0** — Contains system control flags that control operating mode and states of the processor.
- **CR1** — Reserved.
- **CR2** — Contains the page-fault linear address (the linear address that caused a page fault).
- **CR3** — Contains the physical address of the base of the page directory and two flags (PCD and PWT). This register is also known as the page-directory base register (PDBR). Only the most-significant bits (less the lower 12 bits) of the base address are specified; the lower 12 bits of the address are assumed to be 0. The page directory must thus be aligned to a page (4-KByte) boundary. The PCD and PWT flags control caching of the page directory in the processor's internal data caches (they do not control TLB caching of page-directory information).

  When using the physical address extension, the CR3 register contains the base address of the page-directory-pointer table In IA-32e mode, the CR3 register contains the base address of the PML4 table.

  See also: Section 3.8, "36-Bit Physical Addressing Using the PAE Paging Mechanism."
- **CR4** — Contains a group of flags that enable several architectural extensions, and indicate operating system or executive support for specific processor capabilities.

The control registers can be read and loaded (or modified) using the move-to-or-from-control-registers forms of the MOV instruction. In protected mode, the MOV instructions allow the control registers to be read or loaded (at privilege level 0 only). This restriction means that application programs or operating-system procedures (running at privilege levels 1, 2, or 3) are prevented from reading or loading the control registers.

- **CR8** — Provides read and write access to the Task Priority Register (TPR). It specifies the priority threshold value that operating systems use to control the priority class of external interrupts allowed to interrupt the processor. This register is available only in 64-bit mode. However, interrupt filtering continues to apply in compatibility mode.

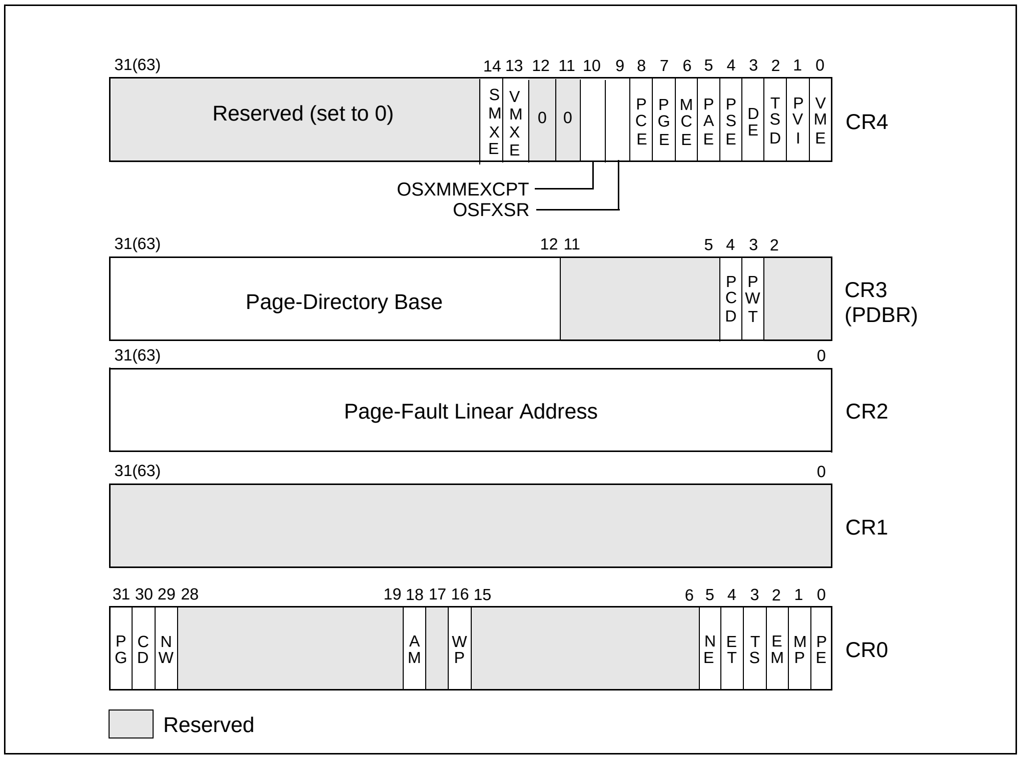


Figure 2-6. Control Registers

When loading a control register, reserved bits should always be set to the values previously read. The flags in control registers are:

PG	**Paging (bit 31 of CR0)** — Enables paging when set; disables paging when clear. When paging is disabled, all linear addresses are treated as physical addresses. The PG flag has no effect if the PE flag (bit 0 of register CR0) is not also set; setting the PG flag when the PE flag is clear causes a general-protection exception (#GP). See also: Section 3.6, “Paging (Virtual Memory) Overview.” On Intel 64 processors, enabling and disabling IA-32e mode operation also requires modifying CR0.PG.
CD	**Cache Disable (bit 30 of CR0)** — When the CD and NW flags are clear, caching of memory locations for the whole of physical memory in the processor’s internal (and external) caches is enabled. When the CD flag is set, caching is restricted as described in Table 10-5. To prevent the processor from accessing and

updating its caches, the CD flag must be set and the caches must be invalidated so that no cache hits can occur.

See also: Section 10.5.3, “Preventing Caching,” and Section 10.5, “Cache Control.”

NW **Not Write-through (bit 29 of CR0)** — When the NW and CD flags are clear, write-back (for Pentium 4, Intel Xeon, P6 family, and Pentium processors) or write-through (for Intel486 processors) is enabled for writes that hit the cache and invalidation cycles are enabled. See Table 10-5 for detailed information about the affect of the NW flag on caching for other settings of the CD and NW flags.

AM **Alignment Mask (bit 18 of CR0)** — Enables automatic alignment checking when set; disables alignment checking when clear. Alignment checking is performed only when the AM flag is set, the AC flag in the EFLAGS register is set, CPL is 3, and the processor is operating in either protected or virtual-8086 mode.

WP **Write Protect (bit 16 of CR0)** — Inhibits supervisor-level procedures from writing into user-level read-only pages when set; allows supervisor-level procedures to write into user-level read-only pages when clear (regardless of the U/S bit setting; see Section 3.7.6). This flag facilitates implementation of the copy-on-write method of creating a new process (forking) used by operating systems such as UNIX.

NE **Numeric Error (bit 5 of CR0)** — Enables the native (internal) mechanism for reporting x87 FPU errors when set; enables the PC-style x87 FPU error reporting mechanism when clear. When the NE flag is clear and the IGNNE# input is asserted, x87 FPU errors are ignored. When the NE flag is clear and the IGNNE# input is deasserted, an unmasked x87 FPU error causes the processor to assert the FERR# pin to generate an external interrupt and to stop instruction execution immediately before executing the next waiting floating-point instruction or WAIT/FWAIT instruction.

The FERR# pin is intended to drive an input to an external interrupt controller (the FERR# pin emulates the ERROR# pin of the Intel 287 and Intel 387 DX math coprocessors). The NE flag, IGNNE# pin, and FERR# pin are used with external logic to implement PC-style error reporting.

See also: “Software Exception Handling” in Chapter 8, “Programming with the x87 FPU,” and Appendix A, “Eflags Cross-Reference,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.

ET **Extension Type (bit 4 of CR0)** — Reserved in the Pentium 4, Intel Xeon, P6 family, and Pentium processors. In the Pentium 4, Intel Xeon, and P6 family processors, this flag is hardcoded to 1. In the Intel386 and Intel486 processors, this flag indicates support of Intel 387 DX math coprocessor instructions when set.

TS **Task Switched (bit 3 of CR0)** — Allows the saving of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context on a task switch to be delayed until an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction is actually executed by the new task. The processor sets this flag on every task switch and tests it when executing x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

- If the TS flag is set and the EM flag (bit 2 of CR0) is clear, a device-not-available exception (#NM) is raised prior to the execution of any x87 FPU/MMX/SSE/ SSE2/SSE3/SSSE3/SSE4 instruction; with the exception of PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. See the paragraph below for the special case of the WAIT/FWAIT instructions.

- If the TS flag is set and the MP flag (bit 1 of CR0) and EM flag are clear, an #NM exception is not raised prior to the execution of an x87 FPU WAIT/FWAIT instruction.
- If the EM flag is set, the setting of the TS flag has no affect on the execution of x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

Table 2-1 shows the actions taken when the processor encounters an x87 FPU instruction based on the settings of the TS, EM, and MP flags. Table 11-1 and 12-1 show the actions taken when the processor encounters an MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction.

The processor does not automatically save the context of the x87 FPU, XMM, and MXCSR registers on a task switch. Instead, it sets the TS flag, which causes the processor to raise an #NM exception whenever it encounters an x87 FPU/MMX/SSE /SSE2/SSE3/SSSE3/SSE4 instruction in the instruction stream for the new task (with the exception of the instructions listed above).

