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

Volume 3B: System Programming Guide, Part 2

NOTE: The *Intel® 64 and IA-32 Architectures Software Developer's Manual* consists of five volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-M*, Order Number 253666; *Instruction Set Reference N-Z*, Order Number 253667; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669. Refer to all five volumes when evaluating your design needs.

Order Number: 253669-027US
July 2008

Intel may make changes to specifications and product descriptions at any time, without notice.

Developers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Improper use of reserved or undefined features or instructions may cause unpredictable behavior or failure in developer's software code when running on an Intel processor. Intel reserves these features or instructions for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from their unauthorized use.

The Intel® 64 architecture processors may contain design defects or errors known as errata. Current characterized errata are available on request.

Hyper-Threading Technology requires a computer system with an Intel® processor supporting Hyper-Threading Technology and an HT Technology enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. For more information, see http://www.intel.com/technology/hyperthread/index.htm; including details on which processors support HT Technology.

Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and for some uses, certain platform software enabled for it. Functionality, performance or other benefits will_vary depending on hardware and software configurations. Intel® Virtualization Technology-enabled BIOS and VMM applications are currently in development.

64-bit computing on Intel architecture requires a computer system with a processor, chipset, BIOS, operating system, device drivers and applications enabled for Intel® 64 architecture. Processors will not operate (including 32-bit operation) without an Intel® 64 architecture-enabled BIOS. Performance will vary depending on your hardware and software configurations. Consult with your system vendor for more information.

Enabling Execute Disable Bit functionality requires a PC with a processor with Execute Disable Bit capability and a supporting operating system. Check with your PC manufacturer on whether your system delivers Execute Disable Bit functionality.

Intel, Pentium, Intel Xeon, Intel NetBurst, Intel Core Solo, Intel Core Duo, Intel Core 2 Duo, Intel Core 2 Extreme, Intel Pentium D, Itanium, Intel SpeedStep, MMX, Intel Atom, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 5937
Denver, CO 80217-9808

or call 1-800-548-4725
or visit Intel's website at http://www.intel.com

CHAPTER 18
DEBUGGING AND PERFORMANCE MONITORING

Intel 64 and IA-32 architectures provide debug facilities for use in debugging code and monitoring performance. These facilities are valuable for debugging application software, system software, and multitasking operating systems. Debug support is accessed using debug registers (DB0 through DB7) and model-specific registers (MSRs):

- Debug registers hold the addresses of memory and I/O locations called breakpoints. Breakpoints are user-selected locations in a program, a data-storage area in memory, or specific I/O ports. They are set where a programmer or system designer wishes to halt execution of a program and examine the state of the processor by invoking debugger software. A debug exception (#DB) is generated when a memory or I/O access is made to a breakpoint address.
- MSRs monitor branches, interrupts, and exceptions; they record addresses of the last branch, interrupt or exception taken and the last branch taken before an interrupt or exception.

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. It is especially useful

when more than four breakpoints are desired, or when breakpoints are being placed in the source code.

- **Last branch recording facilities** — Store branch records in the last branch record (LBR) stack MSRs for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consist of a branch-from and a branch-to instruction address. Send branch records out on the system bus as branch trace messages (BTMs).

These facilities allow a debugger to be called as a separate task or as a procedure in the context of the current program or task. The following conditions can be used to invoke the debugger:

- Task switch to a specific task.
- Execution of the breakpoint instruction.
- Execution of any instruction.
- Execution of an instruction at a specified address.
- Read or write to a specified memory address/range.
- Write to a specified memory address/range.
- Input from a specified I/O address/range.
- Output to a specified I/O address/range.
- Attempt to change the contents of a debug register.

18.2 DEBUG REGISTERS

Eight debug registers (see Figure 18-1) control the debug operation of the processor. These registers can be written to and read using the move to/from debug register form of the MOV instruction. A debug register may be the source or destination operand for one of these instructions.

Debug registers are privileged resources; a MOV instruction that accesses these registers can only be executed in real-address mode, in SMM or in protected mode at a CPL of 0. An attempt to read or write the debug registers from any other privilege level generates a general-protection exception (#GP).

The primary function of the debug registers is to set up and monitor from 1 to 4 breakpoints, numbered 0 though 3. For each breakpoint, the following information can be specified:

- The linear address where the breakpoint is to occur.
- The length of the breakpoint location (1, 2, or 4 bytes).
- The operation that must be performed at the address for a debug exception to be generated.
- Whether the breakpoint is enabled.

- Whether the breakpoint condition was present when the debug exception was generated.

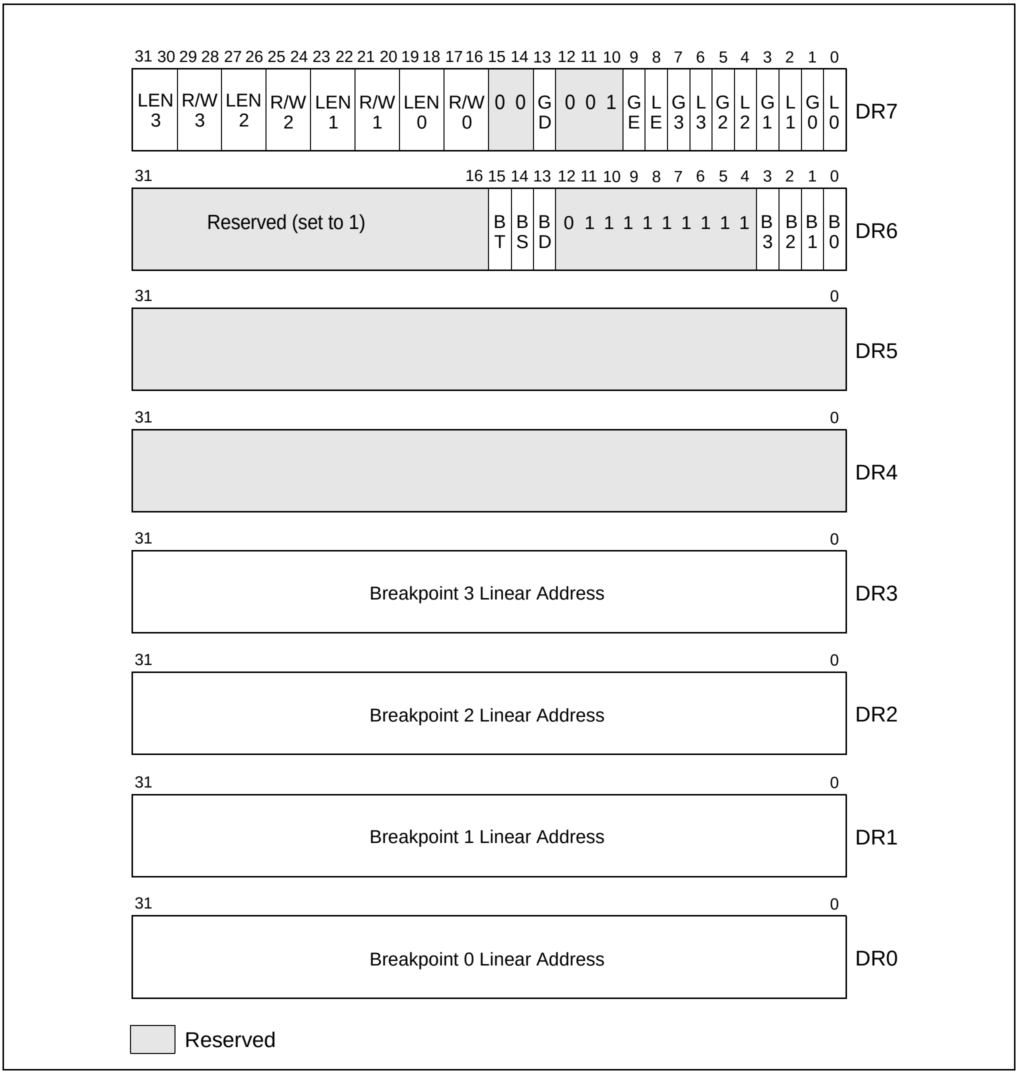

Figure 18-1. Debug Registers

The following paragraphs describe the functions of flags and fields in the debug registers.

18.2.1 Debug Address Registers (DR0-DR3)

Each of the debug-address registers (DR0 through DR3) holds the 32-bit linear address of a breakpoint (see Figure 18-1). Breakpoint comparisons are made before physical address translation occurs. The contents of debug register DR7 further specifies breakpoint conditions.

18.2.2 Debug Registers DR4 and DR5

Debug registers DR4 and DR5 are reserved when debug extensions are enabled (when the DE flag in control register CR4 is set) and attempts to reference the DR4 and DR5 registers cause invalid-opcode exceptions (#UD). When debug extensions are not enabled (when the DE flag is clear), these registers are aliased to debug registers DR6 and DR7.

18.2.3 Debug Status Register (DR6)

The debug status register (DR6) reports debug conditions that were sampled at the time the last debug exception was generated (see Figure 18-1). Updates to this register only occur when an exception is generated. The flags in this register show the following information:

- **B0 through B3 (breakpoint condition detected) flags (bits 0 through 3) —** Indicates (when set) that its associated breakpoint condition was met when a debug exception was generated. These flags are set if the condition described for each breakpoint by the LEN*n*, and R/W*n* flags in debug control register DR7 is true. They are set even if the breakpoint is not enabled by the L*n* and G*n* flags in register DR7. Non-enabled breakpoints matched on a previous iteration of a REP instruction may or may not be recorded in bits 0-3 on a debug exception caused by a later iteration of the REP instruction.
- **BD (debug register access detected) flag (bit 13) —** Indicates that the next instruction in the instruction stream accesses one of the debug registers (DR0 through DR7). This flag is enabled when the GD (general detect) flag in debug control register DR7 is set. See Section 18.2.4, “Debug Control Register (DR7),” for further explanation of the purpose of this flag.
- **BS (single step) flag (bit 14) —** Indicates (when set) that the debug exception was triggered by the single-step execution mode (enabled with the TF flag in the EFLAGS register). The single-step mode is the highest-priority debug exception. When the BS flag is set, any of the other debug status bits also may be set.
- **BT (task switch) flag (bit 15) —** Indicates (when set) that the debug exception resulted from a task switch where the T flag (debug trap flag) in the TSS of the target task was set. See Section 6.2.1, “Task-State Segment (TSS),” for the format of a TSS. There is no flag in debug control register DR7 to enable or disable this exception; the T flag of the TSS is the only enabling flag.

Certain debug exceptions may clear bits 0-3. The remaining contents of the DR6 register are never cleared by the processor. To avoid confusion in identifying debug exceptions, debug handlers should clear the register before returning to the interrupted task.

18.2.4 Debug Control Register (DR7)

The debug control register (DR7) enables or disables breakpoints and sets breakpoint conditions (see Figure 18-1). The flags and fields in this register control the following things:

- **L0 through L3 (local breakpoint enable) flags (bits 0, 2, 4, and 6) —** Enables (when set) the breakpoint condition for the associated breakpoint for the current task. When a breakpoint condition is detected and its associated L*n* flag is set, a debug exception is generated. The processor automatically clears these flags on every task switch to avoid unwanted breakpoint conditions in the new task.
- **G0 through G3 (global breakpoint enable) flags (bits 1, 3, 5, and 7) —** Enables (when set) the breakpoint condition for the associated breakpoint for all tasks. When a breakpoint condition is detected and its associated G*n* flag is set, a debug exception is generated. The processor does not clear these flags on a task switch, allowing a breakpoint to be enabled for all tasks.
- **LE and GE (local and global exact breakpoint enable) flags (bits 8, 9) —** This feature is not supported in the P6 family processors, later IA-32 processors, and Intel 64 processors. When set, these flags cause the processor to detect the exact instruction that caused a data breakpoint condition. For backward and forward compatibility with other Intel processors, we recommend that the LE and GE flags be set to 1 if exact breakpoints are required.
- **GD (general detect enable) flag (bit 13) —** Enables (when set) debug-register protection, which causes a debug exception to be generated prior to any MOV instruction that accesses a debug register. When such a condition is detected, the BD flag in debug status register DR6 is set prior to generating the exception. This condition is provided to support in-circuit emulators.

  When the emulator needs to access the debug registers, emulator software can set the GD flag to prevent interference from the program currently executing on the processor.

  The processor clears the GD flag upon entering to the debug exception handler, to allow the handler access to the debug registers.
- **R/W0 through R/W3 (read/write) fields (bits 16, 17, 20, 21, 24, 25, 28, and 29) —** Specifies the breakpoint condition for the corresponding breakpoint. The DE (debug extensions) flag in control register CR4 determines how the bits in the R/W*n* fields are interpreted. When the DE flag is set, the processor interprets bits as follows:

00 — Break on instruction execution only.
01 — Break on data writes only.
10 — Break on I/O reads or writes.
11 — Break on data reads or writes but not instruction fetches.

When the DE flag is clear, the processor interprets the R/W*n* bits the same as for the Intel386™ and Intel486™ processors, which is as follows:

00 — Break on instruction execution only.
01 — Break on data writes only.
10 — Undefined.
11 — Break on data reads or writes but not instruction fetches.

- **LEN0 through LEN3 (Length) fields (bits 18, 19, 22, 23, 26, 27, 30, and 31) —** Specify the size of the memory location at the address specified in the corresponding breakpoint address register (DR0 through DR3). These fields are interpreted as follows:

  00 — 1-byte length.
  01 — 2-byte length.
  10 — Undefined (or 8 byte length, see note below).
  11 — 4-byte length.

If the corresponding RW*n* field in register DR7 is 00 (instruction execution), then the LEN*n* field should also be 00. The effect of using other lengths is undefined. See Section 18.2.5, "Breakpoint Field Recognition," below.

NOTES

For Pentium® 4 and Intel® Xeon® processors with a CPUID signature corresponding to family 15 (model 3, 4, and 6), break point conditions permit specifying 8-byte length on data read/write with an of encoding 10B in the LENx field.

Encoding 10B is also supported in processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture, the respective CPUID signatures corresponding to family 6, model 15, and family 6, display_model value 23. The Encoding 10B is supported in processors based on Intel Atom microarchitecture, with CPUID signature of family 6, display_model value 28. The encoding 10B is undefined for other processors.

18.2.5 Breakpoint Field Recognition

Breakpoint address registers (debug registers DR0 through DR3) and the LEN*n* fields for each breakpoint define a range of sequential byte addresses for a data or I/O breakpoint. The LEN*n* fields permit specification of a 1-, 2-, 4-, or 8-byte range, beginning at the linear address specified in the corresponding debug register (DR*n*). Two-byte ranges must be aligned on word boundaries; 4-byte ranges must be aligned on doubleword boundaries. I/O addresses are zero-extended (from 16 to 32 bits, for comparison with the breakpoint address in the selected debug register). These requirements are enforced by the processor; it uses LEN*n* field bits to mask the lower address bits in the debug registers. Unaligned data or I/O breakpoint addresses do not yield valid results.

A data breakpoint for reading or writing data is triggered if any of the bytes participating in an access is within the range defined by a breakpoint address register and its LEN*n* field. Table 18-1 provides an example setup of debug registers and data accesses that would subsequently trap or not trap on the breakpoints.

A data breakpoint for an unaligned operand can be constructed using two breakpoints, where each breakpoint is byte-aligned and the two breakpoints together cover the operand. The breakpoints generate exceptions only for the operand, not for neighboring bytes.

Instruction breakpoint addresses must have a length specification of 1 byte (the LEN*n* field is set to 00). Code breakpoints for other operand sizes are undefined. The processor recognizes an instruction breakpoint address only when it points to the first byte of an instruction. If the instruction has prefixes, the breakpoint address must point to the first prefix.

Table 18-1. Breakpoint Examples

<table>
<tr><th colspan="4">Debug Register Setup</th></tr>
<tr><th>Debug Register</th><th>R/W<i>n</i></th><th>Breakpoint Address</th><th>LEN<i>n</i></th></tr>
<tr><td>DR0</td><td>R/W0 = 11 (Read/Write)</td><td>A0001H</td><td>LEN0 = 00 (1 byte)</td></tr>
<tr><td>DR1</td><td>R/W1 = 01 (Write)</td><td>A0002H</td><td>LEN1 = 00 (1 byte)</td></tr>
<tr><td>DR2</td><td>R/W2 = 11 (Read/Write)</td><td>B0002H</td><td>LEN2 = 01) (2 bytes)</td></tr>
<tr><td>DR3</td><td>R/W3 = 01 (Write)</td><td>C0000H</td><td>LEN3 = 11 (4 bytes)</td></tr>
<tr><th colspan="4">Data Accesses</th></tr>
<tr><th colspan="2">Operation</th><td>Address</td><th>Access Length
(In Bytes)</th></tr>
<tr><td colspan="2">Data operations that trap</td><td></td><td></td></tr>
<tr><td colspan="2">- Read or write</td><td>A0001H</td><td>1</td></tr>
<tr><td colspan="2">- Read or write</td><td>A0001H</td><td>2</td></tr>
<tr><td colspan="2">- Write</td><td>A0002H</td><td>1</td></tr>
<tr><td colspan="2">- Write</td><td>A0002H</td><td>2</td></tr>
<tr><td colspan="2">- Read or write</td><td>B0001H</td><td>4</td></tr>
<tr><td colspan="2">- Read or write</td><td>B0002H</td><td>1</td></tr>
<tr><td colspan="2">- Read or write</td><td>B0002H</td><td>2</td></tr>
<tr><td colspan="2">- Write</td><td>C0000H</td><td>4</td></tr>
<tr><td colspan="2">- Write</td><td>C0001H</td><td>2</td></tr>
<tr><td colspan="2">- Write</td><td>C0003H</td><td>1</td></tr>
</table>

Table 18-1. Breakpoint Examples (Contd.)

Debug Register Setup			
Debug Register	**R/W*n***	**Breakpoint Address**	**LEN*n***
Data operations that do not trap			
- Read or write		A0000H	1
- Read		A0002H	1
- Read or write		A0003H	4
- Read or write		B0000H	2
- Read		C0000H	2
- Read or write		C0004H	4

18.2.6 Debug Registers and Intel® 64 Processors

For Intel 64 architecture processors, debug registers DR0–DR7 are 64 bits. In 16-bit or 32-bit modes (protected mode and compatibility mode), writes to a debug register fill the upper 32 bits with zeros. Reads from a debug register return the lower 32 bits. In 64-bit mode, MOV DRn instructions read or write all 64 bits. Operand-size prefixes are ignored.

In 64-bit mode, the upper 32 bits of DR6 and DR7 are reserved and must be written with zeros. Writing 1 to any of the upper 32 bits results in a #GP(0) exception (see Figure 18-2). All 64 bits of DR0–DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0–DR3 are in the linear-address limits of the processor implementation (address matching is supported only on valid addresses generated by the processor implementation). Break point conditions for 8-byte memory read/writes are supported in all modes.