The fault handler for the #NM exception can then be used to clear the TS flag (with the CLTS instruction) and save the context of the x87 FPU, XMM, and MXCSR registers. If the task never encounters an x87 FPU/MMX/SSE/SSE2/SSE3//SSSE3/SSE4 instruction; the x87 FPU/MMX/SSE/SSE2/ SSE3/SSSE3/SSE4 context is never saved.

Table 2-1. Action Taken By x87 FPU Instructions for Different Combinations of EM, MP, and TS

CR0 Flags			x87 FPU Instruction Type	
EM	MP	TS	Floating-Point	WAIT/FWAIT
0	0	0	Execute	Execute.
0	0	1	#NM Exception	Execute.
0	1	0	Execute	Execute.
0	1	1	#NM Exception	#NM exception.
1	0	0	#NM Exception	Execute.
1	0	1	#NM Exception	Execute.
1	1	0	#NM Exception	Execute.
1	1	1	#NM Exception	#NM exception.

EM **Emulation (bit 2 of CR0)** — Indicates that the processor does not have an internal or external x87 FPU when set; indicates an x87 FPU is present when clear. This flag also affects the execution of MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

When the EM flag is set, execution of an x87 FPU instruction generates a device-not-available exception (#NM). This flag must be set when the processor does not have an internal x87 FPU or is not connected to an external math coprocessor. Setting this flag forces all floating-point instructions to be handled by software emulation. Table 9-2 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-1 shows the interaction of the EM, MP, and TS flags.

Also, when the EM flag is set, execution of an MMX instruction causes an invalid-opcode exception (#UD) to be generated (see Table 11-1). Thus, if an IA-32 or Intel 64 processor incorporates MMX technology, the EM flag must be set to 0 to enable execution of MMX instructions.

Similarly for SSE/SSE2/SSE3/SSSE3/SSE4 extensions, when the EM flag is set, execution of most SSE/SSE2/SSE3/SSSE3/SSE4 instructions causes an invalid opcode exception (#UD) to be generated (see Table 12-1). If an IA-32 or Intel 64 processor incorporates the SSE/SSE2/SSE3/SSSE3/SSE4 extensions, the EM flag must be set to 0 to enable execution of these extensions. SSE/SSE2/SSE3/SSSE3/SSE4 instructions not affected by the EM flag include: PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

MP **Monitor Coprocessor (bit 1 of CR0).** — Controls the interaction of the WAIT (or FWAIT) instruction with the TS flag (bit 3 of CR0). If the MP flag is set, a WAIT instruction generates a device-not-available exception (#NM) if the TS flag is also set. If the MP flag is clear, the WAIT instruction ignores the setting of the TS flag. Table 9-2 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-1 shows the interaction of the MP, EM, and TS flags.

PE **Protection Enable (bit 0 of CR0)** — Enables protected mode when set; enables real-address mode when clear. This flag does not enable paging directly. It only enables segment-level protection. To enable paging, both the PE and PG flags must be set.

See also: Section 9.9, “Mode Switching.”

PCD **Page-level Cache Disable (bit 4 of CR3)** — Controls caching of the current page directory. When the PCD flag is set, caching of the page-directory is prevented; when the flag is clear, the page-directory can be cached. This flag affects only the processor’s internal caches (both L1 and L2, when present). The processor ignores this flag if paging is not used (the PG flag in register CR0 is clear) or the CD (cache disable) flag in CR0 is set.

See also: Chapter 10, “Memory Cache Control” (for more about the use of the PCD flag) and Section 3.7.6, “Page-Directory and Page-Table Entries” (for a description of a companion PCD flag in page-directory and page-table entries).

PWT **Page-level Writes Transparent (bit 3 of CR3)** — Controls the write-through or write-back caching policy of the current page directory. When the PWT flag is set, write-through caching is enabled; when the flag is clear, write-back caching is enabled. This flag affects only internal caches (both L1 and L2, when present). The processor ignores this flag if paging is not used (the PG flag in register CR0 is clear) or the CD (cache disable) flag in CR0 is set.

See also: Section 10.5, “Cache Control” (for more information about the use of this flag), and Section 3.7.6, “Page-Directory and Page-Table Entries” (for a description of a companion PCD flag in the page-directory and page-table entries).

VME **Virtual-8086 Mode Extensions (bit 0 of CR4)** — Enables interrupt- and exception-handling extensions in virtual-8086 mode when set; disables the extensions when clear. Use of the virtual mode extensions can improve the performance of virtual-8086 applications by eliminating the overhead of calling the virtual-8086 monitor to handle interrupts and exceptions that occur while executing an 8086 program and, instead, redirecting the interrupts and exceptions back to the 8086 program’s handlers. It also provides hardware support for a virtual interrupt flag (VIF) to improve reliability of running 8086 programs in multitasking and multiple-processor environments.

See also: Section 15.3, “Interrupt and Exception Handling in Virtual-8086 Mode.”

PVI **Protected-Mode Virtual Interrupts (bit 1 of CR4)** — Enables hardware support for a virtual interrupt flag (VIF) in protected mode when set; disables the VIF flag in protected mode when clear.

See also: Section 15.4, “Protected-Mode Virtual Interrupts.”

TSD **Time Stamp Disable (bit 2 of CR4)** — Restricts the execution of the RDTSC instruction to procedures running at privilege level 0 when set; allows RDTSC instruction to be executed at any privilege level when clear.

DE **Debugging Extensions (bit 3 of CR4)** — References to debug registers DR4 and DR5 cause an undefined opcode (#UD) exception to be generated when set; when clear, processor aliases references to registers DR4 and DR5 for compatibility with software written to run on earlier IA-32 processors.

See also: Section 18.2.2, “Debug Registers DR4 and DR5.”

PSE **Page Size Extensions (bit 4 of CR4)** — Enables large page sizes (2 or 4-MByte pages) when set; restricts pages to 4 KBytes when clear.

See also: Section 3.6.1, “Paging Options.”

PAE **Physical Address Extension (bit 5 of CR4)** — When set, enables paging mechanism to reference greater-or-equal-than-36-bit physical addresses. When clear, restricts physical addresses to 32 bits. PAE must be enabled to enable IA-32e mode operation. Enabling and disabling IA-32e mode operation also requires modifying CR4.PAE.

See also: Section 3.8, “36-Bit Physical Addressing Using the PAE Paging Mechanism.”

MCE **Machine-Check Enable (bit 6 of CR4)** — Enables the machine-check exception when set; disables the machine-check exception when clear.

See also: Chapter 14, “Machine-Check Architecture.”

PGE **Page Global Enable (bit 7 of CR4)** — (Introduced in the P6 family processors.) Enables the global page feature when set; disables the global page feature when clear. The global page feature allows frequently used or shared pages to be marked as global to all users (done with the global flag, bit 8, in a page-directory or page-table entry). Global pages are not flushed from the translation-lookaside buffer (TLB) on a task switch or a write to register CR3.

When enabling the global page feature, paging must be enabled (by setting the PG flag in control register CR0) before the PGE flag is set. Reversing this sequence may affect program correctness, and processor performance will be impacted.

See also: Section 3.12, “Translation Lookaside Buffers (TLBs).”

PCE **Performance-Monitoring Counter Enable (bit 8 of CR4)** — Enables execution of the RDPMC instruction for programs or procedures running at any protection level when set; RDPMC instruction can be executed only at protection level 0 when clear.

OSFXSR

Operating System Support for FXSAVE and FXRSTOR instructions (bit 9 of CR4) — When set, this flag: (1) indicates to software that the operating system supports the use of the FXSAVE and FXRSTOR instructions, (2) enables the FXSAVE and FXRSTOR instructions to save and restore the contents of the XMM and MXCSR registers along with the contents of the x87 FPU and MMX registers, and (3) enables the processor to execute SSE/SSE2/SSE3/SSSE3/SSE4 instructions, with the exception of the PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

If this flag is clear, the FXSAVE and FXRSTOR instructions will save and restore the contents of the x87 FPU and MMX instructions, but they may not save and restore the contents of the XMM and MXCSR registers. Also, the processor will generate an invalid opcode exception (#UD) if it attempts to execute any SSE/SSE2/SSE3/SSSE3/SSE4, with the exception of PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. The operating system or executive must explicitly set this flag.

NOTE

CPUID feature flags FXSR indicates availability of the FXSAVE/FXRESTOR instructions. The OSFXSR bit provides operating system software with a means of enabling FXSAVE/FXRESTOR to save/restore the contents of the X87 FPU, XMM and MXCSR registers. Consequently OSFXSR bit indicates that the operating system provides context switch support for SSE/SSE2/SSE3/SSSE3/SSE4.