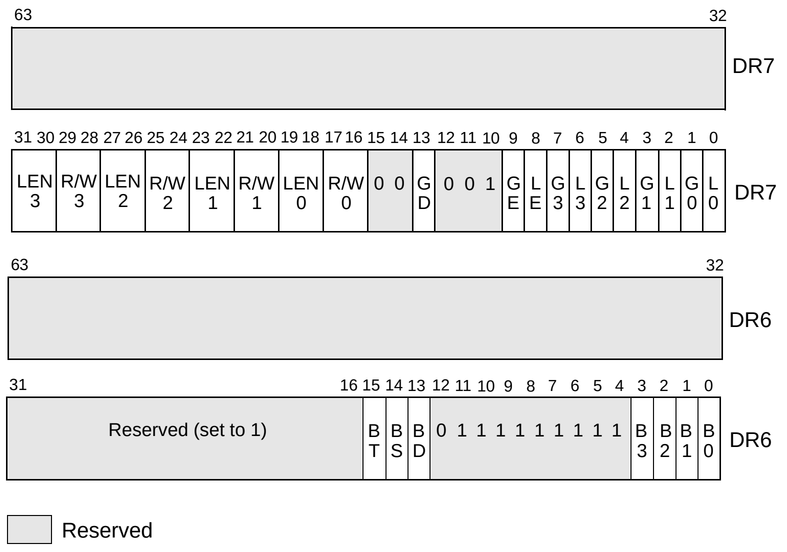

Figure 18-2. DR6/DR7 Layout on Processors Supporting Intel 64 Technology

18.3 DEBUG EXCEPTIONS

The Intel 64 and IA-32 architectures dedicate two interrupt vectors to handling debug exceptions: vector 1 (debug exception, #DB) and vector 3 (breakpoint exception, #BP). The following sections describe how these exceptions are generated and typical exception handler operations.

18.3.1 Debug Exception (#DB)—Interrupt Vector 1

The debug-exception handler is usually a debugger program or part of a larger software system. The processor generates a debug exception for any of several conditions. The debugger checks flags in the DR6 and DR7 registers to determine which condition caused the exception and which other conditions might apply. Table 18-2 shows the states of these flags following the generation of each kind of breakpoint condition.

Instruction-breakpoint and general-detect condition (see Section 18.3.1.3, “General-Detect Exception Condition”) result in faults; other debug-exception conditions result in traps. The debug exception may report one or both at one time. The following sections describe each class of debug exception.

See also: Chapter 5, “Interrupt 1—Debug Exception (#DB),” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

Table 18-2. Debug Exception Conditions

Debug or Breakpoint Condition	DR6 Flags Tested	DR7 Flags Tested	Exception Class
Single-step trap	BS = 1		Trap
Instruction breakpoint, at addresses defined by DR*n* and LEN*n*	B*n* = 1 and (G*n* or L*n* = 1)	R/Wn = 0	Fault
Data write breakpoint, at addresses defined by DR*n* and LEN*n*	B*n* = 1 and (G*n* or L*n* = 1)	R/Wn = 1	Trap
I/O read or write breakpoint, at addresses defined by DR*n* and LEN*n*	B*n* = 1 and (G*n* or L*n* = 1)	R/Wn = 2	Trap
Data read or write (but not instruction fetches), at addresses defined by DR*n* and LEN*n*	B*n* = 1 and (G*n* or L*n* = 1)	R/Wn = 3	Trap
General detect fault, resulting from an attempt to modify debug registers (usually in conjunction with in-circuit emulation)	BD = 1		Fault
Task switch	BT = 1		Trap

18.3.1.1 Instruction-Breakpoint Exception Condition

The processor reports an instruction breakpoint when it attempts to execute an instruction at an address specified in a breakpoint-address register (DB0 through DR3) that has been set up to detect instruction execution (R/W flag is set to 0). Upon reporting the instruction breakpoint, the processor generates a fault-class, debug exception (#DB) before it executes the target instruction for the breakpoint.

Instruction breakpoints are the highest priority debug exceptions. They are serviced before any other exceptions detected during the decoding or execution of an instruction. However, if a code instruction breakpoint is placed on an instruction located immediately after a POP SS/MOV SS instruction, the breakpoint may not be triggered. In most situations, POP SS/MOV SS will inhibit such interrupts (see “MOV—Move” and “POP—Pop a Value from the Stack” in Chapters 3 and 4 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A & 2B*).

Because the debug exception for an instruction breakpoint is generated before the instruction is executed, if the instruction breakpoint is not removed by the exception handler; the processor will detect the instruction breakpoint again when the instruction is restarted and generate another debug exception. To prevent looping on an instruction breakpoint, the Intel 64 and IA-32 architectures provide the RF flag (resume flag) in the EFLAGS register (see Section 2.3, “System Flags and Fields in the EFLAGS Register,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*). When the RF flag is set, the processor ignores instruction breakpoints.

All Intel 64 and IA-32 processors manage the RF flag as follows. The RF Flag is cleared at the start of the instruction after the check for code breakpoint, CS limit violation and FP exceptions. Task Switches and IRETD/IRETQ instructions transfer the RF image from the TSS/stack to the EFLAGS register.

When calling an event handler, Intel 64 and IA-32 processors establish the value of the RF flag in the EFLAGS image pushed on the stack:

- For any fault-class exception except a debug exception generated in response to an instruction breakpoint, the value pushed for RF is 1.
- For any interrupt arriving after any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For any trap-class exception generated by any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For other cases, the value pushed for RF is the value that was in EFLAG.RF at the time the event handler was called. This includes:
    - — Debug exceptions generated in response to instruction breakpoints
    - — Hardware-generated interrupts arriving between instructions (including those arriving after the last iteration of a repeated string instruction)
    - — Trap-class exceptions generated after an instruction completes (including those generated after the last iteration of a repeated string instruction)
    - — Software-generated interrupts (RF is pushed as 0, since it was cleared at the start of the software interrupt)

As noted above, the processor does not set the RF flag prior to calling the debug exception handler for debug exceptions resulting from instruction breakpoints. The debug exception handler can prevent recurrence of the instruction breakpoint by setting the RF flag in the EFLAGS image on the stack. If the RF flag in the EFLAGS image is set when the processor returns from the exception handler, it is copied into the RF flag in the EFLAGS register by IRETD/IRETQ or a task switch that causes the return. The processor then ignores instruction breakpoints for the duration of the next instruction. (Note that the POPF, POPFD, and IRET instructions do not transfer the RF image into the EFLAGS register.) Setting the RF flag does not prevent other types of debug-exception conditions (such as, I/O or data breakpoints) from being detected, nor does it prevent non-debug exceptions from being generated.

For the Pentium processor, when an instruction breakpoint coincides with another fault-type exception (such as a page fault), the processor may generate one spurious debug exception after the second exception has been handled, even though the debug exception handler set the RF flag in the EFLAGS image. To prevent a spurious exception with Pentium processors, all fault-class exception handlers should set the RF flag in the EFLAGS image.

18.3.1.2 Data Memory and I/O Breakpoint Exception Conditions

Data memory and I/O breakpoints are reported when the processor attempts to access a memory or I/O address specified in a breakpoint-address register (DB0 through DR3) that has been set up to detect data or I/O accesses (R/W flag is set to 1, 2, or 3). The processor generates the exception after it executes the instruction that made the access, so these breakpoint condition causes a trap-class exception to be generated.

Because data breakpoints are traps, the original data is overwritten before the trap exception is generated. If a debugger needs to save the contents of a write breakpoint location, it should save the original contents before setting the breakpoint. The handler can report the saved value after the breakpoint is triggered. The address in the debug registers can be used to locate the new value stored by the instruction that triggered the breakpoint.

Intel486 and later processors ignore the GE and LE flags in DR7. In Intel386 processors, exact data breakpoint matching does not occur unless it is enabled by setting the LE and/or the GE flags.

P6 family processors are unable to report data breakpoints exactly for the REP MOVS and REP STOS instructions until the completion of the iteration after the iteration in which the breakpoint occurred.

For repeated INS and OUTS instructions that generate an I/O-breakpoint debug exception, the processor generates the exception after the completion of the first iteration. Repeated INS and OUTS instructions generate a memory-breakpoint debug exception after the iteration in which the memory address breakpoint location is accessed.

18.3.1.3 General-Detect Exception Condition

When the GD flag in DR7 is set, the general-detect debug exception occurs when a program attempts to access any of the debug registers (DR0 through DR7) at the same time they are being used by another application, such as an emulator or debugger. This protection feature guarantees full control over the debug registers when required. The debug exception handler can detect this condition by checking the state of the BD flag in the DR6 register. The processor generates the exception before it executes the MOV instruction that accesses a debug register, which causes a fault-class exception to be generated.

18.3.1.4 Single-Step Exception Condition

The processor generates a single-step debug exception if (while an instruction is being executed) it detects that the TF flag in the EFLAGS register is set. The exception is a trap-class exception, because the exception is generated after the instruction is executed. The processor will not generate this exception after the instruction that sets the TF flag. For example, if the POPF instruction is used to set the TF flag, a single-step trap does not occur until after the instruction that follows the POPF instruction.

The processor clears the TF flag before calling the exception handler. If the TF flag was set in a TSS at the time of a task switch, the exception occurs after the first instruction is executed in the new task.

The TF flag normally is not cleared by privilege changes inside a task. The INT *n* and INTO instructions, however, do clear this flag. Therefore, software debuggers that single-step code must recognize and emulate INT *n* or INTO instructions rather than executing them directly. To maintain protection, the operating system should check the CPL after any single-step trap to see if single stepping should continue at the current privilege level.

The interrupt priorities guarantee that, if an external interrupt occurs, single stepping stops. When both an external interrupt and a single-step interrupt occur together, the single-step interrupt is processed first. This operation clears the TF flag. After saving the return address or switching tasks, the external interrupt input is examined before the first instruction of the single-step handler executes. If the external interrupt is still pending, then it is serviced. The external interrupt handler does not run in single-step mode. To single step an interrupt handler, single step an INT *n* instruction that calls the interrupt handler.

18.3.1.5 Task-Switch Exception Condition

The processor generates a debug exception after a task switch if the T flag of the new task's TSS is set. This exception is generated after program control has passed to the new task, and prior to the execution of the first instruction of that task. The exception handler can detect this condition by examining the BT flag of the DR6 register.

If the debug exception handler is a task, the T bit of its TSS should not be set. Failure to observe this rule will put the processor in a loop.

18.3.2 Breakpoint Exception (#BP)—Interrupt Vector 3

The breakpoint exception (interrupt 3) is caused by execution of an INT 3 instruction. See Chapter 5, “Interrupt 3—Breakpoint Exception (#BP).” Debuggers use break exceptions in the same way that they use the breakpoint registers; that is, as a mechanism for suspending program execution to examine registers and memory locations. With earlier IA-32 processors, breakpoint exceptions are used extensively for setting instruction breakpoints.

With the Intel386 and later IA-32 processors, it is more convenient to set breakpoints with the breakpoint-address registers (DR0 through DR3). However, the breakpoint exception still is useful for breakpointing debuggers, because a breakpoint exception can call a separate exception handler. The breakpoint exception is also useful when it is necessary to set more breakpoints than there are debug registers or when breakpoints are being placed in the source code of a program under development.

18.4 LAST BRANCH RECORDING OVERVIEW

P6 family processors introduced the ability to set breakpoints on taken branches, interrupts, and exceptions, and to single-step from one branch to the next. This capability has been modified and extended in the Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo and Intel® Atom™ processors to allow logging of branch trace messages in a branch trace store (BTS) buffer in memory.

See the following sections:

— Section 18.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™2 Duo and Intel® Atom™ Processor Family)”

— Section 18.6, “Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)”

— Section 18.7, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ Solo and Intel® Core™ Duo Processors)”

— Section 18.8, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors)”

— Section 18.9, “Last Branch, Interrupt, and Exception Recording (P6 Family Processors)”

The last branch recording mechanism tracks not only branch instructions (like JMP, Jcc, LOOP and CALL instructions), but also other operations that cause a change in the instruction pointer (like external interrupts, traps and faults).

18.5 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™2 DUO AND INTEL® ATOM™ PROCESSOR FAMILY)

The Intel Core 2 Duo processor family and Intel Xeon processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture provide last branch interrupt and exception recording. The facilities described in this section also apply to Intel Atom processor family. These capabilities are similar to those found in Pentium 4 processors, including support for the following facilities:

- **Debug Trace and Branch Recording Control —** The IA32_DEBUGCTL MSR provide bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 18.5.1.
- **Last branch record (LBR) stack —** There are a collection of MSR pairs that store the source and destination addresses related to recently executed branches. See Section 18.5.2.

- **Monitoring and single-stepping of branches, exceptions, and interrupts** — See Section 18.6.4 and Section 18.6.5. In addition, the ability to freeze the LBR stack on a PMI request is available.
- **Branch trace messages and last exception records** — See Section 18.6.6 and Section 18.6.7.
- **Branch trace store and CPL-qualified BTS** — See Section 18.6.8.

18.5.1 IA32_DEBUGCTL MSR

The **IA32_DEBUGCTL** MSR provides bit field controls to enable debug trace interrupts, debug trace stores, trace messages enable, single stepping on branches, last branch record recording, and to control freezing of LBR stack or performance counters on a PMI request. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 18-3 for the MSR layout and the bullets below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the Section 18.5.2, “LBR Stack”.
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches, interrupts, and exceptions. See Section 18.6.5, “Single-Stepping on Branches, Exceptions, and Interrupts,” for more information about the BTF flag.
- **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 18.6.6, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7)** — When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 18.16.5, “DS Save Area.”
- **BTINT (branch trace interrupt) flag (bit 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 18.6.8, “Branch Trace Store (BTS),” for a description of this mechanism.

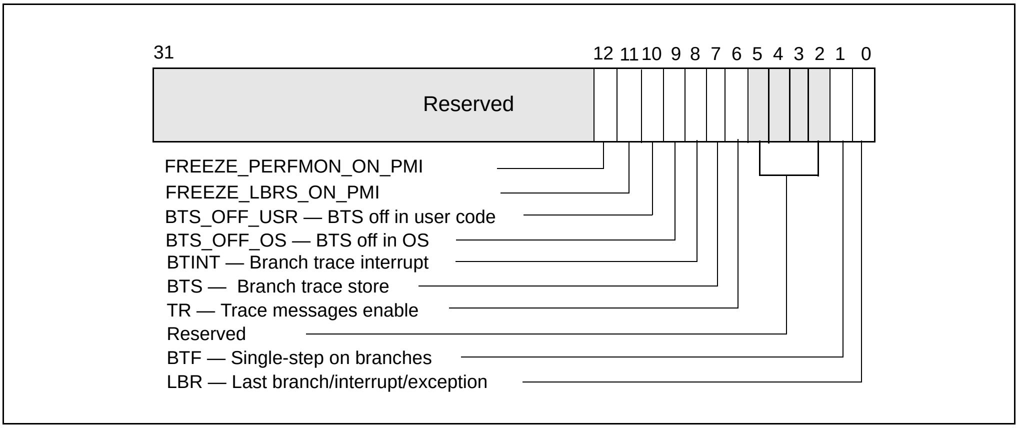
31
12
11
10
9
8
7
6
5
4
3
2
1
0
Reserved
FREEZE_PERFMON_ON_PMI
FREEZE_LBRS_ON_PMI
BTS_OFF_USR — BTS off in user code
BTS_OFF_OS — BTS off in OS
BTINT — Branch trace interrupt
BTS — Branch trace store
TR — Trace messages enable
Reserved
BTF — Single-step on branches
LBR — Last branch/interrupt/exception

Figure 18-3. IA32_DEBUGCTL MSR for Processors based on Intel Core microarchitecture

- **BTS_OFF_OS (branch trace off in privileged code) flag (bit 9) —** When set, BTS or BTM is skipped if CPL is 0. See Section 18.6.1.
- **BTS_OFF_USR (branch trace off in user code) flag (bit 10) —** When set, BTS or BTM is skipped if CPL is greater than 0. See Section 18.6.1.
- **FREEZE_LBRS_ON_PMI flag (bits 11) —** When set, the LBR stack is frozen on a hardware PMI request (e.g. when a counter overflows and is configured to trigger PMI).
- **FREEZE_PERFMON_ON_PMI flag (bits 12) —** When set, a PMI request clears each of the "ENABLE" field of MSR_PERF_GLOBAL_CTRL MSR (see Figure 18-21) to disable all the counters.

18.5.2 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel Core 2, Intel Xeon and Intel Atom processor families. However, the number of MSRs in the LBR stack and the valid range of TOS pointer value can vary between different processor families. Table 18-3 lists the LBR stack size and TOS pointer range for several processor families according to the CPUID signatures of DisplayFamily/DisplayModel encoding (see CPUID instruction in Chapter 3 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*).

Table 18-3. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_17H	4	0 to 3
06_0FH	4	0 to 3
06_1CH	8	0 to 7

- **Last Branch Record (LBR) Stack —** The LBR consists of N pairs of MSRs (N is listed in the LBR stack size column of Table 18-3) that store source and destination address of recent branches (see Figure 18-4):
  - — MSR_LASTBRANCH_0_FROM_IP (address 40H) through MSR_LASTBRANCH_(N-1)_FROM_IP (address 40H + N-1) stores source addresses
  - — MSR_LASTBRANCH_0_TO_IP (address 60H) through MSR_LASTBRANCH_(N-1)_To_IP (address 60H + N-1) stores destination addresses.

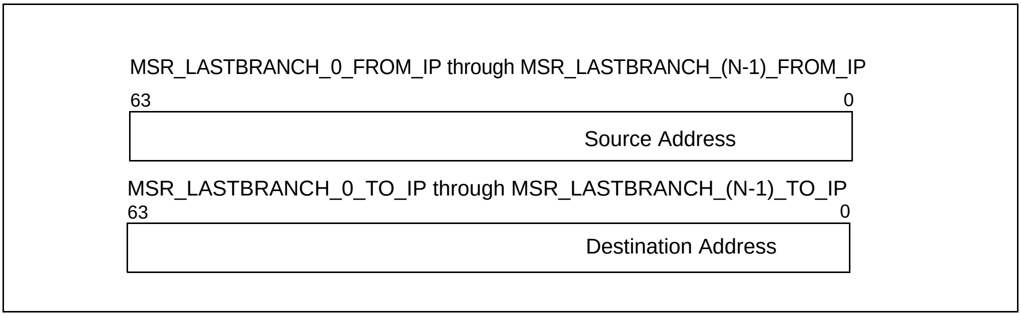
MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_(N-1)_FROM_IP
63
0
Source Address
MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_(N-1)_TO_IP
63
0
Destination Address

Figure 18-4. 64-bit Address Layout of LBR MSR

Software should query an architectural MSR IA32_PERF_CAPABILITIES[5:0] about the format of the address that is stored in the LBR stack. Three formats are defined by following encoding:

— **000000B (32-bit record format) —** Stores 32-bit offset in current CS of respective source/destination,

— **000001B (64-bit LIP record format) —** Stores 64-bit linear address of respective source/destination,

— **000010B (64-bit EIP record format) —** Stores 64-bit offset (effective address) of respective source/destination.

Processor's support for the architectural MSR IA32_PERF_CAPABILITIES is provided by CPUID.01H:ECX[PERF_CAPAB_MSR] (bit 15).

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest signifcant M bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains an M-bit pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. The valid range of the M-bit POS pointer is given in Table 18-3.