OSXMMEXCPT

Operating System Support for Unmasked SIMD Floating-Point Exceptions (bit 10 of CR4) — When set, indicates that the operating system supports the handling of unmasked SIMD floating-point exceptions through an exception handler that is invoked when a SIMD floating-point exception (#XF) is generated. SIMD floating-point exceptions are only generated by SSE/SSE2/SSE3/SSE4.1 SIMD floating-point instructions.

The operating system or executive must explicitly set this flag. If this flag is not set, the processor will generate an invalid opcode exception (#UD) whenever it detects an unmasked SIMD floating-point exception.

VMXE

VMX-Enable Bit (bit 13 of CR4) — Enables VMX operation when set. See Chapter 19, "Introduction to Virtual-Machine Extensions."

SMXE

SMX-Enable Bit (bit 14 of CR4) — Enables SMX operation when set. See Chapter 6, "Safer Mode Extensions Reference" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

TPL **Task Priority Level (bit 3:0 of CR8)** — This sets the threshold value corresponding to the highest-priority interrupt to be blocked. A value of 0 means all interrupts are enabled. This field is available in 64-bit mode. A value of 15 means all interrupts will be disabled.

2. Interrupt and Exception Handling (Chapter 5, Vol 3A)

Change bars show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide*.

5.3.1 External Interrupts

External interrupts are received through pins on the processor or through the local APIC. The primary interrupt pins on Pentium 4, Intel Xeon, P6 family, and Pentium processors are the LINT[1:0] pins, which are connected to the local APIC (see Chapter 8, "Advanced Programmable Interrupt Controller (APIC)"). When the local APIC is enabled, the LINT[1:0] pins can be programmed through the APIC's local vector table (LVT) to be associated with any of the processor's exception or interrupt vectors.

When the local APIC is global/hardware disabled, these pins are configured as INTR and NMI pins, respectively. Asserting the INTR pin signals the processor that an external interrupt has occurred. The processor reads from the system bus the interrupt vector number provided by an external interrupt controller, such as an 8259A (see Section 5.2, "Exception and Interrupt Vectors"). Asserting the NMI pin signals a non-maskable interrupt (NMI), which is assigned to interrupt vector 2.

Table 5-1. Protected-Mode Exceptions and Interrupts

	Mne-monic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	RESERVED	Fault/ Trap	No	For Intel use only.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT 3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD2 instruction or reserved opcode.[1]
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction.[2]
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.

Table 5-1. Protected-Mode Exceptions and Interrupts (Continued)

14	#PF	Page Fault	Fault	Yes	Any memory reference.
15	—	(Intel reserved. Do not use.)		No	
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory.[3]
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent.[4]
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3/SSE4.1 floating-point instructions[5]
20-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Non-reserved) Interrupts	Interrupt		External interrupt or INT *n* instruction.

NOTES:

1. The UD2 instruction was introduced in the Pentium Pro processor.
2. Processors after the Intel386 processor do not generate this exception.
3. This exception was introduced in the Intel486 processor.
4. This exception was introduced in the Pentium processor and enhanced in the P6 family processors.
5. This exception was introduced in the Pentium III processor.

The processor’s local APIC is normally connected to a system-based I/O APIC. Here, external interrupts received at the I/O APIC’s pins can be directed to the local APIC through the system bus (Pentium 4 and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors). The I/O APIC determines the vector number of the interrupt and sends this number to the local APIC. When a system contains multiple processors, processors can also send interrupts to one another by means of the system bus (Pentium 4 and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors).

The LINT[1:0] pins are not available on the Intel486 processor and earlier Pentium processors that do not contain an on-chip local APIC. These processors have dedicated NMI and INTR pins. With these processors, external interrupts are typically generated by a system-based interrupt controller (8259A), with the interrupts being signaled through the INTR pin.

Note that several other pins on the processor can cause a processor interrupt to occur. However, these interrupts are not handled by the interrupt and exception mechanism described in this chapter. These pins include the RESET#, FLUSH#, STPCLK#, SMI#, R/S#, and INIT# pins. Whether they are included on a particular processor is implementation dependent. Pin functions are described in the data books for the individual processors. The SMI# pin is described in Chapter 24, “System Management.”

...

5.15 EXCEPTION AND INTERRUPT REFERENCE

...

Interrupt 16—x87 FPU Floating-Point Error (#MF)

Exception Class Fault.

Description

Indicates that the x87 FPU has detected a floating-point error. The NE flag in the register CR0 must be set for an interrupt 16 (floating-point error exception) to be generated. (See Section 2.5, "Control Registers," for a detailed description of the NE flag.)

NOTE

SIMD floating-point exceptions (#XM) are signaled through interrupt 19.

While executing x87 FPU instructions, the x87 FPU detects and reports six types of floating-point error conditions:

- Invalid operation (#I)
  - Stack overflow or underflow (#IS)
  - Invalid arithmetic operation (#IA)
- Divide-by-zero (#Z)
- Denormalized operand (#D)
- Numeric overflow (#O)
- Numeric underflow (#U)
- Inexact result (precision) (#P)

Each of these error conditions represents an x87 FPU exception type, and for each of exception type, the x87 FPU provides a flag in the x87 FPU status register and a mask bit in the x87 FPU control register. If the x87 FPU detects a floating-point error and the mask bit for the exception type is set, the x87 FPU handles the exception automatically by generating a predefined (default) response and continuing program execution. The default responses have been designed to provide a reasonable result for most floating-point applications.

If the mask for the exception is clear and the NE flag in register CR0 is set, the x87 FPU does the following:

1. Sets the necessary flag in the FPU status register.
2. Waits until the next "waiting" x87 FPU instruction or WAIT/FWAIT instruction is encountered in the program's instruction stream.
3. Generates an internal error signal that cause the processor to generate a floating-point exception (#MF).

Interrupt 19—SIMD Floating-Point Exception (#XM)

Exception Class Fault.

Description

Indicates the processor has detected an SSE/SSE2/SSE3/SSE4.1 SIMD floating-point exception. The appropriate status flag in the MXCSR register must be set and the particular exception unmasked for this interrupt to be generated.

There are six classes of numeric exception conditions that can occur while executing an SSE/SSE2/SSE3/SSE4.1 SIMD floating-point instruction:

- Invalid operation (#I)
- Divide-by-zero (#Z)
- Denormal operand (#D)
- Numeric overflow (#O)
- Numeric underflow (#U)
- Inexact result (Precision) (#P)

The invalid operation, divide-by-zero, and denormal-operand exceptions are pre-computation exceptions; that is, they are detected before any arithmetic operation occurs. The numeric underflow, numeric overflow, and inexact result exceptions are post-computational exceptions.

See "SIMD Floating-Point Exceptions" in Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, for additional information about the SIMD floating-point exception classes.

When a SIMD floating-point exception occurs, the processor does either of the following things:

- It handles the exception automatically by producing the most reasonable result and allowing program execution to continue undisturbed. This is the response to masked exceptions.
- It generates a SIMD floating-point exception, which in turn invokes a software exception handler. This is the response to unmasked exceptions.

Each of the six SIMD floating-point exception conditions has a corresponding flag bit and mask bit in the MXCSR register. If an exception is masked (the corresponding mask bit in the MXCSR register is set), the processor takes an appropriate automatic default action and continues with the computation. If the exception is unmasked (the corresponding mask bit is clear) and the operating system supports SIMD floating-point exceptions (the OSXMMEXCPT flag in control register CR4 is set), a software exception handler is invoked through a SIMD floating-point exception. If the exception is unmasked and the OSXMMEXCPT bit is clear (indicating that the operating system does not support unmasked SIMD floating-point exceptions), an invalid opcode exception (#UD) is signaled instead of a SIMD floating-point exception.

3. System Programming For Streaming SIMD Instruction Sets (Chapter 12, Vol 3A)

Change bars show changes to Chapter 12 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide*.

This chapter describes features of the streaming SIMD extensions (SSE), streaming SIMD extensions 2 (SSE2), streaming SIMD extensions 3 (SSE3), Supplemental SSE3 (SSSE3), and streaming SIMD extensions 4 (SSE4). These must be considered when designing or enhancing an operating system to support Intel 64 and IA-32 processors.

The chapter covers enabling SSE/SSE2/SSE3/SSSE3/SSE4 extensions, providing operating system or executive support for the SSE/SSE2/SSE3/SSSE3/SSE4 extensions, SIMD floating-point exceptions, exception handling, and task (context) switching.

12.1 PROVIDING OPERATING SYSTEM SUPPORT FOR SSE/SSE2/SSE3/SSSE3/SSE4 EXTENSIONS

To use SSE/SSE2/SSE3/SSSE3/SSE4 extensions, the operating system or executive must provide support for initializing the processor to use these extensions, for handling the FXSAVE and FXRSTOR state saving instructions, and for handling SIMD floating-point exceptions. The following sections provide system programming guidelines for this support. Because SSE/SSE2/SSE3/SSSE3/SSE4 extensions share the same state, experience the same sets of non-numerical and numerical exception behavior, these guidelines that apply to SSE also apply to other sets of SIMD extensions that operate on the same processor state and subject to the same sets of of non-numerical and numerical exception behavior.