For compatibility, the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) duplicate functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

18.5.3 BTS and Related Facilities

The **Debug store (DS)** feature flag (bit 21), returned by CPUID.1:EDX[21] Indicates that the processor provides the debug store (DS) mechanism. This mechanism allows BTMs to be stored in a memory-resident BTS buffer. See Section 18.6.8, "Branch Trace Store (BTS)." Precise event-based sampling (PEBS) also uses the DS save area provided by debug store mechanism.

18.5.3.1 Freezing LBR and Performance Counters on PMI

Many issues may generate a performance monitoring interrupt (PMI); a PMI service handler will need to determine cause to handle the situation. Two capabilities that allow a PMI service routine to improve branch tracing and performance monitoring are:

- **Freezing LBRs on PMI** — The processor freezes LBRs on a PMI request by clearing the LBR bit (bit 0) in IA32_DEBUGCTL. Software must then re-enable IA32_DEBUGCTL.[0] to continue monitoring branches. When using this feature, software should be careful about writes to IA32_DEBUGCTL to avoid re-enabling LBRs by accident if they were just disabled.
- **Freezing PMCs on PMI** — The processor freezes the performance counters on a PMI request by clearing the MSR_PERF_GLOBAL_CTRL MSR (see Figure 18-21). The PMCs affected include both general-purpose counters and fixed-function counters (see Section 18.14.1, "Fixed-function Performance Counters"). Software must re-enable counts by writing 1s to the corresponding enable bits in MSR_PERF_GLOBAL_CTRL before leaving a PMI service routine to continue counter operation.

Freezing LBRs and PMCs on PMIs occur when:

- A performance counter had an overflow and was programmed to signal a PMI in case of an overflow.
  - — For the general-purpose counters; this is done by setting bit 20 of the IA32_PERFEVTSELx register.

  - — For the fixed-function counters; this is done by setting the 3rd bit in the corresponding 4-bit control field of the MSR_PERF_FIXED_CTR_CTRL register (see Figure 18-20) or IA32_FIXED_CTR_CTRL MSR (see Figure 18-13).
- The PEBS buffer is almost full and reaches the interrupt threshold.
- The BTS buffer is almost full and reaches the interrupt threshold.

18.5.3.2 Debug Store (DS) Mechanism

The debug store mechanism provides the DS save area for software to collect branch records or precise-event-based-sampling (PEBS) records. Fields in the buffer management area of a DS save area are described in Section 18.16.5.

The format of a branch trace record and a PEBS record are the same as the 64-bit record formats shown in Figures 18-31 and Figures 18-32, with the exception that the branch predicted bit is not supported by Intel Core microarchitecture. The 64-bit record formats for BTS and PEBS apply to DS save area for all operating modes.

The procedures used to program IA32_DEBUG_CTRL MSR to set up a BTS buffer or a CPL-qualified BTS are described in Section 18.6.8.3 and Section 18.6.8.4.

Required elements for writing a DS interrupt service routine are largely the same as those described in Section 18.6.8.5. However, instead of re-enabling counting using CCCRs like on processors based on Intel NetBurst® microarchitecture, a DS interrupt service routine on processors based on Intel Core or Intel Atom microarchitecture should:

- Re-enable the enable bits in MSR_PERF_GLOBAL_CTRL MSR if it is servicing an overflow PMI due to PEBS.
- Clear overflow indications by writing to MSR_PERF_GLOBAL_OVF_CTRL when a counting configuration is changed. This includes bit 62 (ClrOvfBuffer) and the overflow indication of counters used in either PEBS or general-purpose counting (specifically: bits 0 or 1; see Figures 18-23).

18.6 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PROCESSORS BASED ON INTEL NETBURST® MICROARCHITECTURE)

Pentium 4 and Intel Xeon processors based on Intel NetBurst microarchitecture provide the following methods for recording taken branches, interrupts and exceptions:

- Store branch records in the last branch record (LBR) stack MSRs for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consist of a branch-from and a branch-to instruction address.
- Send the branch records out on the system bus as branch trace messages (BTMs).

- Log BTMs in a memory-resident branch trace store (BTS) buffer.

To support these functions, the processor provides the following MSRs and related facilities:

- **MSR_DEBUGCTLA MSR** — Enables last branch, interrupt, and exception recording; single-stepping on taken branches; branch trace messages (BTMs); and branch trace store (BTS). This register is named DebugCtlMSR in the P6 family processors.
- **Debug store (DS) feature flag (CPUID.1:EDX.DS[bit 21])** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer.
- **CPL-qualified debug store (DS) feature flag (CPUID.1:ECX.DS-CPL[bit 4])** — Indicates that the processor provides a CPL-qualified debug store (DS) mechanism, which allows software to selectively skip sending and storing BTMs, according to specified current privilege level settings, into a memory-resident BTS buffer.
- **IA32_MISC_ENABLE MSR** — Indicates that the processor provides the BTS facilities.
- **Last branch record (LBR) stack** — The LBR stack is a circular stack that consists of four MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. The LBR stack consists of 16 MSR pairs (MSR_LASTBRANCH_0_FROM_LIP through MSR_LASTBRANCH_15_FROM_LIP and MSR_LASTBRANCH_0_TO_LIP through MSR_LASTBRANCH_15_TO_LIP) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H].
- **Last branch record top-of-stack (TOS) pointer** — The TOS Pointer MSR contains a 2-bit pointer (0-3) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. This pointer becomes a 4-bit pointer (0-15) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H]. See also: Table 18-4, Figure 18-5, and Section 18.6.3, “LBR Stack for Processors Based on Intel NetBurst Microarchitecture.”
- **Last exception record** — See Section 18.6.7, “Last Exception Records.”

18.6.1 CPL-Qualified Branch Trace Mechanism

CPL-qualified branch trace mechanism is available to a subset of Intel 64 and IA-32 processors that support the branch trace storing mechanism. The processor supports the CPL-qualified branch trace mechanism if CPUID.01H:ECX[bit 4] = 1.

The CPL-qualified branch trace mechanism is described in Section 18.6.8.4. System software can selectively specify CPL qualification to not send/store Branch Trace Messages associated with a specified privilege level. Two bit fields, BTS_OFF_USR and BTS_OFF_OS, are provided in the debug control register to specify the CPL of BTMs that will not be logged in the BTS buffer or sent on the bus.

Table 18-4. LBR MSR Stack Structure for the Pentium® 4 and the Intel® Xeon® Processor Family

LBR MSRs for Family 0FH, Models 0H-02H; MSRs at locations 1DBH-1DEH.	**Decimal Value of TOS Pointer in MSR_LASTBRANCH_TOS (bits 0-1)**
MSR_LASTBRANCH_0	0
MSR_LASTBRANCH_1	1
MSR_LASTBRANCH_2	2
MSR_LASTBRANCH_3	3
LBR MSRs for Family 0FH, Models; MSRs at locations 680H-68FH.	**Decimal Value of TOS Pointer in MSR_LASTBRANCH_TOS (bits 0-3**
MSR_LASTBRANCH_0_FROM_LIP	0
MSR_LASTBRANCH_1_FROM_LIP	1
MSR_LASTBRANCH_2_FROM_LIP	2

Table 18-4. LBR MSR Stack Structure for the Pentium® 4 and the Intel® Xeon® Processor Family (Contd.)

LBR MSRs for Family 0FH, Models; MSRs at locations 680H-68FH.	**Decimal Value of TOS Pointer in MSR_LASTBRANCH_TOS (bits 0-3)**
MSR_LASTBRANCH_3_FROM_LIP	3
MSR_LASTBRANCH_4_FROM_LIP	4
MSR_LASTBRANCH_5_FROM_LIP	5
MSR_LASTBRANCH_6_FROM_LIP	6
MSR_LASTBRANCH_7_FROM_LIP	7
MSR_LASTBRANCH_8_FROM_LIP	8
MSR_LASTBRANCH_9_FROM_LIP	9
MSR_LASTBRANCH_10_FROM_LIP	10
MSR_LASTBRANCH_11_FROM_LIP	11
MSR_LASTBRANCH_12_FROM_LIP	12
MSR_LASTBRANCH_13_FROM_LIP	13
MSR_LASTBRANCH_14_FROM_LIP	14
MSR_LASTBRANCH_15_FROM_LIP	15
LBR MSRs for Family 0FH, Model 03H; MSRs at locations 6C0H-6CFH.	
MSR_LASTBRANCH_0_TO_LIP	0
MSR_LASTBRANCH_1_TO_LIP	1
MSR_LASTBRANCH_2_TO_LIP	2
MSR_LASTBRANCH_3_TO_LIP	3
MSR_LASTBRANCH_4_TO_LIP	4
MSR_LASTBRANCH_5_TO_LIP	5
MSR_LASTBRANCH_6_TO_LIP	6
MSR_LASTBRANCH_7_TO_LIP	7
MSR_LASTBRANCH_8_TO_LIP	8
MSR_LASTBRANCH_9_TO_LIP	9
MSR_LASTBRANCH_10_TO_LIP	10
MSR_LASTBRANCH_11_TO_LIP	11
MSR_LASTBRANCH_12_TO_LIP	12
MSR_LASTBRANCH_13_TO_LIP	13
MSR_LASTBRANCH_14_TO_LIP	14
MSR_LASTBRANCH_15_TO_LIP	15

NOTE

> The initial implementation of BTS_OFF_USR and BTS_OFF_OS in MSR_DEBUGCTLA is shown in Figure 18-5. The BTS_OFF_USR and BTS_OFF_OS fields may be implemented on other model-specific debug control register at different locations.

The following sections describe the MSR_DEBUGCTLA MSR and the various last branch recording mechanisms. See Appendix B, “Model-Specific Registers (MSRs),” for a detailed description of each of the last branch recording MSRs.

18.6.2 MSR_DEBUGCTLA MSR

The MSR_DEBUGCTLA MSR enables and disables the various last branch recording mechanisms described in the previous section. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode. A protected-mode operating system procedure is required to provide user access to this register. Figure 18-5 shows the flags in the MSR_DEBUGCTLA MSR. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0) —** When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. Each branch, interrupt, or exception is recorded as a 64-bit branch record. The processor clears this flag whenever a debug exception is generated (for example, when an instruction or data breakpoint or a single-step trap occurs). See Section 18.6.3, “LBR Stack for Processors Based on Intel NetBurst Microarchitecture.”
- **BTF (single-step on branches) flag (bit 1) —** When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches, interrupts, and exceptions. See Section 18.6.5, “Single-Stepping on Branches, Exceptions, and Interrupts.”
- **TR (trace message enable) flag (bit 2) —** When set, branch trace messages are enabled. Thereafter, when the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 18.6.6, “Branch Trace Messages.”

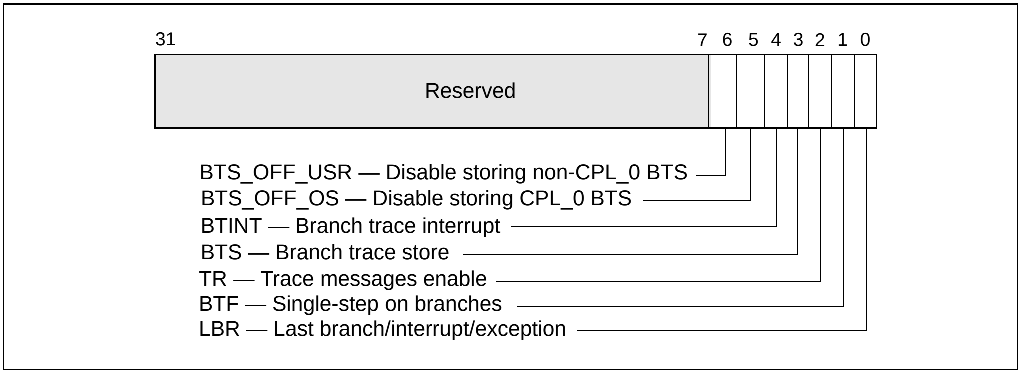
31
7 6 5 4 3 2 1 0
Reserved
BTS_OFF_USR — Disable storing non-CPL_0 BTS
BTS_OFF_OS — Disable storing CPL_0 BTS
BTINT — Branch trace interrupt
BTS — Branch trace store
TR — Trace messages enable
BTF — Single-step on branches
LBR — Last branch/interrupt/exception

Figure 18-5. MSR_DEBUGCTLA MSR for Pentium 4 and Intel Xeon Processors

- **BTS (branch trace store) flag (bit 3) —** When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 18.16.5, “DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 4) —** When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 18.6.8, “Branch Trace Store (BTS).”
- **BTS_OFF_OS (disable ring 0 branch trace store) flag (bit 5) —** When set, enables the BTS facilities to skip sending/logging CPL_0 BTMs to the memory-resident BTS buffer. See Section 18.6.1, “CPL-Qualified Branch Trace Mechanism.”
- **BTS_OFF_USR (disable ring 0 branch trace store) flag (bit 6) —** When set, enables the BTS facilities to skip sending/logging non-CPL_0 BTMs to the memory-resident BTS buffer. See Section 18.6.1, “CPL-Qualified Branch Trace Mechanism.”

18.6.3 LBR Stack for Processors Based on Intel NetBurst Microarchitecture

The LBR stack is made up of LBR MSRs that are treated by the processor as a circular stack. The TOS pointer (MSR_LASTBRANCH_TOS MSR) points to the LBR MSR (or LBR MSR pair) that contains the most recent (last) branch record placed on the stack. Prior to placing a new branch record on the stack, the TOS is incremented by 1. When the TOS pointer reaches it maximum value, it wraps around to 0. See Table 18-4 and Figure 18-5.

The registers in the LBR MSR stack and the MSR_LASTBRANCH_TOS MSR are read-only and can be read using the RDMSR instruction.

Figure 18-6 shows the layout of a branch record in an LBR MSR (or MSR pair). Each branch record consists of two linear addresses, which represent the “from” and “to” instruction pointers for a branch, interrupt, or exception. The contents of the from and to addresses differ, depending on the source of the branch:

- **Taken branch** — If the record is for a taken branch, the "from" address is the address of the branch instruction and the "to" address is the target instruction of the branch.
- **Interrupt** — If the record is for an interrupt, the "from" address the return instruction pointer (RIP) saved for the interrupt and the "to" address is the address of the first instruction in the interrupt handler routine. The RIP is the linear address of the next instruction to be executed upon returning from the interrupt handler.
- **Exception** — If the record is for an exception, the "from" address is the linear address of the instruction that caused the exception to be generated and the "to" address is the address of the first instruction in the exception handler routine.

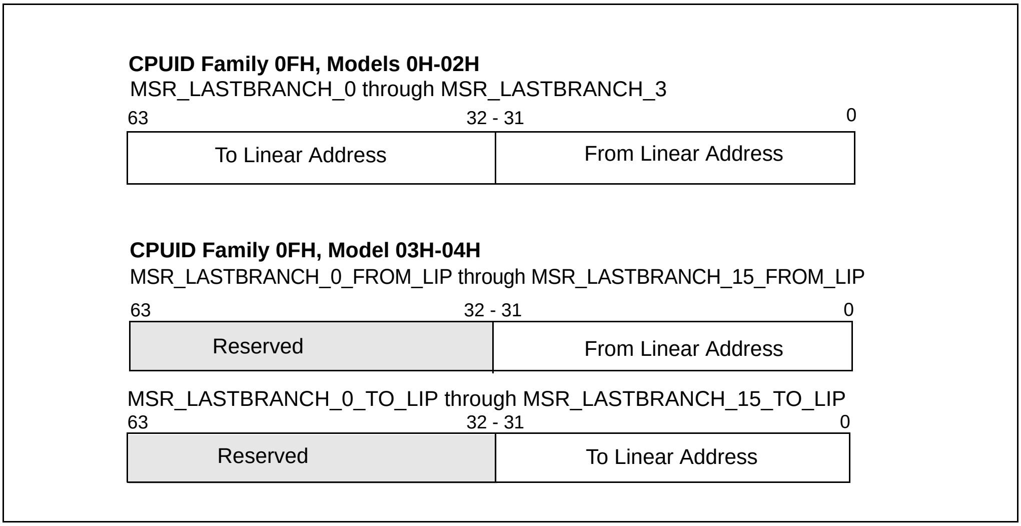
CPUID Family 0FH, Models 0H-02H
MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3
63
32 - 31
0
To Linear Address
From Linear Address
CPUID Family 0FH, Model 03H-04H
MSR_LASTBRANCH_0_FROM_LIP through MSR_LASTBRANCH_15_FROM_LIP
63
32 - 31
0
Reserved
From Linear Address
MSR_LASTBRANCH_0_TO_LIP through MSR_LASTBRANCH_15_TO_LIP
63
32 - 31
0
Reserved
To Linear Address

Figure 18-6. LBR MSR Branch Record Layout for the Pentium 4 and Intel Xeon Processor Family

Additional information is saved if an exception or interrupt occurs in conjunction with a branch instruction. If a branch instruction generates a trap type exception, two branch records are stored in the LBR stack: a branch record for the branch instruction followed by a branch record for the exception.

If a branch instruction is immediately followed by an interrupt, a branch record is stored in the LBR stack for the branch instruction followed by a record for the interrupt.

18.6.3.1 LBR Stack and Intel® 64 Processors

LBR MSRs are 64-bits. If IA-32e mode is disabled, only the lower 32-bits of the address is recorded. If IA-32e mode is enabled, the processor writes 64-bit values into the MSR.

In 64-bit mode, last branch records store 64-bit addresses; in compatibility mode, the upper 32-bits of last branch records are cleared.

18.6.4 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag in the MSR_DEBUGCTLA MSR is set, the processor automatically begins recording branch records for taken branches, interrupts, and exceptions (except for debug exceptions) in the LBR stack MSRs.

When the processor generates a a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler. This action does not clear previously stored LBR stack MSRs. The branch record for the last four taken branches, interrupts and/or exceptions are retained for analysis.

A debugger can use the linear addresses in the LBR stack to re-set breakpoints in the breakpoint address registers (DR0 through DR3). This allows a backward trace from the manifestation of a particular bug toward its source.

If the LBR flag is cleared and TR flag in the MSR_DEBUGCTLA MSR remains set, the processor will continue to update LBR stack MSRs. This is because BTM information must be generated from entries in the LBR stack (see 14.5.5). A #DB does not automatically clear the TR flag.

18.6.5 Single-Stepping on Branches, Exceptions, and Interrupts

When software sets both the BTF flag in the MSR_DEBUGCTLA MSR and the TF flag in the EFLAGS register, the processor generates a single-step debug exception the next time it takes a branch, services an interrupt, or generates an exception. This mechanism allows the debugger to single-step on control transfers caused by branches, interrupts, and exceptions. This “control-flow single stepping” helps isolate a bug to a particular block of code before instruction single-stepping further narrows the search. If the BTF flag is set when the processor generates a debug exception, the processor clears the BTF flag along with the TF flag. The debugger must reset the BTF and TF flags before resuming program execution to continue control-flow single stepping.

18.6.6 Branch Trace Messages

Setting the TR flag in the MSR_DEBUGCTLA (see Figure 18-5), IA32_DEBUGCTL (see Figure 18-7), or MSR_DEBUGCTLB (see Figure 18-9) MSR enables branch trace messages (BTMs). Thereafter, when the processor detects a branch, exception, or interrupt, it sends a branch record out on the system bus as a BTM. A debugging device that is monitoring the system bus can read these messages and synchronize operations with taken branch, interrupt, and exception events.