Chapter 11, "Programming with Streaming SIMD Extensions 2 (SSE2)," and Chapter 12, "Programming with SSE3, SSSE3 and SSE4," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, discuss support for SSE/SSE2/SSE3/SSSE3/SSE4 from an applications point of view program.

12.1.1 Adding Support to an Operating System for SSE/SSE2/SSE3/SSSE3/SSE4 Extensions

The following guidelines describe functions that an operating system or executive must perform to support SSE/SSE2/SSE3/SSSE3/SSE4 extensions:

1. Check that the processor supports the SSE/SSE2/SSE3/SSSE3/SSE4 extensions.
2. Check that the processor supports the FXSAVE and FXRESTOR instructions.
3. Provide an initialization for the SSE, SSE2 SSE3, SSSE3 and SSE4 states.
4. Provide support for the FXSAVE and FXRSTOR instructions.
5. Provide support (if necessary) in non-numeric exception handlers for exceptions generated by the SSE, SSE2, SSE3 and SSE4 instructions.
6. Provide an exception handler for the SIMD floating-point exception (#XM).

The following sections describe how to implement each of these guidelines.

12.1.2 Checking for SSE/SSE2/SSE3/SSSE3/SSE4 Extension Support

If the processor attempts to execute an unsupported SSE/SSE2/SSE3/SSSE3/SSE4 instruction, the processor generates an invalid-opcode exception (#UD).

Before an operating system or executive attempts to use SSE/SSE2/SSE3/SSSE3/SSE4 extensions, it should check that support is present. Make sure:

- CPUID.1:EDX.SSE[bit 25] = 1
- CPUID.1:EDX.SSE2[bit 26] = 1
- CPUID.1:ECX.SSE3[bit 0] = 1
- CPUID.1:ECX.SSSE3[bit 9] = 1
- CPUID.1:ECX.SSE4_1[bit 19] = 1
- CPUID.1:ECX.SSE4_2[bit 20] = 1

To use POPCNT instruction, software must check CPUID.1:ECX.POPCNT[bit 23] = 1

12.1.3 Checking for Support for the FXSAVE and FXRSTOR Instructions

A separate check must be made to insure that the processor supports FXSAVE and FXRSTOR. Make sure:

- CPUID.1:EDX.FXSR[bit 24] = 1

12.1.4 Initialization of the SSE/SSE2/SSE3/SSSE3/SSE4 Extensions

The operating system or executive should carry out the following steps to set up SSE/SSE2/SSE3/SSSE3/SSE4 extensions for use by application programs:

1. Set CR4.OSFXSR[bit 9] = 1. Setting this flag assumes that the operating system provides facilities for saving and restoring SSE/SSE2/SSE3/SSSE3/SSE4 states using FXSAVE and FXRSTOR instructions. These instructions are commonly used to save the SSE/SSE2/SSE3/SSSE3/SSE4 state during task switches and when invoking the SIMD floating-point exception (#XM) handler (see Section 12.4, “Saving the SSE/SSE2/SSE3/SSSE3/SSE4 State on Task or Context Switches,” and Section 12.1.6, “Providing an Handler for the SIMD Floating-Point Exception (#XM),” respectively).

   If the processor does not support the FXSAVE and FXRSTOR instructions, attempting to set the OSFXSR flag will cause an exception (#GP) to be generated.

2. Set CR4.OSXMMEXCPT[bit 10] = 1. Setting this flag assumes that the operating system provides an SIMD floating-point exception (#XM) handler (see Section 12.1.6, “Providing an Handler for the SIMD Floating-Point Exception (#XM)”).

   NOTE

   The OSFXSR and OSXMMEXCPT bits in control register CR4 must be set by the operating system. The processor has no other way of detecting operating-system support for the FXSAVE and FXRSTOR instructions or for handling SIMD floating-point exceptions.

3. Clear CR0.EM[bit 2] = 0. This action disables emulation of the x87 FPU, which is required when executing SSE/SSE2/SSE3/SSSE3/SSE4 instructions (see Section 2.5, “Control Registers”).

4. Set CR0.MP[bit 1] = 1. This setting is the required setting for Intel 64 and IA-32 processors that support the SSE/SSE2/SSE3/SSSE3/SSE4 extensions (see Section 9.2.1, “Configuring the x87 FPU Environment”).

Table 12-1 and Table 12-2 show the actions of the processor when an SSE/SSE2/SSE3/SSSE3/SSE4 instruction is executed, depending on the:

- OSFXSR and OSXMMEXCPT flags in control register CR4
- SSE/SSE2/SSE3/SSSE3/SSE4 feature flags returned by CPUID
- EM, MP, and TS flags in control register CR0

Table 12-1. Action Taken for Combinations of OSFXSR, OSXMMEXCPT, SSE, SSE2, SSE3, SSE4 EM, MP, and TS[1]

CR4		CPUID	CR0 Flags			Action
OSFXSR	OSXMMEXCPT	SSE, SSE2, SSE3[2] SSE4_1[3]	EM	MP [4]	TS	
0	X[5]	X	X	1	X	#UD exception.
1	X	0	X	1	X	#UD exception.
1	X	1	1	1	X	#UD exception.
1	0	1	0	1	0	Execute instruction; #UD exception if unmasked SIMD floating-point exception is detected.
1	1	1	0	1	0	Execute instruction; #XM exception if unmasked SIMD floating-point exception is detected.
1	X	1	0	1	1	#NM exception.

NOTES:

1. For execution of any SSE/SSE2/SSE3/SSE4 instruction except the PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, MOVNTI, and CLFLUSH instructions.
2. Exception conditions due to CR4.OSFXSR or CR4.OSXMMEXCPT do not apply to FISTTP.
3. Only applies to DPPS, DPPD, ROUNDPS, ROUNDPD, ROUNDSS, ROUNDSD.
4. For processors that support the MMX instructions, the MP flag should be set.
5. X — Don’t care.

Table 12-2. Action Taken for Combinations of OSFXSR, SSSE3, SSE4, EM, and TS

CR4	CPUID	CR0 Flags		Action
OSFXSR	SSSE3 SSE4_1* SSE4_2**	EM	TS	
0	X***	X	X	#UD exception.
1	0	X	X	#UD exception.
1	1	1	X	#UD exception.
1	1	0	1	#NM exception.

NOTES:

* Applies to SSE4_1 instructions except DPPS, DPPD, ROUNDPS, ROUNDPD, ROUNDSS, ROUNDSD.

** Applies to SSE4_2 instructions except CRC32 and POPCNT.

***X — Don't care.

The SIMD floating-point exception mask bits (bits 7 through 12), the flush-to-zero flag (bit 15), the denormals-are-zero flag (bit 6), and the rounding control field (bits 13 and 14) in the MXCSR register should be left in their default values of 0. This permits the application to determine how these features are to be used.

12.1.5 Providing Non-Numeric Exception Handlers for Exceptions Generated by the SSE/SSE2/SSE3/SSSE3/SSE4 Instructions

SSE/SSE2/SSE3/SSSE3/SSE4 instructions can generate the same type of memory access exceptions (such as, page fault, segment not present, and limit violations) and other non-numeric exceptions as other Intel 64 and IA-32 architecture instructions generate.

Ordinarily, existing exception handlers can handle these and other non-numeric exceptions without code modification. However, depending on the mechanisms used in existing exception handlers, some modifications might need to be made.

The SSE/SSE2/SSE3/SSSE3/SSE4 extensions can generate the non-numeric exceptions listed below:

- Memory Access Exceptions:
  - — Invalid opcode (#UD).
  - — Stack-segment fault (#SS).
  - — General protection (#GP). Executing most SSE/SSE2/SSE3 instructions with an unaligned 128-bit memory reference generates a general-protection exception. (The MOVUPS and MOVUPD instructions allow unaligned a loads or stores of 128-bit memory locations, without generating a general-protection exception.) A 128-bit reference within the stack segment that is not aligned to a 16-byte boundary will also generate a general-protection exception, instead a stack-segment fault exception (#SS).
  - — Page fault (#PF).
  - — Alignment check (#AC). When enabled, this type of alignment check operates on operands that are less than 128-bits in size: 16-bit, 32-bit, and 64-bit. To enable the generation of alignment check exceptions, do the following:
    - Set the AM flag (bit 18 of control register CR0)

    - Set the AC flag (bit 18 of the EFLAGS register)
    - CPL must be 3.

    If alignment check exceptions are enabled, 16-bit, 32-bit, and 64-bit misalignment will be detected for the MOVUPD and MOVUPS instructions; detection of 128-bit misalignment is not guaranteed and may vary with implementation.

- System Exceptions:
  - — Invalid-opcode exception (#UD). This exception is generated when executing SSE/SSE2/SSE3/SSSE3/SSE4 instructions under the following conditions:
    - SSE/SSE2/SSE3/SSSE3/SSE4_1/SSE4_2 feature flags returned by CPUID are set to 0. This condition does not affect the CLFLUSH instruction, nor POPCNT.
    - The CLFSH feature flag returned by the CPUID instruction is set to 0. This exception condition only pertains to the execution of the CLFLUSH instruction.
    - The POPCNT feature flag returned by the CPUID instruction is set to 0. This exception condition only pertains to the execution of the POPCNT instruction.
    - The EM flag (bit 2) in control register CR0 is set to 1, regardless of the value of TS flag (bit 3) of CR0. This condition does not affect the PAUSE, PREFETCH*h*, MOVNTI, SFENCE, LFENCE, MFENSE, CLFLUSH, CRC32 and POPCNT instructions.
    - The OSFXSR flag (bit 9) in control register CR4 is set to 0. This condition does not affect the PAVGB, PAVGW, PEXTRW, PINSRW, PMAXSW, PMAXUB, PMINSW, PMINUB, PMOVMSKB, PMULHUW, PSADBW, PSHUFW, MASKMOVQ, MOVNTQ, MOVNTI, PAUSE, PREFETCH*h*, SFENCE, LFENCE, MFENCE, CLFLUSH, CRC32 and POPCNT instructions.
    - Executing a instruction that causes a SIMD floating-point exception when the OSXMMEXCPT flag (bit 10) in control register CR4 is set to 0. See Section 0.8.1, “Using the TS Flag to Control the Saving of the x87 FPU, MMX, SSE, SSE2, SSE3 SSSE3 and SSE4 State.”
  - — Device not available (#NM). This exception is generated by executing a SSE/SSE2/SSE3/SSSE3/SSE4 instruction when the TS flag (bit 3) of CR0 is set to 1.

Other exceptions can occur indirectly due to faulty execution of the above exceptions.

12.1.6 Providing an Handler for the SIMD Floating-Point Exception (#XM)

SSE/SSE2/SSE3/SSSE3/SSE4 instructions do not generate numeric exceptions on packed integer operations. They can generate the following numeric (SIMD floating-point) exceptions on packed and scalar single-precision and double-precision floating-point operations.

- Invalid operation (#I)
- Divide-by-zero (#Z)
- Denormal operand (#D)
- Numeric overflow (#O)
- Numeric underflow (#U)
- Inexact result (Precision) (#P)

These SIMD floating-point exceptions (with the exception of the denormal operand exception) are defined in the IEEE Standard 754 for Binary Floating-Point Arithmetic and represent the same conditions that cause x87 FPU floating-point error exceptions (#MF) to be generated for x87 FPU instructions.

Each of these exceptions can be masked, in which case the processor returns a reasonable result to the destination operand without invoking an exception handler. However, if any of these exceptions are left unmasked, detection of the exception condition results in a SIMD floating-point exception (#XM) being generated. See Chapter 5, "Generates an internal error signal that cause the processor to generate a floating-point exception (#MF).."

To handle unmasked SIMD floating-point exceptions, the operating system or executive must provide an exception handler. The section titled "SSE and SSE2 SIMD Floating-Point Exceptions" in Chapter 11, "Programming with Streaming SIMD Extensions 2 (SSE2)," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1*, describes the SIMD floating-point exception classes and gives suggestions for writing an exception handler to handle them.

To indicate that the operating system provides a handler for SIMD floating-point exceptions (#XM), the OSXMMEXCPT flag (bit 10) must be set in control register CR0.

12.1.6.1 Numeric Error flag and IGNNE#

SSE/SSE2/SSE3/SSE4 extensions ignore the NE flag in control register CR0 (that is, treats it as if it were always set) and the IGNNE# pin. When an unmasked SIMD floating-point exception is detected, it is always reported by generating a SIMD floating-point exception (#XM).

12.2 EMULATION OF SSE/SSE2/SSE3/SSSE3/SSE4 EXTENSIONS

The Intel 64 and IA-32 architecture does not support emulation of the SSE/SSE2/SSE3/SSSE3/SSE4 instructions, as they do for x87 FPU instructions.

The EM flag in control register CR0 (provided to invoke emulation of x87 FPU instructions) cannot be used to invoke emulation of SSE/SSE2/SSE3/SSSE3/SSE4 instructions. If an SSE/SSE2/SSE3/SSSE3/SSE4 instruction is executed when CR0.EM = 1, an invalid opcode exception (#UD) is generated. See Table 12-1.

12.3 SAVING AND RESTORING THE SSE/SSE2/SSE3/SSSE3/SSE4 STATE

The SSE/SSE2/SSE3/SSSE3/SSE4 state consists of the state of the XMM and MXCSR registers. The recommended method for saving and restoring this state follows:

- Execute an FXSAVE instruction to save the state of the XMM and MXCSR registers to memory.
- Execute an FXRSTOR instruction to restore the state of the XMM and MXCSR registers from the image saved in memory by the FXSAVE instruction.

This save and restore method is required for all operating systems. See Section 12.5, “Designing OS Facilities for AUTOMATICALLY Saving x87 FPU, MMX, and SSE/SSE2/SSE3/SSSE3/SSE4 state on Task or Context Switches.”

In some cases, applications can only save the XMM and MXCSR registers in the following way:

- Execute MOVDQ instructions to save the contents of each XMM registers to memory.
- Execute a STMXCSR instruction to save the state of the MXCSR register to memory.

In some cases, applications can only restore the XMM and MXCSR registers in the following way:

- Execute MOVDQ instructions to read the saved contents of each XMM registers from memory to XMM registers.
- Execute a LDMXCSR instruction to restore the state of the MXCSR register from memory.

12.4 SAVING THE SSE/SSE2/SSE3/SSSE3/SSE4 STATE ON TASK OR CONTEXT SWITCHES

When switching from one task or context to another, it is often necessary to save the SSE/SSE2/SSE3/SSSE3/SSE4 state. FXSAVE and FXRSTOR instructions provide a simple method for saving and restoring this state. See Section 12.3, “Saving and Restoring the SSE/SSE2/SSE3/SSSE3/SSE4 State.” These instructions offer the added benefit of saving x87 FPU and MMX state as well.

Guidelines for writing such procedures are in Section 12.5, “Designing OS Facilities for AUTOMATICALLY Saving x87 FPU, MMX, and SSE/SSE2/SSE3/SSSE3/SSE4 state on Task or Context Switches.”

12.5 DESIGNING OS FACILITIES FOR AUTOMATICALLY SAVING X87 FPU, MMX, AND SSE/SSE2/SSE3/SSSE3/SSE4 STATE ON TASK OR CONTEXT SWITCHES

The x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state consist of the state of the x87 FPU, MMX, XMM, and MXCSR registers. The FXSAVE and FXRSTOR instructions provide a fast method for saving ad restoring this state. If task or context switching facilities are already implemented in an operating system or executive and they use FSAVE/FNSAVE and FRSTOR to save the x87 FPU and MMX state, these facilities can be extended to save and restore SSE/SSE2/SSE3/SSSE3/SSE4 state by substituting FXSAVE/FXRSTOR for FSAVE/FNSAVE and FRSTOR.

Where task or content switching facilities must be written from scratch, several approaches can be taken for using the FXSAVE and FXRSTOR instructions to save and restore x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state:

- The operating system can require applications that are intended be run as tasks take responsibility for saving the state of the x87 FPU, MMX, XMM, and MXCSR registers prior to a task suspension during a task switch and for restoring the registers when the task is resumed. This approach is appropriate for cooperative multitasking operating systems, where the application has control over (or is able to determine) when a task switch is about to occur and can save state prior to the task switch.

- The operating system can take the responsibility for automatically saving the x87 FPU, MMX, XMM, and MXCSR registers as part of the task switch process (using an FXSAVE instruction) and automatically restoring the state of the registers when a suspended task is resumed (using an FXRSTOR instruction). Here, the x87 FPU/ MMX/SSE/SSE2/SSE3/SSE4 state must be saved as part of the task state. This approach is appropriate for preemptive multitasking operating systems, where the application cannot know when it is going to be preempted and cannot prepare in advance for task switching. Here, the operating system is responsible for saving and restoring the task and the x87 FPU/MMX/SSE/SSE2/SSE3 state when necessary.
- The operating system can take the responsibility for saving the x87 FPU, MMX, XMM, and MXCSR registers as part of the task switch process, but delay the saving of the MMX and x87 FPU state until an x87 FPU, MMX, or SSE/SSE2/SSE3/SSSE3/SSE4 instruction is actually executed by the new task. Using this approach, the x87 FPU/ MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state is saved only if an x87 FPU/MMX/SSE/ SSE2/SSE3/SSSE3/SSE4 instruction needs to be executed in the new task. (See Section 12.5.1, "Using the TS Flag to Control the Saving of the x87 FPU, MMX, SSE, SSE2, SSE3 SSSE3 and SSE4 State," for more information.)