When interrupts or exceptions occur in conjunction with a taken branch, additional BTMs are sent out on the bus, as described in Section 18.6.4, “Monitoring Branches, Exceptions, and Interrupts.”

Setting this flag (BTS) alone can greatly reduce the performance of the processor. CPL-qualified branch trace storing mechanism can help mitigate the performance impact of sending/logging branch trace messages. See Section 18.6.1, “CPL-Qualified Branch Trace Mechanism.”

Unlike the P6 family processors, the Pentium 4, Atom, and Intel Xeon processors can collect branch records in the LBR stack MSRs while at the same time sending/storing BTMs when both the TR and LBR flags are set in the MSR_DEBUGCTLA/IA32_DEBUGCTL MSR.

18.6.7 Last Exception Records

The Pentium 4 and Intel Xeon processors provide two 32 bit MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate the functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in the P6 family processors. The MSR_LER_TO_LIP and MSR_LER_FROM_LIP MSRs contain a branch record for the last branch that the processor took prior to an exception or interrupt being generated.

18.6.7.1 Last Exception Records and Intel 64 Architecture

The MSRs that store last exception records are 64-bits. If IA-32e mode is disabled, only the lower 32-bits of the address is recorded. If IA-32e mode is enabled, the processor writes 64-bit values into the MSR. In 64-bit mode, last exception records store 64-bit addresses; in compatibility mode, the upper 32-bits of last exception records are cleared.

18.6.8 Branch Trace Store (BTS)

A trace of taken branches, interrupts, and exceptions is useful for debugging code by providing a method of determining the decision path taken to reach a particular code location. The Pentium 4 and Intel Xeon processors provide a mechanism for capturing records of taken branches, interrupts, and exceptions and saving them in the last branch record (LBR) stack MSRs and/or sending them out onto the system bus as BTMs. The branch trace store (BTS) mechanism provides the additional capability of saving the branch records in a memory-resident BTS buffer, which is part of the DS save area. The BTS buffer can be configured to be circular so that the most recent branch records are always available or it can be configured to generate an interrupt when the buffer is nearly full so that all the branch records can be saved. See Section 18.16.5, "DS Save Area."

18.6.8.1 Detection of the BTS Facilities

The DS feature flag (bit 21) returned by the CPUID instruction indicates (when set) the availability of the DS mechanism in the processor, which supports the BTS (and PEBS) facilities. When this bit is set, the following BTS facilities are available:

- The BTS_UNAVAILABLE flag in the IA32_MISC_ENABLE MSR indicates (when clear) the availability of the BTS facilities, including the ability to set the BTS and BTINT bits in the MSR_DEBUGCTLA MSR.
- The IA32_DS_AREA MSR can be programmed to point to the DS save area.

18.6.8.2 Setting Up the DS Save Area

To save branch records with the BTS buffer, the DS save area must first be set up in memory as described in the following procedure. See Section 18.6.8.3, "Setting Up the BTS Buffer," and Section 18.16.8.3, "Setting Up the PEBS Buffer," for instructions for setting up a BTS buffer and/or a PEBS buffer, respectively, in the DS save area:

1. Create the DS buffer management information area in memory (see Section 18.16.5, "DS Save Area," and Section 18.16.5.1, "DS Save Area and IA-32e Mode Operation"). Also see the additional notes in this section.
2. Write the base linear address of the DS buffer management area into the IA32_DS_AREA MSR.
3. Set up the performance counter entry in the xAPIC LVT for fixed delivery and edge sensitive. See Section 9.5.1, "Local Vector Table."
4. Establish an interrupt handler in the IDT for the vector associated with the performance counter entry in the xAPIC LVT.
5. Write an interrupt service routine to handle the interrupt. See Section 18.6.8.5, "Writing the DS Interrupt Service Routine."

The following restrictions should be applied to the DS save area.

- The three DS save area sections should be allocated from a non-paged pool, and marked accessed and dirty. It is the responsibility of the operating system to keep the pages that contain the buffer present and to mark them accessed and dirty. The implication is that the operating system cannot do "lazy" page-table entry propagation for these pages.
- The DS save area can be larger than a page, but the pages must be mapped to contiguous linear addresses. The buffer may share a page, so it need not be aligned on a 4-KByte boundary. For performance reasons, the base of the buffer must be aligned on a doubleword boundary and should be aligned on a cache line boundary.

- It is recommended that the buffer size for the BTS buffer and the PEBS buffer be an integer multiple of the corresponding record sizes.
- The precise event records buffer should be large enough to hold the number of precise event records that can occur while waiting for the interrupt to be serviced.
- The DS save area should be in kernel space. It must not be on the same page as code, to avoid triggering self-modifying code actions.
- There are no memory type restrictions on the buffers, although it is recommended that the buffers be designated as WB memory type for performance considerations.
- Either the system must be prevented from entering A20M mode while DS save area is active, or bit 20 of all addresses within buffer bounds must be 0.
- Pages that contain buffers must be mapped to the same physical addresses for all processes, such that any change to control register CR3 will not change the DS addresses.
- The DS save area is expected to used only on systems with an enabled APIC. The LVT Performance Counter entry in the APCI must be initialized to use an interrupt gate instead of the trap gate.

18.6.8.3 Setting Up the BTS Buffer

Three flags in the MSR_DEBUGCTLA MSR (see Table 18-5), IA32_DEBUGCTL (see Figure 18-7), or MSR_DEBUGCTLB (see Figure 18-9) control the generation of branch records and storing of them in the BTS buffer; these are TR, BTS, and BTINT. The TR flag enables the generation of BTMs. The BTS flag determines whether the BTMs are sent out on the system bus (clear) or stored in the BTS buffer (set). BTMs cannot be simultaneously sent to the system bus and logged in the BTS buffer. The BTINT flag enables the generation of an interrupt when the BTS buffer is full. When this flag is clear, the BTS buffer is a circular buffer.

Table 18-5. MSR_DEBUGCTLA, IA32_DEBUGCTL, MSR_DEBUGCLTB Flag Encodings

TR	BTS	BTINT	Description
0	X	X	Branch trace messages (BTMs) off
1	0	X	Generate BTMs
1	1	0	Store BTMs in the BTS buffer, used here as a circular buffer
1	1	1	Store BTMs in the BTS buffer, and generate an interrupt when the buffer is nearly full

The following procedure describes how to set up a Pentium 4 or Intel Xeon processor to collect branch records in the BTS buffer in the DS save area:

1. Place values in the BTS buffer base, BTS index, BTS absolute maximum, and BTS interrupt threshold fields of the DS buffer management area to set up the BTS buffer in memory.
2. Set the TR and BTS flags in the MSR_DEBUGCTLA MSR (or IA32_DEBUGCTL for Intel Core Solo and Intel Core Duo processors; or MSR_DEBUGCTLB for Pentium M processors).
3. Either clear the BTINT flag in the MSR_DEBUGCTLA MSR (to set up a circular BTS buffer) or set the BTINT flag (to generate an interrupt when the BTS buffer is nearly full). For Intel Core Solo and Intel Core Duo processors, do the same in IA32_DEBUGCTL; in MSR_DEBUGCTLB for Pentium M processors.

NOTES

If the buffer size is set to less than the minimum allowable value (i.e. BTS absolute maximum < 1 + size of BTS record), the results of BTS is undefined.

In order to prevent generating an interrupt, when working with circular BTS buffer, SW need to set BTS interrupt threshold to a value greater than BTS absolute maximum (fields of the DS buffer management area). It's not enough to clear the BTINT flag itself only.

18.6.8.4 Setting Up CPL-Qualified BTS

If the processor supports CPL-qualified last branch recording mechanism, the generation of branch records and storing of them in the BTS buffer are determined by: TR, BTS, BTS_OFF_OS, BTS_OFF_USR, and BTINT. The encoding of these five bits are shown in Table 18-6.

Table 18-6. CPL-Qualified Branch Trace Store Encodings

TR	BTS	BTS_OFF_OS	BTS_OFF_USR	BTINT	Description
0	X	X	X	X	Branch trace messages (BTMs) off
1	0	X	X	X	Generates BTMs but do not store BTMs
1	1	0	0	0	Store all BTMs in the BTS buffer, used here as a circular buffer
1	1	1	0	0	Store BTMs with CPL > 0 in the BTS buffer
1	1	0	1	0	Store BTMs with CPL = 0 in the BTS buffer
1	1	1	1	X	Generate BTMs but do not store BTMs
1	1	0	0	1	Store all BTMs in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	1	0	1	Store BTMs with CPL > 0 in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	0	1	1	Store BTMs with CPL = 0 in the BTS buffer; generate an interrupt when the buffer is nearly full

18.6.8.5 Writing the DS Interrupt Service Routine

The BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector and interrupt service routine (called the debug store interrupt service routine or DS ISR). To handle BTS, non-precise event-based sampling, and PEBS interrupts: separate handler routines must be included in the DS ISR. Use the following guidelines when writing a DS ISR to handle BTS, non-precise event-based sampling, and/or PEBS interrupts.

- The DS interrupt service routine (ISR) must be part of a kernel driver and operate at a current privilege level of 0 to secure the buffer storage area.
- Because the BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector, the DS ISR must check for all the possible causes of interrupts from these facilities and pass control on to the appropriate handler.

  BTS and PEBS buffer overflow would be the sources of the interrupt if the buffer index matches/exceeds the interrupt threshold specified. Detection of non-precise event-based sampling as the source of the interrupt is accomplished by checking for counter overflow.

- There must be separate save areas, buffers, and state for each processor in an MP system.
- Upon entering the ISR, branch trace messages and PEBS should be disabled to prevent race conditions during access to the DS save area. This is done by clearing TR flag in the MSR_DEBUGCTLA MSR and by clearing the precise event enable flag in the MSR_PEBS_ENABLE MSR. These settings should be restored to their original values when exiting the ISR.
- The processor will not disable the DS save area when the buffer is full and the circular mode has not been selected. The current DS setting must be retained and restored by the ISR on exit.
- After reading the data in the appropriate buffer, up to but not including the current index into the buffer, the ISR must reset the buffer index to the beginning of the buffer. Otherwise, everything up to the index will look like new entries upon the next invocation of the ISR.
- The ISR must clear the mask bit in the performance counter LVT entry.
- The ISR must re-enable the CCCR's ENABLE bit if it is servicing an overflow PMI due to PEBS.
- The Pentium 4 Processor and Intel Xeon Processor mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

18.7 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS)

Intel Core Solo and Intel Core Duo processors provide last branch interrupt and exception recording. This capability is almost identical to that found in Pentium 4 and Intel Xeon processors. There are differences in the stack and in some MSR names and locations.

Note the following:

- **IA32_DEBUGCTL MSR —** Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. IA32_DEBUGCTL MSR is located at register address 01D9H.

  See Figure 18-7 the layout and the entries below for a description of the flags:

  — **LBR (last branch/interrupt/exception) flag (bit 0) —** When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the "Last Branch Record (LBR) Stack" below.

- **BTF (single-step on branches) flag (bit 1) —** When set, the processor treats the TF flag in the EFLAGS register as a "single-step on branches" flag rather than a "single-step on instructions" flag. This mechanism allows single-stepping the processor on taken branches, interrupts, and exceptions. See Section 18.6.5, "Single-Stepping on Branches, Exceptions, and Interrupts," for more information about the BTF flag.
- **TR (trace message enable) flag (bit 6) —** When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 18.6.6, "Branch Trace Messages," for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7) —** When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 18.16.5, "DS Save Area."
- **BTINT (branch trace interrupt) flag (bits 8) —** When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 18.6.8, "Branch Trace Store (BTS)," for a description of this mechanism.

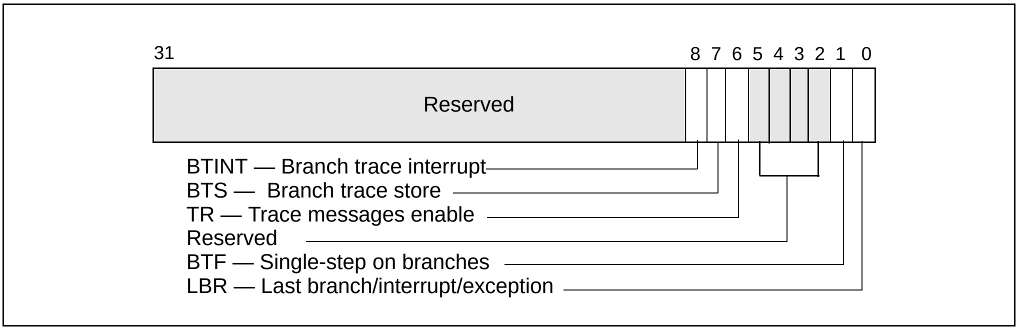
31
8 7 6 5 4 3 2 1 0
Reserved
BTINT — Branch trace interrupt
BTS — Branch trace store
TR — Trace messages enable
Reserved
BTF — Single-step on branches
LBR — Last branch/interrupt/exception

Figure 18-7. IA32_DEBUGCTL MSR for Intel Core Solo and Intel Core Duo Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 18.6.8, "Branch Trace Store (BTS)."
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the 'from' address, bits 63-32 hold the 'to' address (MSR addresses start at 40H). See Figure 18-8.

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Intel Core Solo and Intel Core Duo processors, this MSR is located at register address 01C9H.

For compatibility, the Intel Core Solo and Intel Core Duo processors provide two 32-bit MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

For details, see Section 18.6, “Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture),” and Appendix B.5, “MSRs In Intel® Core™ Solo and Intel® Core™ Duo Processors.”

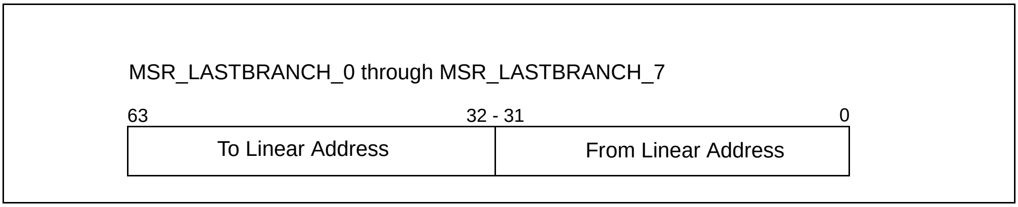

Figure 18-8. LBR Branch Record Layout for the Intel Core Solo and Intel Core Duo Processor

18.8 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PENTIUM M PROCESSORS)

Like the Pentium 4 and Intel Xeon processor family, Pentium M processors provide last branch interrupt and exception recording. The capability operates almost identically to that found in Pentium 4 and Intel Xeon processors. There are differences in the shape of the stack and in some MSR names and locations. Note the following:

- **MSR_DEBUGCTLB MSR —** Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. For Pentium M processors, this MSR is located at register address 01D9H. See Figure 18-9 and the entries below for a description of the flags.
  - **LBR (last branch/interrupt/exception) flag (bit 0) —** When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the “Last Branch Record (LBR) Stack” bullet below.
  - **BTF (single-step on branches) flag (bit 1) —** When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows

single-stepping the processor on taken branches, interrupts, and exceptions. See Section 18.6.5, “Single-Stepping on Branches, Exceptions, and Interrupts,” for more information about the BTF flag.

- **PB*i* (performance monitoring/breakpoint pins) flags (bits 5-2) —** When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding BP*i*# pin when a breakpoint match occurs. When a PB*i* flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
- **TR (trace message enable) flag (bit 6) —** When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 18.6.6, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7) —** When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 18.16.5, “DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 8) —** When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 18.6.8, “Branch Trace Store (BTS),” for a description of this mechanism.

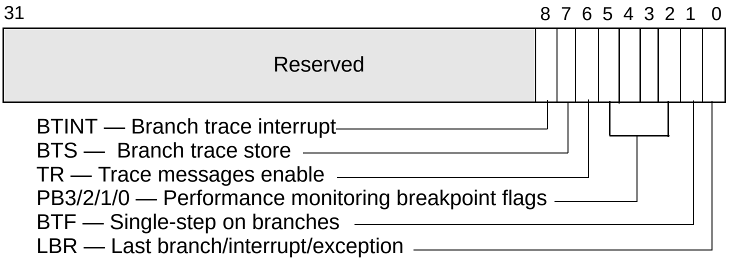
31
8 7 6 5 4 3 2 1 0
Reserved
BTINT — Branch trace interrupt
BTS — Branch trace store
TR — Trace messages enable
PB3/2/1/0 — Performance monitoring breakpoint flags
BTF — Single-step on branches
LBR — Last branch/interrupt/exception

Figure 18-9. MSR_DEBUGCTLB MSR for Pentium M Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 18.6.8, “Branch Trace Store (BTS).”
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the ‘from’

address, bits 63-32 hold the 'to' address. For Pentium M Processors, these pairs are located at register addresses 040H-047H. See Figure 18-10.

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Pentium M Processors, this MSR is located at register address 01C9H.

For compatibility, the Pentium M processor provides two 32-bit MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate the functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

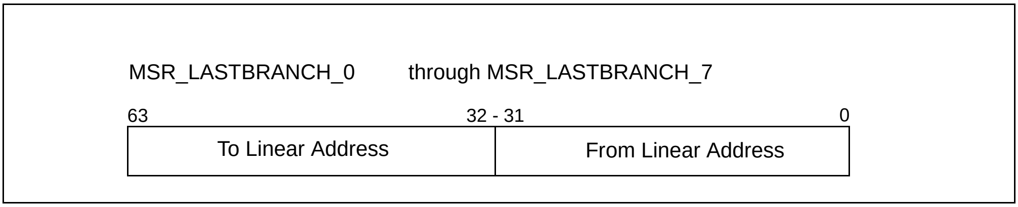

Figure 18-10. LBR Branch Record Layout for the Pentium M Processor

For more detail on these capabilities, see Section 18.6, "Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)," and Appendix B.6, "MSRs In the Pentium M Processor."

18.9 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (P6 FAMILY PROCESSORS)

The P6 family processors provide five MSRs for recording the last branch, interrupt, or exception taken by the processor: DEBUGCTLMSR, LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP. These registers can be used to collect last branch records, to set breakpoints on branches, interrupts, and exceptions, and to single-step from one branch to the next.

See Appendix B, "Model-Specific Registers (MSRs)," for a detailed description of each of the last branch recording MSRs.

18.9.1 DEBUGCTLMSR Register

The version of the DEBUGCTLMSR register found in the P6 family processors enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode. A protected-mode operating system procedure is required to provide user access to this register. Figure 18-11 shows the flags in the DEBUGCTLMSR register for the P6 family processors. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0) —** When set, the processor records the source and target addresses (in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs) for the last branch and the last exception or interrupt taken by the processor prior to a debug exception being generated. The processor clears this flag whenever a debug exception, such as an instruction or data breakpoint or single-step trap occurs.