12.5.1 Using the TS Flag to Control the Saving of the x87 FPU, MMX, SSE, SSE2, SSE3 SSSE3 and SSE4 State

Saving the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state using FXSAVE requires processor overhead. If the new task does not access x87 FPU, MMX, XMM, and MXCSR registers, avoid overhead by not automatically saving the state on a task switch.

The TS flag in control register CR0 is provided to allow the operating system to delay saving the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state until an instruction that actually accesses this state is encountered in a new task. When the TS flag is set, the processor monitors the instruction stream for an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/ SSE4 instruction. When the processor detects one of these instructions, it raises a device-not-available exception (#NM) prior to executing the instruction. The device-not-available exception handler can then be used to save the x87 FPU/MMX/SSE/SSE2/ SSE3/SSSE3/SSE4 state for the previous task (using an FXSAVE instruction) and load the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state for the current task (using an FXRSTOR instruction). If the task never encounters an x87 FPU/MMX/SSE/SSE2/SSE3/ SSSE3/SSE4 instruction, the device-not-available exception will not be raised and a task state will not be saved unnecessarily.

NOTE

The CRC32 and POPCNT instructions do not operate on the x87 FPU/ MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state. They operate on the general-purpose registers and are not involved in the OS's lazy FXSAVE/FXRSTOR technique.

The TS flag can be set either explicitly (by executing a MOV instruction to control register CR0) or implicitly (using the IA-32 architecture's native task switching mechanism). When the native task switching mechanism is used, the processor automatically sets the TS flag on a task switch. After the device-not-available handler has saved the x87 FPU/ MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state, it should execute the CLTS instruction to clear the TS flag.

Figure 12-1 gives an example of an operating system that implements x87 FPU/MMX/ SSE/SSE2/SSE3/SSSE3/SSE4 state saving using the TS flag. In this example, task A is the currently running task and task B is the new task. The operating system maintains a save area for the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state for each task and defines a variable (x87_MMX_SSE_SSE2_SSE3_StateOwner) that indicates the task that “owns” the state. In this example, task A is the current owner.

On a task switch, the operating system task switching code must execute the following pseudo-code to set the TS flag according to the current owner of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 state. If the new task (task B in this example) is not the current owner of this state, the TS flag is set to 1; otherwise, it is set to 0.

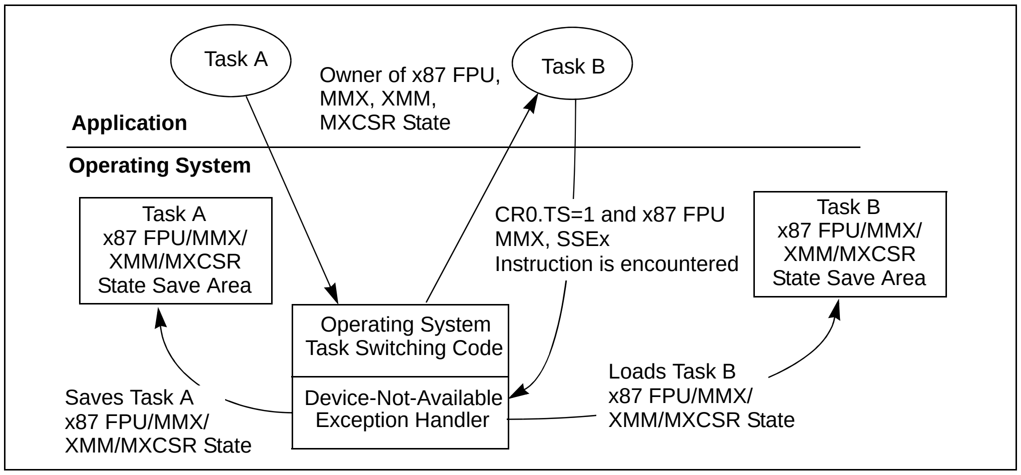

Task A
Owner of x87 FPU, MMX, XMM, MXCSR State
Task B
Application
Operating System
Task A x87 FPU/MMX/ XMM/MXCSR State Save Area
CR0.TS=1 and x87 FPU MMX, SSEx Instruction is encountered
Task B x87 FPU/MMX/ XMM/MXCSR State Save Area
Operating System Task Switching Code
Device-Not-Available Exception Handler
Loads Task B x87 FPU/MMX/ XMM/MXCSR State
Saves Task A x87 FPU/MMX/ XMM/MXCSR State

Figure 12-1. Example of Saving the x87 FPU, MMX, SSE, SSE2, SSE3, SSSE3 and SSE4 State During an Operating-System Controlled Task Switch

4. Power and Thermal Management (Chapter 13, Vol 3A)

Change bars show changes to Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A: System Programming Guide*.

13.1.1 Software Interface For Initiating Performance State Transitions

State transitions are initiated by writing a 16-bit value to the IA32_PERF_CTL register, see Figure 13-2. If a transition is already in progress, transition to a new value will subsequently take effect.

Reads of IA32_PERF_CTL determine the last targeted operating point. The current operating point can be read from IA32_PERF_STATUS. IA32_PERF_STATUS is updated dynamically.

The 16-bit encoding that defines valid operating points is model-specific. Applications and performance tools are not expected to use either IA32_PERF_CTL or IA32_PERF_STATUS and should treat both as reserved. Performance monitoring tools can access model-specific events and report the occurrences of state transitions.

...

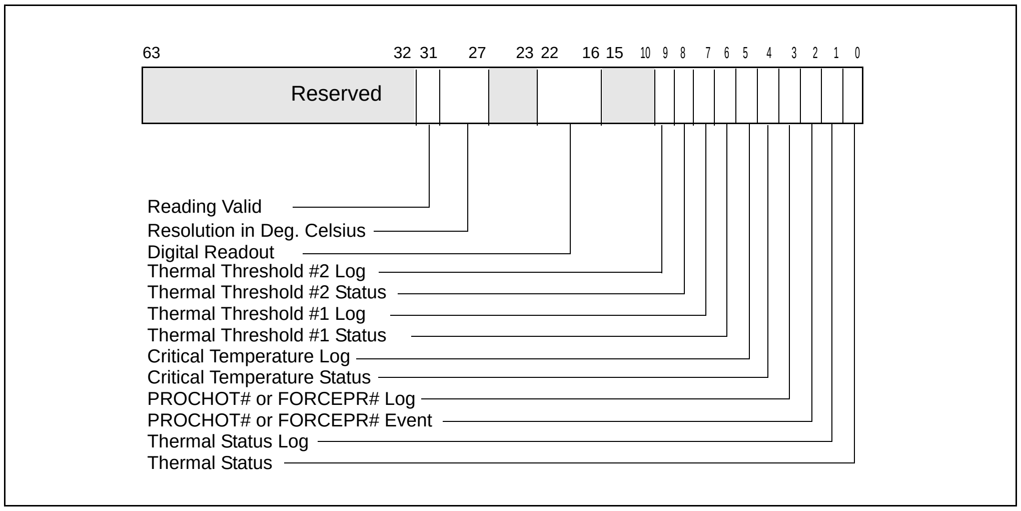
13.5.5.2 Reading the Digital Sensor

Unlike traditional analog thermal devices, the output of the digital thermal sensor is a temperature relative to the maximum supported operating temperature of the processor. Tj(Max).

Temperature measurements returned by digital thermal sensors are always at or below Tj(Max). Critical temperature conditions are detected using the "Critical Temperature Status" bit. When this bit is set, the processor is operating at a critical temperature and immediate shutdown of the system should occur. Once the "Critical Temperature Status" bit is set, reliable operation is not guaranteed.

See Figure 13-9 for the layout of IA32_THERM_STATUS MSR. Bit fields include:

- **Thermal Status (bit 0, RO)** — This bit indicates whether the digital thermal sensor high-temperature output signal (PROCHOT#) is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.
- **Thermal Status Log (bit 1, R/WC0)** — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (PROCHOT#). Bit 1 = 1 if PROCHOT# has been asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero.
- **PROCHOT# or FORCEPR# Event (bit 2, RO)** — Indicates whether PROCHOT# or FORCEPR# is being asserted by another agent on the platform.