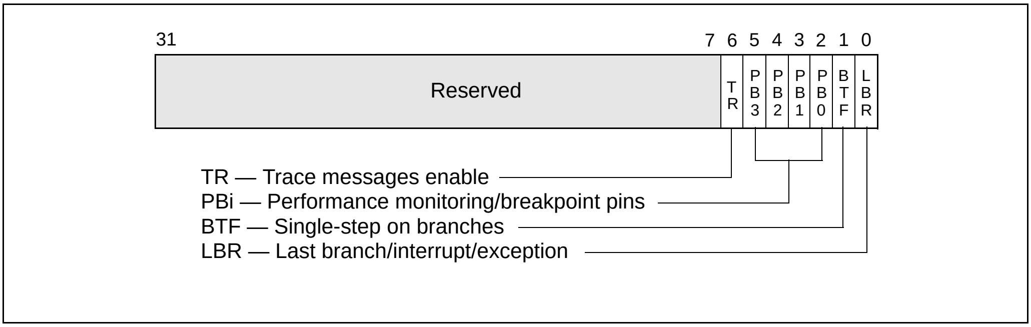
31
7
6
5
4
3
2
1
0
Reserved
TR
PB3
PB2
PB1
PB0
BTF
LBR
TR — Trace messages enable
PBi — Performance monitoring/breakpoint pins
BTF — Single-step on branches
LBR — Last branch/interrupt/exception

Figure 18-11. DEBUGCTLMSR Register (P6 Family Processors)

- **BTF (single-step on branches) flag (bit 1) —** When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag. See Section 18.6.5, “Single-Stepping on Branches, Exceptions, and Interrupts.”
- **PB*i* (performance monitoring/breakpoint pins) flags (bits 2 through 5) —** When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding BP*i*# pin when a breakpoint match occurs. When a PB*i* flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
- **TR (trace message enable) flag (bit 6) —** When set, trace messages are enabled as described in Section 18.6.6, “Branch Trace Messages.” Setting this flag greatly reduces the performance of the processor. When trace messages are enabled, the values stored in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are undefined.

18.9.2 Last Branch and Last Exception MSRs

The LastBranchToIP and LastBranchFromIP MSRs are 32-bit registers for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated. When a branch occurs, the processor loads the address of the branch instruction into the LastBranchFromIP MSR and loads the target address for the branch into the LastBranchToIP MSR.

When an interrupt or exception occurs (other than a debug exception), the address of the instruction that was interrupted by the exception or interrupt is loaded into the LastBranchFromIP MSR and the address of the exception or interrupt handler that is called is loaded into the LastBranchToIP MSR.

The LastExceptionToIP and LastExceptionFromIP MSRs (also 32-bit registers) record the instruction pointers for the last branch that the processor took prior to an exception or interrupt being generated. When an exception or interrupt occurs, the contents of the LastBranchToIP and LastBranchFromIP MSRs are copied into these registers before the to and from addresses of the exception or interrupt are recorded in the LastBranchToIP and LastBranchFromIP MSRs.

These registers can be read using the RDMSR instruction.

Note that the values stored in the LastBranchToIP, LastBranchFromIP, LastException-ToIP, and LastExceptionFromIP MSRs are offsets into the current code segment, as opposed to linear addresses, which are saved in last branch records for the Pentium 4 and Intel Xeon processors.

18.9.3 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag in the DEBUGCTLMSR register is set, the processor automatically begins recording branches that it takes, exceptions that are generated (except for debug exceptions), and interrupts that are serviced. Each time a branch, exception, or interrupt occurs, the processor records the to and from instruction pointers in the LastBranchToIP and LastBranchFromIP MSRs. In addition, for interrupts and exceptions, the processor copies the contents of the LastBranchToIP and LastBranch-FromIP MSRs into the LastExceptionToIP and LastExceptionFromIP MSRs prior to recording the to and from addresses of the interrupt or exception.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler, but does not touch the last branch and last exception MSRs. The addresses for the last branch, interrupt, or exception taken are thus retained in the LastBranchToIP and LastBranchFromIP MSRs and the addresses of the last branch prior to an interrupt or exception are retained in the LastExceptionToIP, and LastExceptionFromIP MSRs.

The debugger can use the last branch, interrupt, and/or exception addresses in combination with code-segment selectors retrieved from the stack to reset breakpoints in the breakpoint-address registers (DR0 through DR3), allowing a backward trace from the manifestation of a particular bug toward its source. Because the instruction pointers recorded in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are offsets into a code segment, software must determine the segment base address of the code segment associated with the control transfer to calculate the linear address to be placed in the breakpoint-address registers. The segment base address can be determined by reading the segment selector for the code segment from the stack and using it to locate the segment descriptor for the segment in the GDT or LDT. The segment base address can then be read from the segment descriptor.

Before resuming program execution from a debug-exception handler, the handler must set the LBR flag again to re-enable last branch and last exception/interrupt recording.

18.10 TIME-STAMP COUNTER

The Intel 64 and IA-32 architectures (beginning with the Pentium processor) define a time-stamp counter mechanism that can be used to monitor and identify the relative time occurrence of processor events. The counter's architecture includes the following components:

- **TSC flag —** A feature bit that indicates the availability of the time-stamp counter. The counter is available in an if the function CPUID.1:EDX.TSC[bit 4] = 1.
- **IA32_TIME_STAMP_COUNTER MSR** (called TSC MSR in P6 family and Pentium processors) **—** The MSR used as the counter.
- **RDTSC instruction —** An instruction used to read the time-stamp counter.
- **TSD flag —** A control register flag is used to enable or disable the time-stamp counter (enabled if CR4.TSD[bit 2] = 1).

The time-stamp counter (as implemented in the P6 family, Pentium, Pentium M, Pentium 4, Intel Xeon, Intel Core Solo and Intel Core Duo processors and later processors) is a 64-bit counter that is set to 0 following a RESET of the processor. Following a RESET, the counter increments even when the processor is halted by the HLT instruction or the external STPCLK# pin. Note that the assertion of the external DPSLP# pin may cause the time-stamp counter to stop.

Processor families increment the time-stamp counter differently:

- For Pentium M processors (family [06H], models [09H, 0DH]); for Pentium 4 processors, Intel Xeon processors (family [0FH], models [00H, 01H, or 02H]); and for P6 family processors: the time-stamp counter increments with every internal processor clock cycle.

  The internal processor clock cycle is determined by the current core-clock to bus-clock ratio. Intel® SpeedStep® technology transitions may also impact the processor clock.
- For Pentium 4 processors, Intel Xeon processors (family [0FH], models [03H and higher]); for Intel Core Solo and Intel Core Duo processors (family [06H], model [0EH]); for the Intel Xeon processor 5100 series and Intel Core 2 Duo processors (family [06H], model [0FH]); for Intel Core 2 and Intel Xeon processors (family

[06H], display_model [17H]); for Intel Atom processors (family [06H], display_model [1CH]): the time-stamp counter increments at a constant rate. That rate may be set by the maximum core-clock to bus-clock ratio of the processor or may be set by the maximum resolved frequency at which the processor is booted. The maximum resolved frequency may differ from the maximum qualified frequency of the processor, see Section 18.18.5 for more detail.

The specific processor configuration determines the behavior. Constant TSC behavior ensures that the duration of each clock tick is uniform and supports the use of the TSC as a wall clock timer even if the processor core changes frequency. This is the architectural behavior moving forward.

NOTE

To determine average processor clock frequency, Intel recommends the use of EMON logic to count processor core clocks over the period of time for which the average is required. See Section 18.18, “Counting Clocks,” and Appendix A, “Performance-Monitoring Events,” for more information.

The RDTSC instruction reads the time-stamp counter and is guaranteed to return a monotonically increasing unique value whenever executed, except for a 64-bit counter wraparound. Intel guarantees that the time-stamp counter will not wraparound within 10 years after being reset. The period for counter wrap is longer for Pentium 4, Intel Xeon, P6 family, and Pentium processors.

Normally, the RDTSC instruction can be executed by programs and procedures running at any privilege level and in virtual-8086 mode. The TSD flag allows use of this instruction to be restricted to programs and procedures running at privilege level 0. A secure operating system would set the TSD flag during system initialization to disable user access to the time-stamp counter. An operating system that disables user access to the time-stamp counter should emulate the instruction through a user-accessible programming interface.

The RDTSC instruction is not serializing or ordered with other instructions. It does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDTSC instruction operation is performed.

The RDMSR and WRMSR instructions read and write the time-stamp counter, treating the time-stamp counter as an ordinary MSR (address 10H). In the Pentium 4, Intel Xeon, and P6 family processors, all 64-bits of the time-stamp counter are read using RDMSR (just as with RDTSC). When WRMSR is used to write the time-stamp counter on processors before family [0FH], models [03H, 04H]: only the low-order 32-bits of the time-stamp counter can be written (the high-order 32 bits are cleared to 0). For family [0FH], models [03H, 04H, 06H]; for family [06H]], model [0EH, 0FH]; ; for family [06H]], display_model [17H, 1CH]: all 64 bits are writable.

18.11 PERFORMANCE MONITORING OVERVIEW

Performance monitoring was introduced in the Pentium processor with a set of model-specific performance-monitoring counter MSRs. These counters permit selection of processor performance parameters to be monitored and measured. The information obtained from these counters can be used for tuning system and compiler performance.

In Intel P6 family of processors, the performance monitoring mechanism was enhanced to permit a wider selection of events to be monitored and to allow greater control events to be monitored. Next, Pentium 4 and Intel Xeon processors introduced a new performance monitoring mechanism and new set of performance events.

The performance monitoring mechanisms and performance events defined for the Pentium, P6 family, Pentium 4, and Intel Xeon processors are not architectural. They are all model specific (not compatible among processor families). Intel Core Solo and Intel Core Duo processors support a set of architectural performance events and a set of non-architectural performance events. Processors based on Intel Core microarchitecture and Intel Atom microarchitecture support enhanced architectural performance events and non-architectural performance events.

Starting with Intel Core Solo and Intel Core Duo processors, there are two classes of performance monitoring capabilities. The first class supports events for monitoring performance using counting or sampling usage. These events are non-architectural and vary from one processor model to another. They are similar to those available in Pentium M processors. These non-architectural performance monitoring events are specific to the microarchitecture and may change with enhancements. They are discussed in Section 18.13, "Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)." Non-architectural events for a given microarchitecture can not be enumerated using CPUID; and they are listed in Appendix A, "Performance-Monitoring Events."

The second class of performance monitoring capabilities is referred to as architectural performance monitoring. This class supports the same counting and sampling usages, with a smaller set of available events. The visible behavior of architectural performance events is consistent across processor implementations. Availability of architectural performance monitoring capabilities is enumerated using the CPUID.0AH. These events are discussed in Section 18.12.

See also:

— Section 18.12, "Architectural Performance Monitoring"

— Section 18.13, "Performance Monitoring (Intel® Core™ Solo and Intel® Core™ Duo Processors)"

— Section 18.14, "Performance Monitoring (Processors based on Intel® Core™ Microarchitecture)"

— Section 18.15, "Performance Monitoring (Processors based on Intel® Atom™ Microarchitecture)"

- — Section 18.16, “Performance Monitoring (Processors Based on Intel NetBurst microarchitecture)”
- — Section 18.17, “Performance Monitoring and Hyper-Threading Technology in Processors Based on Intel NetBurst Microarchitecture”
- — Section 18.20, “Performance Monitoring and Dual-Core Technology”
- — Section 18.21, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache”
- — Section 18.23, “Performance Monitoring (P6 Family Processor)”
- — Section 18.24, “Performance Monitoring (Pentium Processors)”

18.12 ARCHITECTURAL PERFORMANCE MONITORING

Performance monitoring events are architectural when they behave consistently across microarchitectures. Intel Core Solo and Intel Core Duo processors introduced architectural performance monitoring. The feature provides a mechanism for software to enumerate performance events and provides configuration and counting facilities for events.

Architectural performance monitoring does allow for enhancement across processor implementations. The CPUID.0AH leaf provides version ID for each enhancement. Intel Core Solo and Intel Core Duo processors support base level functionality identified by version ID of 1. Processors based on Intel Core microarchitecture support, at a minimum, the base level functionality of architectural performance monitoring. Intel Core 2 Duo processor T 7700 and newer processors based on Intel Core microarchitecture support both the base level functionality and enhanced architectural performance monitoring identified by version ID of 2.

Intel Atom processor family supports the base level functionality, enhanced architectural performance monitoring identified by version ID of 2 and version ID of 3.

18.12.1 Architectural Performance Monitoring Version 1

Configuring an architectural performance monitoring event involves programming performance event select registers. There are a finite number of performance event select MSRs (IA32_PERFEVTSELx MSRs). The result of a performance monitoring event is reported in a performance monitoring counter (IA32_PMCx MSR). Performance monitoring counters are paired with performance monitoring select registers.

Performance monitoring select registers and counters are architectural in the following respects:

- Bit field layout of IA32_PERFEVTSELx is consistent across microarchitectures.
- Addresses of IA32_PERFEVTSELx MSRs remain the same across microarchitectures.

- Addresses of IA32_PMC MSRs remain the same across microarchitectures.
- Each logical processor has its own set of IA32_PERFEVTSELx and IA32_PMCx MSRs. Configuration facilities and counters are not shared between logical processors sharing a processor core.

Architectural performance monitoring provides a CPUID mechanism for enumerating the following information:

- Number of performance monitoring counters available in a logical processor (each IA32_PERFEVTSELx MSR is paired to the corresponding IA32_PMCx MSR)
- Number of bits supported in each IA32_PMCx
- Number of architectural performance monitoring events supported in a logical processor

Software can use CPUID to discover architectural performance monitoring availability (CPUID.0AH). The architectural performance monitoring leaf provides an identifier corresponding to the version number of architectural performance monitoring available in the processor.

The version identifier is retrieved by querying CPUID.0AH:EAX[bits 7:0] (see Chapter 3, “Instruction Set Reference, A-M,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*). If the version identifier is greater than zero, architectural performance monitoring capability is supported. Software queries the CPUID.0AH for the version identifier first; it then analyzes the value returned in CPUID.0AH.EAX, CPUID.0AH.EBX to determine the facilities available.

In the initial implementation of architectural performance monitoring; software can determine how many IA32_PERFEVTSELx/ IA32_PMCx MSR pairs are supported per core, the bit-width of PMC, and the number of architectural performance monitoring events available.

18.12.1.1 Architectural Performance Monitoring Version 1 Facilities

Architectural performance monitoring facilities include a set of performance monitoring counters and performance event select registers. These MSRs have the following properties:

- IA32_PMCx MSRs start at address 0C1H and occupy a contiguous block of MSR address space; the number of MSRs per logical processor is reported using CPUID.0AH.
- IA32_PERFEVTSELx MSRs start at address 186H and occupy a contiguous block of MSR address space. Each performance event select register is paired with a corresponding performance counter in the 0C1H address block.
- The bit width of an IA32_PMCx MSR is reported using the CPUID.0AH leaf. Bits beyond the width of the programmable counter are undefined, and are ignored when written to. In the initial implementation, the bit width for read operations is reported using CPUID; write operations are limited to the low 32 bits of registers.
- Bit field layout of IA32_PERFEVTSELx MSRs is defined architecturally.

See Figure 18-12 for the bit field layout of IA32_PERFEVTSELx MSRs. The bit fields are:

- **Event select field (bits 0 through 7) —** Selects the event logic unit used to detect microarchitectural conditions (see Table 18-7, for a list of architectural events and their 8-bit codes). The set of values for this field is defined architecturally; each value corresponds to an event logic unit for use with an architectural performance event. The number of architectural events is queried using CPUID.0AH:EAX. A processor may support only a subset of pre-defined values.

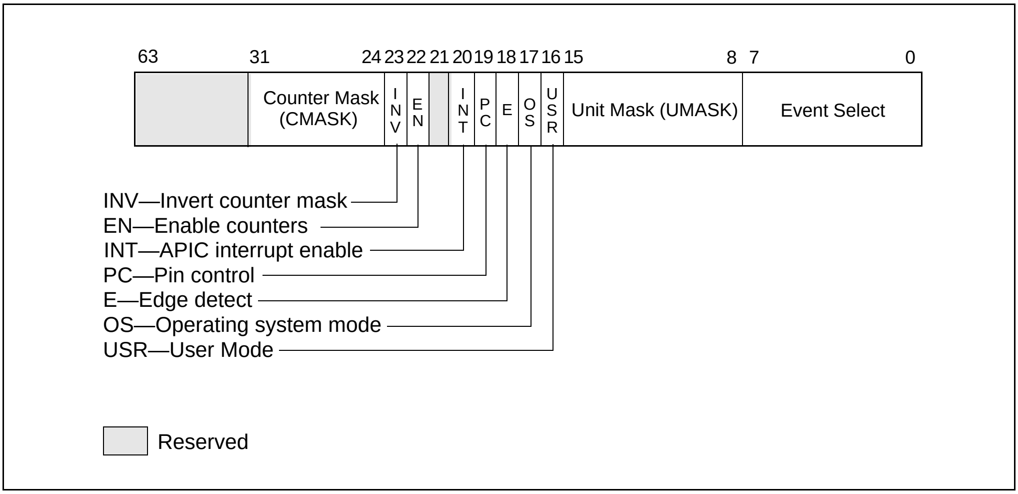

Figure 18-12. Layout of IA32_PERFEVTSELx MSRs

- **Unit mask (UMASK) field (bits 8 through 15)** — These bits qualify the condition that the selected event logic unit detects. Valid UMASK values for each event logic unit are specific to the unit. For each architectural performance event, its corresponding UMASK value defines a specific microarchitectural condition.

  A pre-defined microarchitectural condition associated with an architectural event may not be applicable to a given processor. The processor then reports only a subset of pre-defined architectural events. Pre-defined architectural events are listed in Table 18-7; support for pre-defined architectural events is enumerated using CPUID.0AH:EBX. Architectural performance events available in the initial implementation are listed in Table A-1.

- **USR (user mode) flag (bit 16) —** Specifies that the selected microarchitectural condition is counted only when the logical processor is operating at privilege levels 1, 2 or 3. This flag can be used with the OS flag.

- **OS (operating system mode) flag (bit 17) —** Specifies that the selected microarchitectural condition is counted only when the logical processor is operating at privilege level 0. This flag can be used with the USR flag.

- **E (edge detect) flag (bit 18)** — Enables (when set) edge detection of the selected microarchitectural condition. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by the other fields. The mechanism does not permit back-to-back assertions to be distinguished.

  This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).
- **PC (pin control) flag (bit 19)** — When set, the logical processor toggles the PM*i* pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PM*i* pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.
- **INT (APIC interrupt enable) flag (bit 20)** — When set, the logical processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — When set, performance counting is enabled in the corresponding performance-monitoring counter; when clear, the corresponding counter is disabled. The event logic unit for a UMASK must be disabled by setting IA32_PERFEVTSELx[bit 22] = 0, before writing to IA32_PMCx.
- **INV (invert) flag (bit 23)** — Inverts the result of the counter-mask comparison when set, so that both greater than and less than comparisons can be made.
- **Counter mask (CMASK) field (bits 24 through 31)** — When this field is not zero, a logical processor compares this mask to the events count of the detected microarchitectural condition during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented.

  This mask is intended for software to characterize microarchitectural conditions that can count multiple occurrences per cycle (for example, two or more instructions retired per clock; or bus queue occupations). If the counter-mask field is 0, then the counter is incremented each cycle by the event count associated with multiple occurrences.