Fgure 13-9. IA32_THERM_STATUS Register

- **PROCHOT# or FORCEPR# Log (bit 3, R/WC0)** — Sticky bit that indicates whether PROCHOT# or FORCEPR# has been asserted by another agent on the platform since the last clearing of this bit or a reset. If bit 3 = 1, PROCHOT# or FORCEPR# has been externally asserted. Software may clear this bit by writing a zero. External PROCHOT# assertions are only acknowledged if the Bidirectional Prochot feature is enabled.
- **Critical Temperature Status (bit 4, RO)** — Indicates whether the critical temperature detector output signal is currently active. If bit 4 = 1, the critical temperature detector output signal is currently active.

- **Critical Temperature Log (bit 5, R/WC0)** — Sticky bit that indicates whether the critical temperature detector output signal has been asserted since the last clearing of this bit or reset. If bit 5 = 1, the output signal has been asserted. Software may clear this bit by writing a zero.
- **Thermal Threshold #1 Status (bit 6, RO)** — Indicates whether the actual temperature is currently higher than or equal to the value set in Thermal Threshold #1. If bit 6 = 0, the actual temperature is lower. If bit 6 = 1, the actual temperature is greater than or equal to TT#1.
- **Thermal Threshold #1 Log (bit 7, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #1 has been reached since the last clearing of this bit or a reset. If bit 7 = 1, the Threshold #1 has been reached. Software may clear this bit by writing a zero.
- **Thermal Threshold #2 Status (bit 8, RO)** — Indicates whether actual temperature is currently higher than or equal to the value set in Thermal Threshold #2. If bit 8 = 0, the actual temperature is lower. If bit 8 = 1, the actual temperature is greater than or equal to TT#2.
- **Thermal Threshold #2 Log (bit 9, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #2 has been reached since the last clearing of this bit or a reset. If bit 9 = 1, the Thermal Threshold #2 has been reached. Software may clear this bit by writing a zero.
- **Digital Readout (bits 22:16, RO)** — Digital temperature reading in 1 degree Celsius relative to the PROCHOT # activation temperature - 1 degrees C. See the processor's data sheet for details regarding PROCHOT # activation temperature.
- **Resolution in Degrees Celsius (bits 30:27, RO)** — Specifies the resolution (or tolerance) of the digital thermal sensor. The value is in degrees Celsius. It is recommended that new threshold values be offset from the current temperature by at least the resolution + 1 in order to avoid hysteresis of interrupt generation.
- Reading Valid (bit 31, RO) — Indicates if the digital readout in bits 22:16 is valid. The readout is valid if bit 31 = 1.

Changes to temperature can be detected using two thresholds (see Figure 13-10); one is set above and the other below the current temperature. These thresholds have the capability of generating interrupts using the core's local APIC which software must then service. Note that the local APIC entries used by these thresholds are also used by the Intel® Thermal Monitor; it is up to software to determine the source of a specific interrupt.

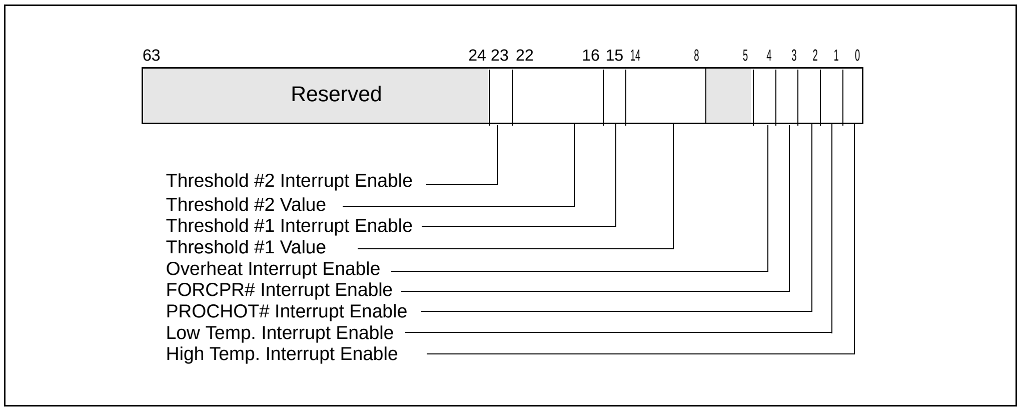

63
24 23 22
16 15 14
8
5 4 3 2 1 0
Reserved
Threshold #2 Interrupt Enable
Threshold #2 Value
Threshold #1 Interrupt Enable
Threshold #1 Value
Overheat Interrupt Enable
FORCPR# Interrupt Enable
PROCHOT# Interrupt Enable
Low Temp. Interrupt Enable
High Temp. Interrupt Enable

Figure 13-10. IA32_THERM_INTERRUPT Register

See Figure 13-10 for the layout of IA32_THERM_INTERRUPT MSR. Bit fields include:

- **High-Temperature Interrupt Enable (bit 0, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from low-temperature to a high-temperature threshold. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **Low-Temperature Interrupt Enable (bit 1, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from high-temperature to a low-temperature. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **PROCHOT# Interrupt Enable (bit 2, R/W)** — When a catastrophic cooling failure occurs, the processor will automatically shutdown. Bit 2 = 0 disables the feature; bit 2 = 1 enables the feature.
- **FORCPR# Interrupt Enable (bit 3, R/W)** — When a source external to the processor asserts PROCHOT#, the processor will throttle. Bit 3 = 0 disables the feature; bit 3 = 1 enables the feature.
- **Overheat Interrupt Enable (bit 4, R/W)** — Enables the generation of an interrupt when an overheat temperature [> PROCHOT # activation temperature] is reached. Bit 4 = 0 disables the feature; bit 4 = 1 enables the feature.
- **Threshold #1 Value (bits 14:8, R/W)** — A temperature threshold, relative to PROCHOT # activation temperature, used to generate an interrupt when the Threshold #1 Interrupt Enable bit is set and the actual relative temperature reaches or crosses this value.
- **Threshold #1 Interrupt Enable (bit 15, R/W)** — Enables the generation of an interrupt when the actual temperature relative to PROCHOT # activation temperature reaches the value specified in Threshold #1 value. Bit 15 = 0 disables the feature; bit 15 = 1 enables the feature.
- **Threshold #2 Value (bits 22:16, R/W)** — A temperature threshold, relative to PROCHOT # activation temperature, used to generate an interrupt when the Threshold #2 Interrupt Enable bit is set and the actual relative temperature reaches or crosses this value.
- **Threshold #2 Interrupt Enable (bit 23, R/W)** — Enables the generation of an interrupt when the actual temperature relative to PROCHOT # activation temperature reaches the value specified in Threshold #2 value. Bit 23 = 0 disables the feature; bit 23 = 1 enables the feature.

5. Machine-Check Architecture (Chapter 14, Vol 3A)

Change bars show changes to Chapter 14 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide*.

14.3.2 Error-Reporting Register Banks

Each error-reporting register bank can contain the IA32_MCi_CTL, IA32_MCi_STATUS, IA32_MCi_ADDR, and IA32_MCi_MISC MSRs. The number of reporting banks is indicated by bits [7:0] of IA32_MCG_CAP MSR (address 0179H). The first error-reporting register (IA32_MC0_CTL) always starts at address 400H.

See Appendix B, "Model-Specific Registers (MSRs)," for addresses of the error-reporting registers in the Pentium 4 and Intel Xeon processors; and for addresses of the error-reporting registers P6 family processors.

6. Debugging and Performance Monitoring (Chapter 18, Vol 3B)

Change bars show changes to Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide*.

18.1 OVERVIEW OF DEBUG SUPPORT FACILITIES

The following processor facilities support debugging and performance monitoring:

- **Debug exception (#DB) —** Transfers program control to a debug procedure or task when a debug event occurs.
- **Breakpoint exception (#BP) —** See breakpoint instruction (INT 3) below.
- **Breakpoint-address registers (DR0 through DR3) —** Specifies the addresses of up to 4 breakpoints.
- **Debug status register (DR6) —** Reports the conditions that were in effect when a debug or breakpoint exception was generated.
- **Debug control register (DR7) —** Specifies the forms of memory or I/O access that cause breakpoints to be generated.
- **T (trap) flag, TSS —** Generates a debug exception (#DB) when an attempt is made to switch to a task with the T flag set in its TSS.
- **RF (resume) flag, EFLAGS register —** Suppresses multiple exceptions to the same instruction.
- **TF (trap) flag, EFLAGS register —** Generates a debug exception (#DB) after every execution of an instruction.
- **Breakpoint instruction (INT 3) —** Generates a breakpoint exception (#BP) that transfers program control to the debugger procedure or task. This instruction is an alternative way to set code breakpoints.

...

18.12.2.1 Architectural Performance Monitoring Version 2 Facilities

The facilities provided by architectural performance monitoring version 2 can be queried from CPUID leaf 0AH by examining the content of register EDX:

- Bits 0 through 5 of CPUID.0AH.EDX indicates the number of fixed-function performance counters available per core,
- Bits 5 through 12 of CPUID.0AH.EDX indicates the bit-width of fixed-function performance counters. Bits beyond the width of the fixed-function counter are reserved and must be written as zeros.