18.12.2 Architectural Performance Monitoring Version 2

The enhanced features provided by architectural performance monitoring version 2 include the following:

- **Fixed-function performance counter register and associated control register** — Three of the architectural performance events are counted using three fixed-function MSRs (IA32_FIXED_CTR0 through IA32_FIXED_CTR2). Each of the fixed-function PMC can count only one architectural performance event.

  Configuring the fixed-function PMCs is done by writing to bit fields in the MSR (IA32_FIXED_CTR_CTRL) located at address 38DH. Unlike configuring performance events for general-purpose PMCs (IA32_PMCx) via UMASK field in (IA32_PERFEVTSELx), configuring, programming IA32_FIXED_CTR_CTRL for fixed-function PMCs do not require any UMASK.

- **Simplified event programming** — Most frequent operation in programming performance events are enabling/disabling event counting and checking the status of counter overflows. Architectural performance event version 2 provides three architectural MSRs:
  - — IA32_PERF_GLOBAL_CTRL allows software to enable/disable event counting of all or any combination of fixed-function PMCs (IA32_FIXED_CTRx) or any general-purpose PMCs via a single WRMSR.
  - — IA32_PERF_GLOBAL_STATUS allows software to query counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single RDMSR.
  - — IA32_PERF_GLOBAL_OVF_CTRL allows software to clear counter overflow conditions on any combination of fixed-function PMCs or general-purpose PMCs via a single WRMSR.

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 18-13 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:

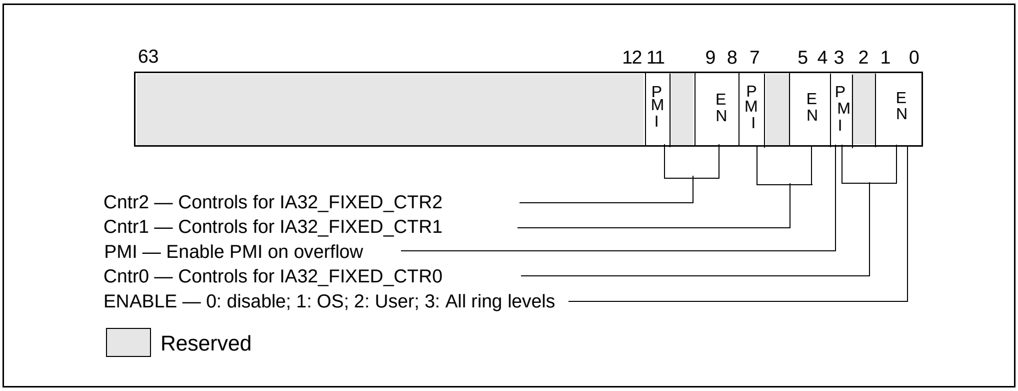
63
12
11
9
8
7
5
4
3
2
1
0
PMI
EN
PMI
EN
PMI
EN
Cntr2 — Controls for IA32_FIXED_CTR2
Cntr1 — Controls for IA32_FIXED_CTR1
PMI — Enable PMI on overflow
Cntr0 — Controls for IA32_FIXED_CTR0
ENABLE — 0: disable; 1: OS; 2: User; 3: All ring levels
Reserved

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.

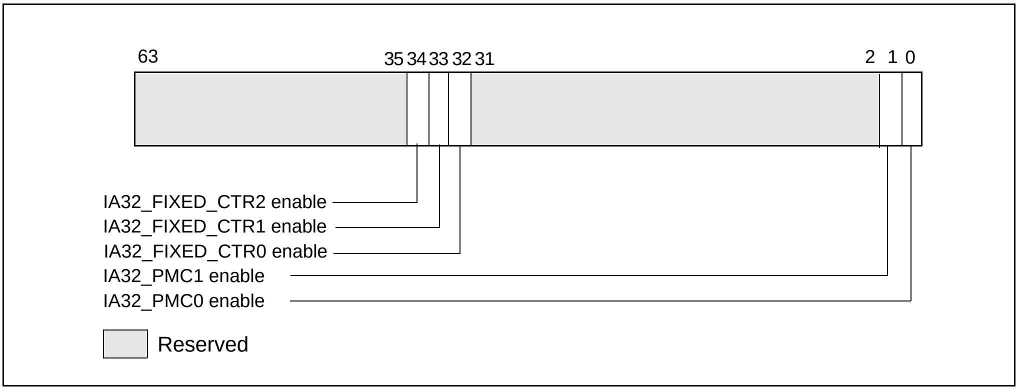

Figure 18-14. Layout of IA32_PERF_GLOBAL_CTRL MSR

The fixed-function performance counters supported by architectural performance version 2 is listed in Table 18-14, the pairing between each fixed-function performance counter to an architectural performance event is also shown.

IA32_PERF_GLOBAL_STATUS MSR provides single-bit status for software to query the overflow condition of each performance counter. The MSR also provides additional status bit to indicate overflow conditions when counters are programmed for precise-event-based sampling (PEBS). IA32_PERF_GLOBAL_STATUS MSR also provides a sticky bit to indicate changes to the state of performance monitoring hardware. Figure 18-15 shows the layout of IA32_PERF_GLOBAL_STATUS. A value of 1 in bits 0, 1, 32 through 34 indicates a counter overflow condition has occurred in the associated counter.

When a performance counter is configured for PEBS, overflow condition in the counter generates a performance-monitoring interrupt signaling a PEBS event. On a PEBS event, the processor stores data records into the buffer area (see Section 18.15.5), clears the counter overflow status., and sets the “OvfBuffer” bit in IA32_PERF_GLOBAL_STATUS.

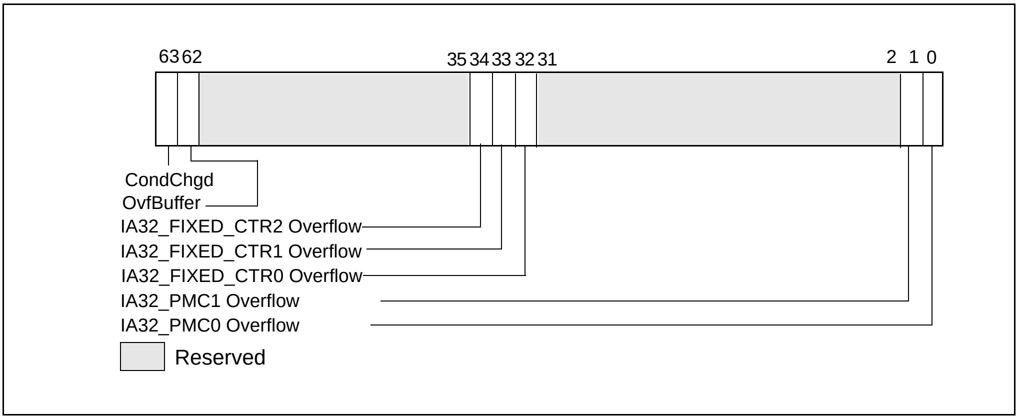

Figure 18-15. Layout of IA32_PERF_GLOBAL_STATUS MSR

IA32_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow indicator(s) of any general-purpose or fixed-function counters via a single WRMSR. Software should clear overflow indications when

- Setting up new values in the event select and/or UMASK field for counting or sampling
- Reloading counter values to continue sampling
- Disabling event counting or sampling.

The layout of IA32_PERF_GLOBAL_OVF_CTL is shown in Figure 18-16.

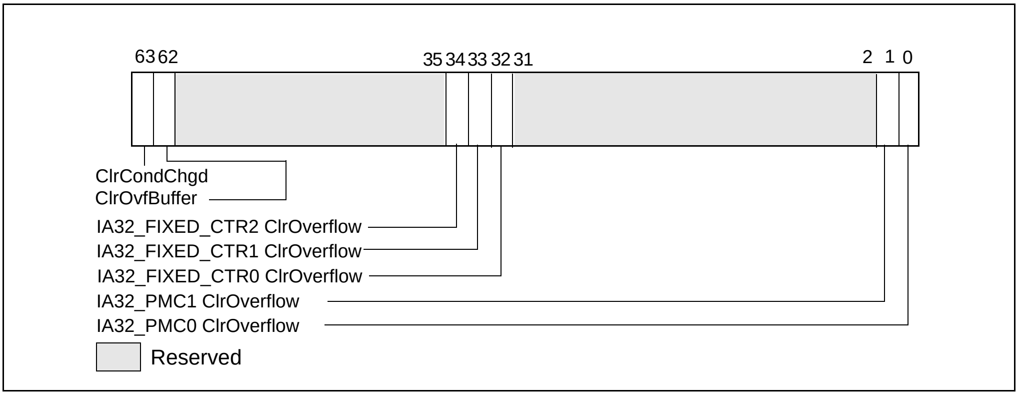

Figure 18-16. Layout of IA32_PERF_GLOBAL_OVF_CTRL MSR

18.12.2.2 Architectural Performance Monitoring Version 3 Facilities

The facilities provided by architectural performance monitoring version 1 and 2 are also supported by architectural performance monitoring version 3. Additionally version 3 provides enhancements to support a processor core comprising of more than one logical processor, i.e. a processor core supporting Hyper-Threading Technology or simultaneous multi-threading capability. Specifically,

- CPUID leaf 0AH provides enumeration mechanisms to query:
  - The number of general-purpose performance counters (IA32_PMCx), the bit width of general-purpose performance counters (see also Section 18.12.1.1)
  - The bit vector representing the set of architectural performance monitoring events supported (see Section 18.12.3)
  - The number of fixed-function performance counters, the bit width of fixed-function performance counters (see also Section 18.12.2.1).
- Each general-purpose performance counter IA32_PMCx (starting at MSR address 0C1H) is associated with a corresponding IA32_PERFEVTSELx MSR (starting at MSR address 186H). The Bit field layout of IA32_PERFEVTSELx MSRs is defined architecturally in Figure 18-17.

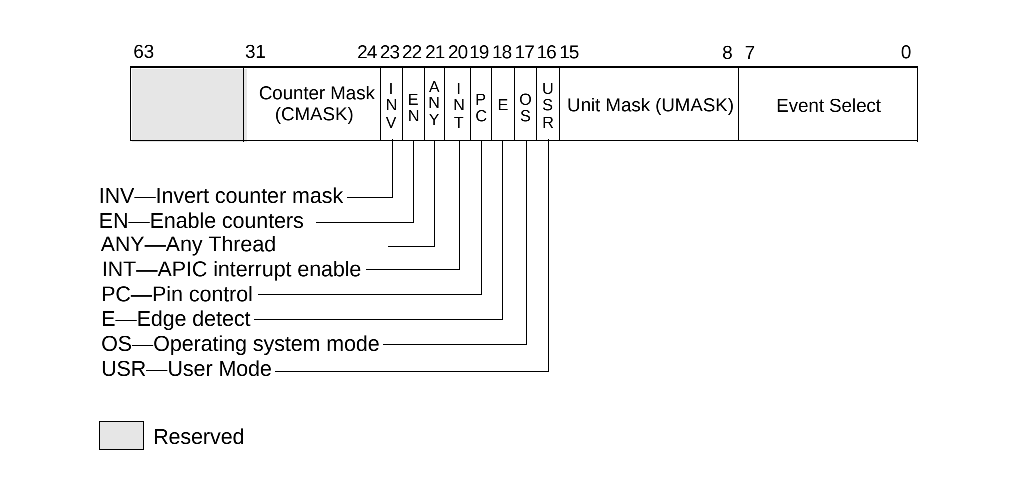

Figure 18-17. Layout of IA32_PERFEVTSELx MSRs Supporting Architectural Performance Monitoring Version 3

Bit 21 (AnyThread) of IA32_PERFEVTSELx is supported in architectural performance monitoring version 3. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the OS/USR setting of IA32_PERFEVTSELx) occurring across all logical processors sharing a processor core. When bit 21 is 0, the counter only increments the associated event conditions (including matching the thread's CPL with the

OS/USR setting of IA32_PERFEVTSELx) occurring in the logical processor which programmed the IA32_PERFEVTSELx MSR.

- Each fixed-function performance counter IA32_FIXED_CTRx (starting at MSR address 309H) is configured by a 4-bit control block in the IA32_PERF_FIXED_CTR_CTRL MSR. The control block also allow thread-specificity configuration using an AnyThread bit. The layout of IA32_PERF_FIXED_CTR_CTRL MSR is shown

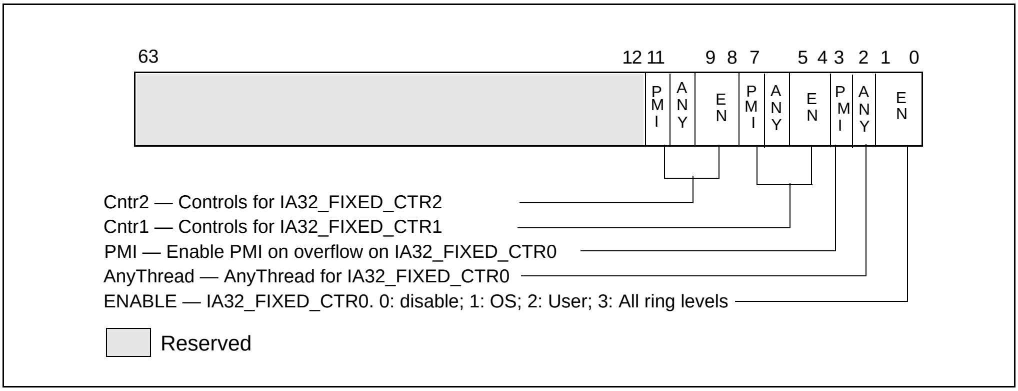
63
12 11
9 8 7
5 4 3 2 1 0
PMI
ANY
EN
PMI
ANY
EN
PMI
ANY
EN
Cntr2 — Controls for IA32_FIXED_CTR2
Cntr1 — Controls for IA32_FIXED_CTR1
PMI — Enable PMI on overflow on IA32_FIXED_CTR0
AnyThread — AnyThread for IA32_FIXED_CTR0
ENABLE — IA32_FIXED_CTR0. 0: disable; 1: OS; 2: User; 3: All ring levels
Reserved

Figure 18-18. Layout of IA32_FIXED_CTR_CTRL MSR Supporting Architectural Performance Monitoring Version 3

Each control block for a fixed-function performance counter provides a **AnyThread** (bit position 2 + 4*N, N= 0, 1, etc) bit. When set to 1, it enables counting the associated event conditions (including matching the thread's CPL with the ENABLE setting of the corresponding control block of IA32_PERF_FIXED_CTR_CTRL) occurring across all logical processors sharing a processor core. When a **AnyThread** bit is 0 in IA32_PERF_FIXED_CTR_CTRL, the corresponding fixed counter only increments the associated event conditions occurring in the logical processor which programmed the IA32_PERF_FIXED_CTR_CTRL MSR.

- The IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, IA32_PERF_GLOBAL_OVF_CTRL MSRs provide single-bit controls/status for each general-purpose and fixed-function performance counter. Figure 18-19 shows the layout of these MSR for N general-purpose performance counters (where N is reported by CPUID.0AH:EAX[15:8]) and three fixed-function counters.

  Note: Intel Atom processor family supports two general-purpose performance monitoring counters (i.e. N =2 in Figure 18-19), other processor families in Intel 64 architecture may support a different value of N in Figure 18-19. The number N is reported by CPUID.0AH:EAX[15:8]

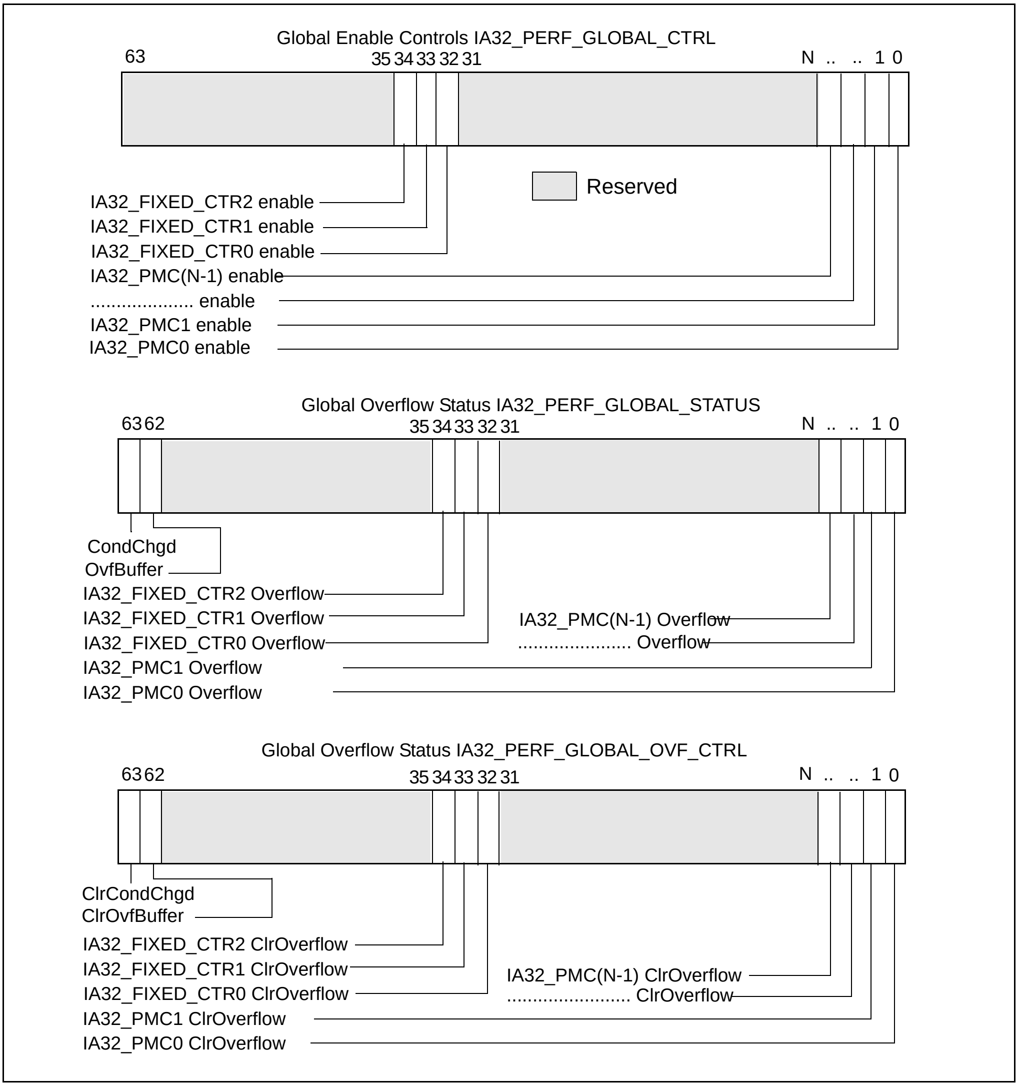

Figure 18-19. Layout of Global Performance Monitoring Control MSR

18.12.3 Pre-defined Architectural Performance Events

Table 18-7 lists architecturally defined events.