NOTE

Early generation of processors based on Intel Core microarchitecture may report in CPUID.0AH:EDX of support for version 2 but indicating incorrect information of version 2 facilities.

The IA32_FIXED_CTR_CTRL MSR include multiple sets of 4-bit field, each 4 bit field controls the operation of a fixed-function performance counter. Figure 18-3 shows the layout of 4-bit controls for each fixed-function PMC. Two sub-fields are currently defined within each control. The definitions of the bit fields are:

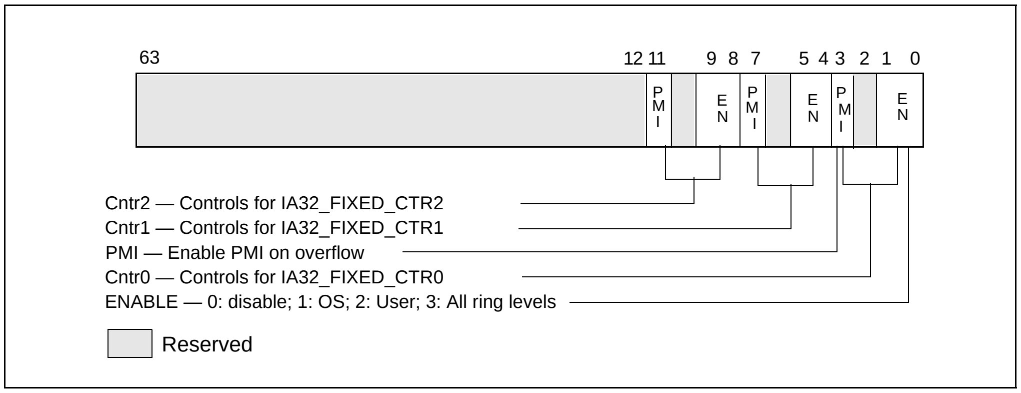

63
12 11
9 8 7
5 4 3 2 1 0
PMI
EN
PMI
EN
PMI
EN
Cntr2 — Controls for IA32_FIXED_CTR2
Cntr1 — Controls for IA32_FIXED_CTR1
PMI — Enable PMI on overflow
Cntr0 — Controls for IA32_FIXED_CTR0
ENABLE — 0: disable; 1: OS; 2: User; 3: All ring levels
Reserved

Figure 18-13. Layout of IA32_FIXED_CTR_CTRL MSR

- **Enable field (lowest 2 bits within each 4-bit control)** — When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring 0. When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment while the target condition associated with the architecture performance event occurred at ring greater than 0. Writing 0 to both bits stops the performance counter. Writing a value of 11B enables the counter to increment irrespective of privilege levels.
- **PMI field (the fourth bit within each 4-bit control)** — When set, the logical processor generates an exception through its local APIC on overflow condition of the respective fixed-function counter.

IA32_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting of each performance counter. Figure 18-14 shows the layout of IA32_PERF_GLOBAL_CTRL. Each enable bit in IA32_PERF_GLOBAL_CTRL is AND'ed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or IA32_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the AND'ed results is true; counting is disabled when the result is false.

...

18.14.2 Global Counter Control Facilities

Processors based on Intel Core microarchitecture provides simplified performance counter control that simplifies the most frequent operations in programming performance events, i.e. enabling/disabling event counting and checking the status of counter overflows. This is done by the following three MSRs:

- MSR_PERF_GLOBAL_CTRL enables/disables event counting for all or any combination of fixed-function PMCs (MSR_PERF_FIXED_CTRx) or general-purpose PMCs via WRMSR once.
- MSR_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs (MSR_PERF_FIXED_CTRx) or general-purpose PMCs via RDMSR once.
- MSR_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs (MSR_PERF_FIXED_CTRx) or general-purpose PMCs via WRMSR once.

MSR_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting in each performance counter (see Figure 18-18). Each enable bit in MSR_PERF_GLOBAL_CTRL is AND'ed with the enable bits for all privilege levels in the respective IA32_PERFEVTSELx or MSR_PERF_FIXED_CTR_CTRL MSRs to start/stop the counting of respective counters. Counting is enabled if the AND'ed results is true; counting is disabled when the result is false.

...

18.14.4 Precise Event Based Sampling (PEBS)

Processors based on Intel Core microarchitecture also support precise event based sampling (PEBS). This feature was introduced by processors based on Intel NetBurst microarchitecture.

PEBS uses a debug store mechanism and a performance monitoring interrupt to store a set of architectural state information for the processor. The information provides architectural state of the instruction executed after the instruction that caused the event (See Section 18.4.4.2).

In cases where the same instruction causes BTS and PEBS to be activated, PEBS is processed before BTS are processed. The PMI request is held until the processor completes processing of PEBS and BTS.

For processors based on Intel Core microarchitecture, events that support precise sampling are listed in Table 18-15. The procedure for detecting availability of PEBS is the same as described in Section 18.15.8.1.

Table 18-15. PEBS Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
INSTR_RETIRED.ANY_P	00H	C0H
X87_OPS_RETIRED.ANY	FEH	C1H
BR_INST_RETIRED.MISPRED	00H	C5H
SIMD_INST_RETIRED.ANY	1FH	C7H
MEM_LOAD_RETIRED.L1D_MISS	01H	CBH
MEM_LOAD_RETIRED.L1D_LINE_MISS	02H	CBH
MEM_LOAD_RETIRED.L2_MISS	04H	CBH
MEM_LOAD_RETIRED.L2_LINE_MISS	08H	CBH
MEM_LOAD_RETIRED.DTLB_MISS	10H	CBH

18.14.4.1 Setting up the PEBS Buffer

For processors based on Intel Core microarchitecture, PEBS is available using IA32_PMC0 only. Use the following procedure to set up the processor and IA32_PMC0 counter for PEBS:

1. Set up the precise event buffering facilities. Place values in the precise event buffer base, precise event index, precise event absolute maximum, precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area. In processors based on Intel Core microarchitecture, PEBS records consist of 64-bit address entries. See Figure 18-27 to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS on PMC0 flag (bit 0) in IA32_PEBS_ENABLE MSR.
3. Set up the IA32_PMC0 performance counter and IA32_PERFEVTSEL0 for an event listed in Table 18-15.

18.14.4.2 PEBS Record Format

The PEBS record format may be extended across different processor implementations. The IA32_PERF_CAPABILITES MSR defines a mechanism for software to handle the evolution of PEBS record format in processors that support architectural performance monitoring with version id 2 or higher. The bit fields of IA32_PERF_CAPABILITES are defined in Table B-2 of Appendix B, “Model-Specific Registers (MSRs)”. The relevant bit fields that govern PEBS are:

- PEBSTrap [bit 6]: When set, PEBS recording is trap-like. After the PEBS-enabled counter has overflowed, PEBS record is recorded for the next PEBS-able event at the completion of the sampled instruction causing the PEBS event. When clear, PEBS recording is fault-like. The PEBS record is recorded before the sampled instruction cuasing the PEBS event.
- PEBSSaveArchRegs [bit 7]: When set, PEBS will save architectural register and state information according to the encoded value of the PEBSRecordFormat field. On processors based on Intel Core microarchitecture, this bit is always 1
- PEBSRecordFormat [bits 11:8]: Valid encodings are:
  - — 0000B: Only general-purpose registers, instruction pointer and RFLAGS registers are saved in each PEBS record (see Section 18.15.8) .

...

18.18 PERFORMANCE MONITORING, BRANCH PROFILING AND SYSTEM EVENTS

When performance monitoring facilities and/or branch profiling facilities (see Section 18.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™2 Duo Processor Family)”) are enabled, these facilities capture event counts, branch records and branch trace messages occurring in a logical processor. The occurrence of interrupts, instruction streams due to various interrupt handlers all contribute to the results recorded by these facilities.

If CPUID.01H:ECX.PDCM[bit 15] is 1, the processor supports the IA32_PERF_CAPABILITIES MSR. If IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is 1, the processor supports the ability for system software using performance monitoring and/or branch profiling facilities to filter out the effects of servicing system management interrupts.

If the FREEZE_WHILE_SMM capability is enabled on a logical processor and after an SMI is delivered, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTM, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler.

The enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored , after the SMI handler issues RSM to complete its servicing.

It is the responsibility of the SMM code to ensure the state of the performance monitoring and branch profiling facilities are preserved upon entry or until prior to exiting the SMM. If any of this state is modified due to actions by the SMM code, the SMM code is required to restore such state to the values present at entry to the SMM handler.

System software is allowed to set IA32_DEBUGCTL.FREEZE_WHILE_SMM_EN[bit 14] to 1 only supported as indicated by IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] reporting 1.

§