Table 18-7. UMask and Event Select Encodings for Pre-Defined Architectural Performance Events

Bit Position CPUID.AH.EBX	Event Name	UMask	Event Select
0	UnHalted Core Cycles	00H	3CH
1	Instruction Retired	00H	C0H
2	UnHalted Reference Cycles	01H	3CH
3	LLC Reference	4FH	2EH
4	LLC Misses	41H	2EH
5	Branch Instruction Retired	00H	C4H
6	Branch Misses Retired	00H	C5H

A processor that supports architectural performance monitoring may not support all the predefined architectural performance events (Table 18-7). The non-zero bits in CPUID.0AH:EBX indicate the events that are available.

The behavior of each architectural performance event is expected to be consistent on all processors that support that event. Minor variations between microarchitectures are noted below:

- **UnHalted Core Cycles —** Event select 3CH, Umask 00H

  This event counts core clock cycles when the clock signal on a specific core is running (not halted). The counter does not advance in the following conditions:

  — an ACPI C-state other than C0 for normal operation

  — HLT

  — STPCLK# pin asserted

  — being throttled by TM1

  — during the frequency switching phase of a performance state transition (see Chapter 13, “Power and Thermal Management”)

  The performance counter for this event counts across performance state transitions using different core clock frequencies

- **Instructions Retired —** Event select C0H, Umask 00H

  This event counts the number of instructions at retirement. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. An instruction with a REP prefix counts as one instruction (not per iteration). Faults before the retirement of the last micro-op of a multi-ops instruction are not counted.

  This event does not increment under VM-exit conditions. Counters continue counting during hardware interrupts, traps, and inside interrupt handlers.

- **UnHalted Reference Cycles —** Event select 3CH, Umask 01H

This event counts reference clock cycles while the clock signal on the core is running. The reference clock operates at a fixed frequency, irrespective of core frequency changes due to performance state transitions. Processors may implement this behavior differently. See Table A-4 and Table A-6 in Appendix A, “Performance-Monitoring Events.”

- **Last Level Cache References —** Event select 2EH, Umask 4FH

  This event counts requests originating from the core that reference a cache line in the last level cache. The event count may include speculation, but excludes cache line fills due to a hardware-prefetch.

  Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Last Level Cache Misses —** Event select 2EH, Umask 41H

  This event counts each cache miss condition for references to the last level cache. The event count may include speculation, but excludes cache line fills due to hardware-prefetch.

  Because cache hierarchy, cache sizes and other implementation-specific characteristics; value comparison to estimate performance differences is not recommended.

- **Branch Instructions Retired —** Event select C4H, Umask 00H

  This event counts branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction.

- **All Branch Mispredict Retired —** Event select C5H, Umask 00H

  This event counts mispredicted branch instructions at retirement. It counts the retirement of the last micro-op of a branch instruction in the architectural path of execution and experienced misprediction in the branch prediction hardware.

  Branch prediction hardware is implementation-specific across microarchitectures; value comparison to estimate performance differences is not recommended.

NOTE

Programming decisions or software precisians on functionality should not be based on the event values or dependent on the existence of performance monitoring events.

18.13 PERFORMANCE MONITORING (INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS)

In Intel Core Solo and Intel Core Duo processors, non-architectural performance monitoring events are programmed using the same facilities (see Figure 18-12) used for architectural performance events.

Non-architectural performance events use event select values that are model-specific. Event mask (Umask) values are also specific to event logic units. Some microarchitectural conditions detectable by a Umask value may have specificity related to processor topology (see Section 7.7, “Detecting Hardware Multi-Threading Support and Topology,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*). As a result, the unit mask field (for example, IA32_PERFEVTSELx[bits 15:8]) may contain sub-fields that specify topology information of processor cores.

The sub-field layout within the Umask field may support two-bit encoding that qualifies the relationship between a microarchitectural condition and the originating core. This data is shown in Table 18-8. The two-bit encoding for core-specificity is only supported for a subset of Umask values (see Appendix A, “Performance Monitoring Events”) and for Intel Core Duo processors. Such events are referred to as core-specific events.

Table 18-8. Core Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 15:14 Encoding	**Description**
11B	All cores
10B	Reserved
01B	This core
00B	Reserved

Some microarchitectural conditions allow detection specificity only at the boundary of physical processors. Some bus events belong to this category, providing specificity between the originating physical processor (a bus agent) versus other agents on the bus. Sub-field encoding for agent specificity is shown in Table 18-9.

Table 18-9. Agent Specificity Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13 Encoding	**Description**
0	This agent
1	Include all agents

Some microarchitectural conditions are detectable only from the originating core. In such cases, unit mask does not support core-specificity or agent-specificity encodings. These are referred to as core-only conditions.

Some microarchitectural conditions allow detection specificity that includes or excludes the action of hardware prefetches. A two-bit encoding may be supported to qualify hardware prefetch actions. Typically, this applies only to some L2 or bus events. The sub-field encoding for hardware prefetch qualification is shown in Table 18-10.

Table 18-10. HW Prefetch Qualification Encoding within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit 13:12 Encoding	**Description**
11B	All inclusive
10B	Reserved
01B	Hardware prefetch only
00B	Exclude hardware prefetch

Some performance events may (a) support none of the three event-specific qualification encodings (b) may support core-specificity and agent specificity simultaneously (c) or may support core-specificity and hardware prefetch qualification simultaneously. Agent-specificity and hardware prefetch qualification are mutually exclusive.

In addition, some L2 events permit qualifications that distinguish cache coherent states. The sub-field definition for cache coherency state qualification is shown in Table 18-11. If no bits in the MESI qualification sub-field are set for an event that requires setting MESI qualification bits, the event count will not increment.

Table 18-11. MESI Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 11:8	**Description**
Bit 11	Counts modified state
Bit 10	Counts exclusive state
Bit 9	Counts shared state
Bit 8	Counts Invalid state

18.14 PERFORMANCE MONITORING (PROCESSORS BASED ON INTEL® CORE™ MICROARCHITECTURE)

In addition to architectural performance monitoring, processors based on the Intel Core microarchitecture support non-architectural performance monitoring events.

Architectural performance events can be collected using general-purpose performance counters. Non-architectural performance events can be collected using general-purpose performance counters (coupled with two IA32_PERFEVTSELx MSRs for detailed event configurations), or fixed-function performance counters (see Section 18.14.1). IA32_PERFEVTSELx MSRs are architectural; their layout is shown in Figure 18-12. Starting with Intel Core 2 processor T 7700, fixed-function performance counters and associated counter control and status MSR becomes part of architectural performance monitoring version 2 facilities (see also Section 18.12.2).

Non-architectural performance events in processors based on Intel Core microarchitecture use event select values that are model-specific. Valid event mask (Umask) bits are listed in Appendix A. The UMASK field may contain sub-fields identical to those listed in Table 18-8, Table 18-9, Table 18-10, and Table 18-11. One or more of these sub-fields may apply to specific events on an event-by-event basis. Details are listed in Table A-4 in Appendix A, “Performance-Monitoring Events.”

In addition, the UMASK filed may also contain a sub-field that allows detection specificity related to snoop responses. Bits of the snoop response qualification sub-field are defined in Table 18-12.

Table 18-12. Bus Snoop Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 11:8	**Description**
Bit 11	HITM response
Bit 10	Reserved
Bit 9	HIT response
Bit 8	CLEAN response

There are also non-architectural events that support qualification of different types of snoop operation. The corresponding bit field for snoop type qualification are listed in Table 18-13.

Table 18-13. Snoop Type Qualification Definitions within a Non-Architectural Umask

IA32_PERFEVTSELx MSRs	
Bit Position 9:8	**Description**
Bit 9	CMP2I snoops
Bit 8	CMP2S snoops

No more than one sub-field of MESI, snoop response, and snoop type qualification sub-fields can be supported in a performance event.

NOTE

Software must write known values to the performance counters prior to enabling the counters. The content of general-purpose counters and fixed-function counters are undefined after INIT or RESET.

18.14.1 Fixed-function Performance Counters

Processors based on Intel Core microarchitecture provide three fixed-function performance counters. Bits beyond the width of the fixed counter are reserved and must be written as zeros. Model-specific fixed-function performance counters on processors that support Architectural Perfmon version 1 are 40 bits wide.

Each of the fixed-function counter is dedicated to count a pre-defined performance monitoring events. The performance monitoring events associated with fixed-function counters and the addresses of these counters are listed in Table 18-14.

Table 18-14. Association of Fixed-Function Performance Counters with Architectural Performance Events

Event Name	Fixed-Function PMC	PMC Address
INSTR_RETIRED.ANY	MSR_PERF_FIXED_CTR0/IA32_FIXED_CTR0	309H
CPU_CLK_UNHALTED.CORE	MSR_PERF_FIXED_CTR1//IA32_FIXED_CTR1	30AH
CPU_CLK_UNHALTED.REF	MSR_PERF_FIXED_CTR2//IA32_FIXED_CTR2	30BH

Programming the fixed-function performance counters does not involve any of the IA32_PERFEVTSELx MSRs, and does not require specifying any event masks. Instead, the MSR MSR_PERF_FIXED_CTR_CTRL provides multiple sets of 4-bit fields; each 4-bit field controls the operation of a fixed-function performance counter (PMC). See Figures 18-20. Two sub-fields are defined for each control. See Figure 18-20; bit fields are:

- **Enable field (low 2 bits in each 4-bit control) —** When bit 0 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring 0.

  When bit 1 is set, performance counting is enabled in the corresponding fixed-function performance counter to increment when the target condition associated with the architecture performance event occurs at ring greater than 0.

  Writing 0 to both bits stops the performance counter. Writing 11B causes the counter to increment irrespective of privilege levels.

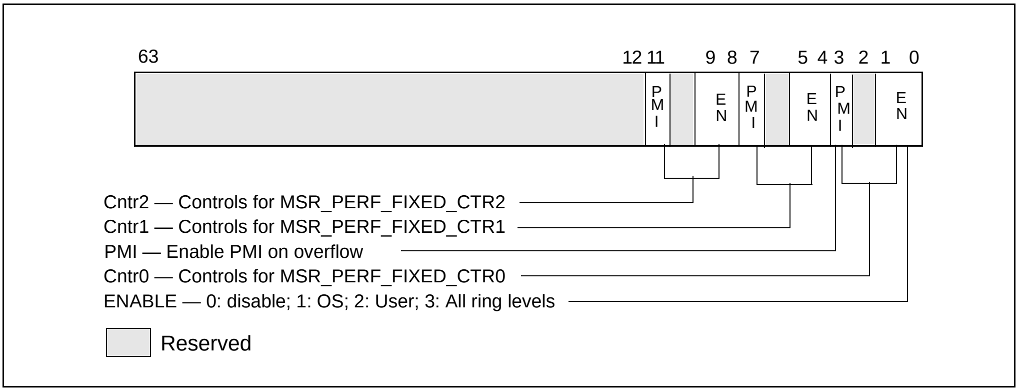
63
12 11
9 8 7
5 4 3
2 1 0
PMI
EN
PMI
EN
PMI
EN
Cntr2 — Controls for MSR_PERF_FIXED_CTR2
Cntr1 — Controls for MSR_PERF_FIXED_CTR1
PMI — Enable PMI on overflow
Cntr0 — Controls for MSR_PERF_FIXED_CTR0
ENABLE — 0: disable; 1: OS; 2: User; 3: All ring levels
Reserved

Figure 18-20. Layout of MSR_PERF_FIXED_CTR_CTRL MSR

- **PMI field (fourth bit in 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.

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 a single WRMSR.
- 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 a single RDMSR.
- 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 a single WRMSR.

MSR_PERF_GLOBAL_CTRL MSR provides single-bit controls to enable counting in each performance counter (see Figure 18-21). 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.

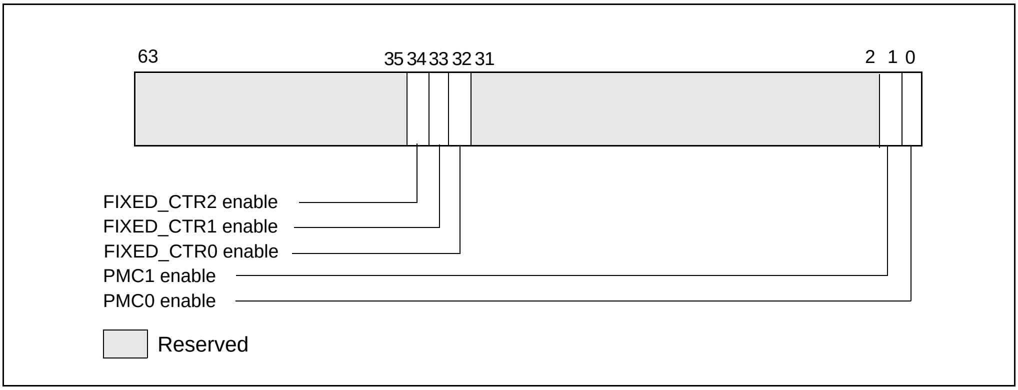

Figure 18-21. Layout of MSR_PERF_GLOBAL_CTRL MSR

MSR_PERF_GLOBAL_STATUS MSR provides single-bit status used by software to query the overflow condition of each performance counter. The MSR also provides additional status bit to indicate overflow conditions when counters are programmed for precise-event-based sampling (PEBS). The MSR_PERF_GLOBAL_STATUS MSR also provides a 'sticky bit' to indicate changes to the state of performance monitoring hardware (see Figure 18-22). A value of 1 in bits 34:32, 1, 0 indicates an overflow condition has occurred in the associated counter.

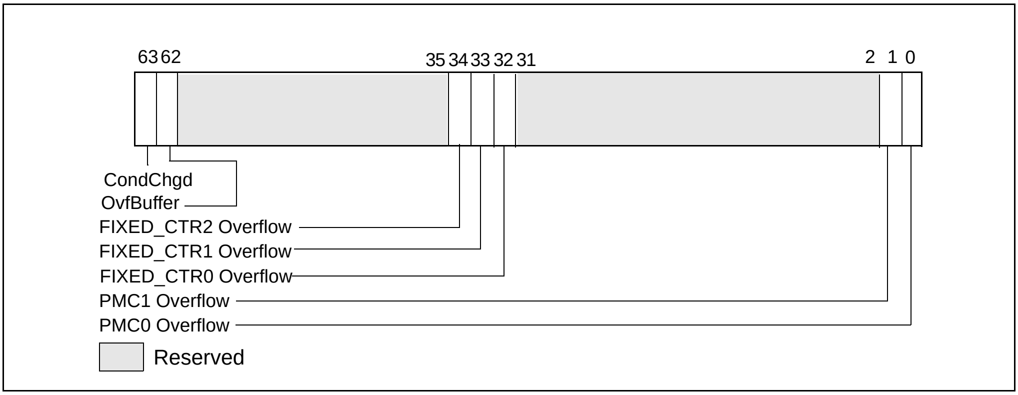

Figure 18-22. Layout of MSR_PERF_GLOBAL_STATUS MSR

When a performance counter is configured for PEBS, an overflow condition in the counter generates a performance-monitoring interrupt this signals a PEBS event. On a PEBS event, the processor stores data records in the buffer area (see Section 18.16.5), clears the counter overflow status, and sets the OvfBuffer bit in MSR_PERF_GLOBAL_STATUS.

MSR_PERF_GLOBAL_OVF_CTL MSR allows software to clear overflow the indicators for general-purpose or fixed-function counters via a single WRMSR (see Figure 18-23). Clear overflow indications when:

- Setting up new values in the event select and/or UMASK field for counting or sampling
- Reloading counter values to continue sampling
- Disabling event counting or sampling

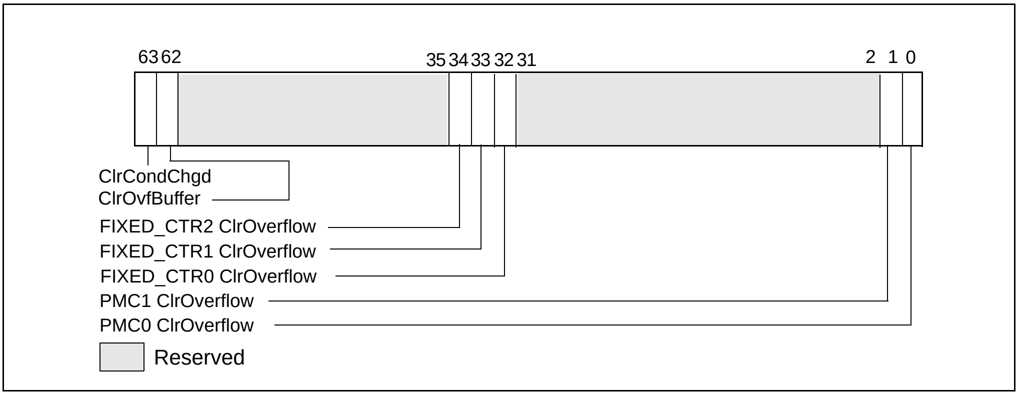

Figure 18-23. Layout of MSR_PERF_GLOBAL_OVF_CTRL MSR

18.14.3 At-Retirement Events

Many non-architectural performance events are impacted by the speculative nature of out-of-order execution. A subset of non-architectural performance events on processors based on Intel Core microarchitecture are enhanced with a tagging mechanism (similar to that found in Intel NetBurst microarchitecture) that exclude contributions that arise from speculative execution. The at-retirement events available in processors based on Intel Core microarchitecture does not require special MSR programming control (see Section 18.16.7, “At-Retirement Counting”), but is limited to IA32_PMC0. See Table 18-15 for a list of events available to processors based on Intel Core microarchitecture.

Table 18-15. At-Retirement Performance Events for Intel Core Microarchitecture

Event Name	UMask	Event Select
ITLB_MISS_RETIRED	00H	C9H
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 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 18.14.4.2).

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 18-16. The procedure for detecting availability of PEBS is the same as described in Section 18.16.8.1.

Table 18-16. 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 18-30 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 18-16.

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 equals 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 governs 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 causing 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 (seeSection 18.16.8) .

18.14.4.3 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the non-precise event-based sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 18.5.3.2, “Debug Store (DS) Mechanism,” for guidelines when writing the DS ISR.

The service routine can query MSR_PERF_GLOBAL_STATUS to determine which counter(s) caused of overflow condition. The service routine should clear overflow indicator by writing to MSR_PERF_GLOBAL_OVF_CTL.

A comparison of the sequence of requirements to program PEBS for processors based on Intel Core and Intel NetBurst microarchitectures is listed in Table 18-17.

Table 18-17. Requirements to Program PEBS

	For Processors based on Intel Core microarchitecture	For Processors based on Intel NetBurst microarchitecture
Verify PEBS support of processor/OS	▪ IA32_MISC_ENABLES.EMON_AVAILABE (bit 7) is set. ▪ IA32_MISC_ENABLES.PEBS_UNAVAILABE (bit 12) is clear.	
Ensure counters are in disabled	On initial set up or changing event configurations, write MSR_PERF_GLOBAL_CTRL MSR (0x38F) with 0. On subsequent entries: ▪ Clear all counters if “Counter Freeze on PMI“ is not enabled. ▪ If IA32_DebugCTL.Freeze is enabled, counters are automatically disabled. Counters MUST be stopped before writing.[1]	Optional
Disable PEBS.	Clear ENABLE PMC0 bit in IA32_PEBS_ENABLE MSR (0x3F1).	Optional
Check overflow conditions.	Check MSR_PERF_GLOBAL_STATUS MSR (0x 38E) handle any overflow conditions.	Check OVF flag of each CCCR for overflow condition
Clear overflow status.	Clear MSR_PERF_GLOBAL_STATUS MSR (0x 38E) using IA32_PERF_GLOBAL_OVF_CTRL MSR (0x390).	Clear OVF flag of each CCCR.
Write “sample-after“ values.	Configure the counter(s) with the sample after value.	
Configure specific counter configuration MSR.	▪ Set local enable bit 22 - 1. ▪ Do NOT set local counter PMI/INT bit, bit 20 - 0. ▪ Event programmed must be PEBS capable.	▪ Set appropriate OVF_PMI bits - 1. ▪ Only CCCR for MSR_IQ_COUNTER4 support PEBS.

Table 18-17. Requirements to Program PEBS (Contd.)

	For Processors based on Intel Core microarchitecture	For Processors based on Intel NetBurst microarchitecture
Allocate buffer for PEBS states.	Allocate a buffer in memory for the precise information.	
Program the IA32_DS_AREA MSR.	Program the IA32_DS_AREA MSR.	
Configure the PEBS buffer management records.	Configure the PEBS buffer management records in the DS buffer management area.	
Configure/Enable PEBS.	Set Enable PMC0 bit in IA32_PEBS_ENABLE MSR (0x3F1).	Configure MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT and MSR_PEBS_MATRIX_HORZ as needed.
Enable counters.	Set Enable bits in MSR_PERF_GLOBAL_CTRL MSR (0x38F).	Set each CCCR enable bit 12 - 1.

NOTES:

1. Counters read while enabled are not guaranteed to be precise with event counts that occur in timing proximity to the RDMSR.

18.15 PERFORMANCE MONITORING (PROCESSORS BASED ON INTEL® ATOM™ MICROARCHITECTURE)

Intel Atom processor family supports architectural performance monitoring capability with version ID 3 (see Section 18.12.2.2) and a host of non-architectural monitoring capabilities. The initial implementation of Intel Atom processor family provides two general-purpose performance counters (IA32_PMC0, IA32_PMC1) and three fixed-function performance counters (IA32_FIXED_CTR0, IA32_FIXED_CTR1, IA32_FIXED_CTR2).

Non-architectural performance monitoring in Intel Atom processor family uses the IA32_PERFEVTSELx MSR to configure a set of non-architecture performance monitoring events to be counted by the corresponding general-purpose performance counter. The list of non-architectural performance monitoring events is listed in Table A-5.

Architectural and non-architectural performance monitoring events in Intel Atom processor family support thread qualification using bit 21 of IA32_PERFEVTSELx MSR.

The bit fields within each IA32_PERFEVTSELx MSR are defined in Figure 18-17 and described in Section 18.12.1.1 and Section 18.12.2.2.

Valid event mask (Umask) bits are listed in Appendix A. The UMASK field may contain sub-fields that provide the same qualifying actions like those listed in Table 18-8, Table 18-9, Table 18-10, and Table 18-11. One or more of these sub-fields may apply to specific events on an event-by-event basis. Details are listed in Table A-5 in Appendix A, “Performance-Monitoring Events.”

18.16 PERFORMANCE MONITORING (PROCESSORS BASED ON INTEL NETBURST MICROARCHITECTURE)

The performance monitoring mechanism provided in Pentium 4 and Intel Xeon processors is different from that provided in the P6 family and Pentium processors. While the general concept of selecting, filtering, counting, and reading performance events through the WRMSR, RDMSR, and RDPMC instructions is unchanged, the setup mechanism and MSR layouts are incompatible with the P6 family and Pentium processor mechanisms. Also, the RDPMC instruction has been enhanced to read the the additional performance counters provided in the Pentium 4 and Intel Xeon processors and to allow faster reading of counters.

The event monitoring mechanism provided with the Pentium 4 and Intel Xeon processors (based on Intel NetBurst microarchitecture) consists of the following facilities:

- The IA32_MISC_ENABLE MSR, which indicates the availability in an Intel 64 or IA-32 processor of the performance monitoring and precise event-based sampling (PEBS) facilities.
- Event selection control (ESCR) MSRs for selecting events to be monitored with specific performance counters. The number available differs by family and model (43 to 45).
- 18 performance counter MSRs for counting events.
- 18 counter configuration control (CCCR) MSRs, with one CCCR associated with each performance counter. CCCRs sets up an associated performance counter for a specific method of counting.
- A debug store (DS) save area in memory for storing PEBS records.
- The IA32_DS_AREA MSR, which establishes the location of the DS save area.
- The debug store (DS) feature flag (bit 21) returned by the CPUID instruction, which indicates the availability of the DS mechanism.
- The MSR_PEBS_ENABLE MSR, which enables the PEBS facilities and replay tagging used in at-retirement event counting.
- A set of predefined events and event metrics that simplify the setting up of the performance counters to count specific events.

Table 18-18 lists the performance counters and their associated CCCRs, along with the ESCRs that select events to be counted for each performance counter. Predefined event metrics and events are listed in Appendix A, “Performance-Monitoring Events.”

Table 18-18. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Pentium 4 and Intel Xeon Processors)

Counter			CCCR		ESCR		
Name	**No.**	**Addr**	**Name**	**Addr**	**Name**	**No.**	**Addr**
MSR_BPU_COUNTER0	0	300H	MSR_BPU_CCCR0	360H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER1	1	301H	MSR_BPU_CCCR1	361H	MSR_BSU_ESCR0 MSR_FSB_ESCR0 MSR_MOB_ESCR0 MSR_PMH_ESCR0 MSR_BPU_ESCR0 MSR_IS_ESCR0 MSR_ITLB_ESCR0 MSR_IX_ESCR0	7 6 2 4 0 1 3 5	3A0H 3A2H 3AAH 3ACH 3B2H 3B4H 3B6H 3C8H
MSR_BPU_COUNTER2	2	302H	MSR_BPU_CCCR2	362H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_BPU_COUNTER3	3	303H	MSR_BPU_CCCR3	363H	MSR_BSU_ESCR1 MSR_FSB_ESCR1 MSR_MOB_ESCR1 MSR_PMH_ESCR1 MSR_BPU_ESCR1 MSR_IS_ESCR1 MSR_ITLB_ESCR1 MSR_IX_ESCR1	7 6 2 4 0 1 3 5	3A1H 3A3H 3ABH 3ADH 3B3H 3B5H 3B7H 3C9H
MSR_MS_COUNTER0	4	304H	MSR_MS_CCCR0	364H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER1	5	305H	MSR_MS_CCCR1	365H	MSR_MS_ESCR0 MSR_TBPU_ESCR0 MSR_TC_ESCR0	0 2 1	3C0H 3C2H 3C4H
MSR_MS_COUNTER2	6	306H	MSR_MS_CCCR2	366H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H
MSR_MS_COUNTER3	7	307H	MSR_MS_CCCR3	367H	MSR_MS_ESCR1 MSR_TBPU_ESCR1 MSR_TC_ESCR1	0 2 1	3C1H 3C3H 3C5H

Table 18-18. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Pentium 4 and Intel Xeon Processors) (Contd.)

Counter			CCCR		ESCR		
Name	**No.**	**Addr**	**Name**	**Addr**	**Name**	**No.**	**Addr**
MSR_FLAME_COUNTER0	8	308H	MSR_FLAME_CCCR0	368H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAAT_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_COUNTER1	9	309H	MSR_FLAME_CCCR1	369H	MSR_FIRM_ESCR0 MSR_FLAME_ESCR0 MSR_DAC_ESCR0 MSR_SAAT_ESCR0 MSR_U2L_ESCR0	1 0 5 2 3	3A4H 3A6H 3A8H 3AEH 3B0H
MSR_FLAME_COUNTER2	10	30AH	MSR_FLAME_CCCR2	36AH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_FLAME_COUNTER3	11	30BH	MSR_FLAME_CCCR3	36BH	MSR_FIRM_ESCR1 MSR_FLAME_ESCR1 MSR_DAC_ESCR1 MSR_SAAT_ESCR1 MSR_U2L_ESCR1	1 0 5 2 3	3A5H 3A7H 3A9H 3AFH 3B1H
MSR_IQ_COUNTER0	12	30CH	MSR_IQ_CCCR0	36CH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0[1] MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER1	13	30DH	MSR_IQ_CCCR1	36DH	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0[1] MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER2	14	30EH	MSR_IQ_CCCR2	36EH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1[1] MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH
MSR_IQ_COUNTER3	15	30FH	MSR_IQ_CCCR3	36FH	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1[1] MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH

Table 18-18. Performance Counter MSRs and Associated CCCR and ESCR MSRs (Pentium 4 and Intel Xeon Processors) (Contd.)

Counter			CCCR		ESCR		
Name	**No.**	**Addr**	**Name**	**Addr**	**Name**	**No.**	**Addr**
MSR_IQ_COUNTER4	16	310H	MSR_IQ_CCCR4	370H	MSR_CRU_ESCR0 MSR_CRU_ESCR2 MSR_CRU_ESCR4 MSR_IQ_ESCR0[1] MSR_RAT_ESCR0 MSR_SSU_ESCR0 MSR_ALF_ESCR0	4 5 6 0 2 3 1	3B8H 3CCH 3E0H 3BAH 3BCH 3BEH 3CAH
MSR_IQ_COUNTER5	17	311H	MSR_IQ_CCCR5	371H	MSR_CRU_ESCR1 MSR_CRU_ESCR3 MSR_CRU_ESCR5 MSR_IQ_ESCR1[1] MSR_RAT_ESCR1 MSR_ALF_ESCR1	4 5 6 0 2 1	3B9H 3CDH 3E1H 3BBH 3BDH 3CBH

NOTES:

[1] MSR_IQ_ESCR0 and MSR_IQ_ESCR1 are available only on early processor builds (family 0FH, models 01H-02H). These MSRs are not available on later versions.

The types of events that can be counted with these performance monitoring facilities are divided into two classes: non-retirement events and at-retirement events.

- Non-retirement events (see Table A-7) are events that occur any time during instruction execution (such as bus transactions or cache transactions).
- At-retirement events (see Table A-8) are events that are counted at the retirement stage of instruction execution, which allows finer granularity in counting events and capturing machine state.

  The at-retirement counting mechanism includes facilities for tagging μops that have encountered a particular performance event during instruction execution. Tagging allows events to be sorted between those that occurred on an execution path that resulted in architectural state being committed at retirement as well as events that occurred on an execution path where the results were eventually cancelled and never committed to architectural state (such as, the execution of a mispredicted branch).

The Pentium 4 and Intel Xeon processor performance monitoring facilities support the three usage models described below. The first two models can be used to count both non-retirement and at-retirement events; the third model is used to count a subset of at-retirement events:

- **Event counting** — A performance counter is configured to count one or more types of events. While the counter is counting, software reads the counter at selected intervals to determine the number of events that have been counted between the intervals.
- **Non-precise event-based sampling** — A performance counter is configured to count one or more types of events and to generate an interrupt when it

overflows. To trigger an overflow, the counter is preset to a modulus value that will cause the counter to overflow after a specific number of events have been counted.

When the counter overflows, the processor generates a performance monitoring interrupt (PMI). The interrupt service routine for the PMI then records the return instruction pointer (RIP), resets the modulus, and restarts the counter. Code performance can be analyzed by examining the distribution of RIPs with a tool like the VTune™ Performance Analyzer.

- **Precise event-based sampling (PEBS) —** This type of performance monitoring is similar to non-precise event-based sampling, except that a memory buffer is used to save a record of the architectural state of the processor whenever the counter overflows. The records of architectural state provide additional information for use in performance tuning. Precise event-based sampling can be used to count only a subset of at-retirement events.

The following sections describe the MSRs and data structures used for performance monitoring in the Pentium 4 and Intel Xeon processors.

18.16.1 ESCR MSRs

The 45 ESCR MSRs (see Table 18-18) allow software to select specific events to be countered. Each ESCR is usually associated with a pair of performance counters (see Table 18-18) and each performance counter has several ESCRs associated with it (allowing the events counted to be selected from a variety of events).

Figure 18-24 shows the layout of an ESCR MSR. The functions of the flags and fields are:

- **USR flag, bit 2 —** When set, events are counted when the processor is operating at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.
- **OS flag, bit 3 —** When set, events are counted when the processor is operating at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the OS and USR flags are set, events are counted at all privilege levels.)

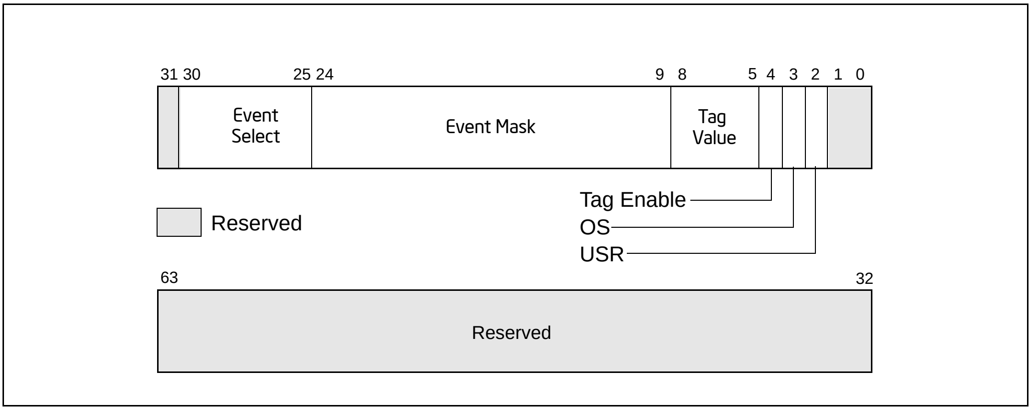

Figure 18-24. Event Selection Control Register (ESCR) for Pentium 4 and Intel Xeon Processors without HT Technology Support

- **Tag enable, bit 4 —** When set, enables tagging of µops to assist in at-retirement event counting; when clear, disables tagging. See Section 18.16.7, "At-Retirement Counting."
- **Tag value field, bits 5 through 8 —** Selects a tag value to associate with a µop to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24 —** Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30) —** Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

When setting up an ESCR, the event select field is used to select a specific class of events to count, such as retired branches. The event mask field is then used to select one or more of the specific events within the class to be counted. For example, when counting retired branches, four different events can be counted: branch not taken predicted, branch not taken mispredicted, branch taken predicted, and branch taken mispredicted. The OS and USR flags allow counts to be enabled for events that occur when operating system code and/or application code are being executed. If neither the OS nor USR flag is set, no events will be counted.

The ESCRs are initialized to all 0s on reset. The flags and fields of an ESCR are configured by writing to the ESCR using the WRMSR instruction. Table 18-18 gives the addresses of the ESCR MSRs.

Writing to an ESCR MSR does not enable counting with its associated performance counter; it only selects the event or events to be counted. The CCCR for the selected performance counter must also be configured. Configuration of the CCCR includes selecting the ESCR and enabling the counter.

18.16.2 Performance Counters

The performance counters in conjunction with the counter configuration control registers (CCCRs) are used for filtering and counting the events selected by the ESCRs. The Pentium 4 and Intel Xeon processors provide 18 performance counters organized into 9 pairs. A pair of performance counters is associated with a particular subset of events and ESCR's (see Table 18-18). The counter pairs are partitioned into four groups:

- The BPU group, includes two performance counter pairs:
  - MSR_BPU_COUNTER0 and MSR_BPU_COUNTER1.
  - MSR_BPU_COUNTER2 and MSR_BPU_COUNTER3.
- The MS group, includes two performance counter pairs:
  - MSR_MS_COUNTER0 and MSR_MS_COUNTER1.
  - MSR_MS_COUNTER2 and MSR_MS_COUNTER3.
- The FLAME group, includes two performance counter pairs:
  - MSR_FLAME_COUNTER0 and MSR_FLAME_COUNTER1.
  - MSR_FLAME_COUNTER2 and MSR_FLAME_COUNTER3.
- The IQ group, includes three performance counter pairs:
  - MSR_IQ_COUNTER0 and MSR_IQ_COUNTER1.
  - MSR_IQ_COUNTER2 and MSR_IQ_COUNTER3.
  - MSR_IQ_COUNTER4 and MSR_IQ_COUNTER5.

The MSR_IQ_COUNTER4 counter in the IQ group provides support for the PEBS.

Alternate counters in each group can be cascaded: the first counter in one pair can start the first counter in the second pair and vice versa. A similar cascading is possible for the second counters in each pair. For example, within the BPU group of counters, MSR_BPU_COUNTER0 can start MSR_BPU_COUNTER2 and vice versa, and MSR_BPU_COUNTER1 can start MSR_BPU_COUNTER3 and vice versa (see Section 18.16.6.6, "Cascading Counters"). The cascade flag in the CCCR register for the performance counter enables the cascading of counters.

Each performance counter is 40-bits wide (see Figure 18-25). The RDPMC instruction has been enhanced in the Pentium 4 and Intel Xeon processors to allow reading of either the full counter-width (40-bits) or the low 32-bits of the counter. Reading the low 32-bits is faster than reading the full counter width and is appropriate in situations where the count is small enough to be contained in 32 bits.

The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

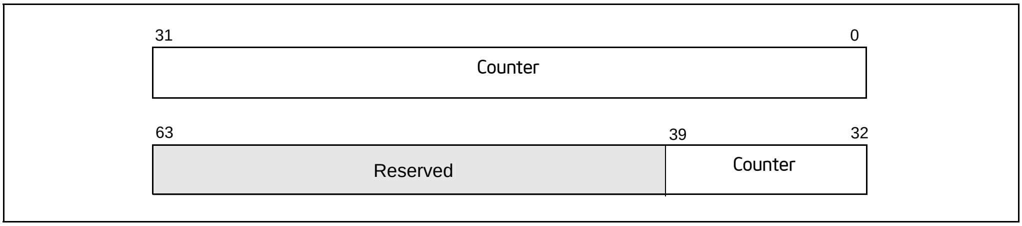

Figure 18-25. Performance Counter (Pentium 4 and Intel Xeon Processors)

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

Some uses of the performance counters require the counters to be preset before counting begins (that is, before the counter is enabled). This can be accomplished by writing to the counter using the WRMSR instruction. To set a counter to a specified number of counts before overflow, enter a 2s complement negative integer in the counter. The counter will then count from the preset value up to -1 and overflow. Writing to a performance counter in a Pentium 4 or Intel Xeon processor with the WRMSR instruction causes all 40 bits of the counter to be written.

18.16.3 CCCR MSRs

Each of the 18 performance counters in a Pentium 4 or Intel Xeon processor has one CCCR MSR associated with it (see Table 18-18). The CCCRs control the filtering and counting of events as well as interrupt generation. Figure 18-26 shows the layout of an CCCR MSR. The functions of the flags and fields are as follows:

- **Enable flag, bit 12 —** When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset.
- **ESCR select field, bits 13 through 15 —** Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.
- **Compare flag, bit 18 —** When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.
- **Complement flag, bit 19 —** Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance

counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 18.16.6.2, “Filtering Events”). The complement flag is not active unless the compare flag is set.

- **Threshold field, bits 20 through 23 —** Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 18.16.6.2, “Filtering Events”).
- **Edge flag, bit 24 —** When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.

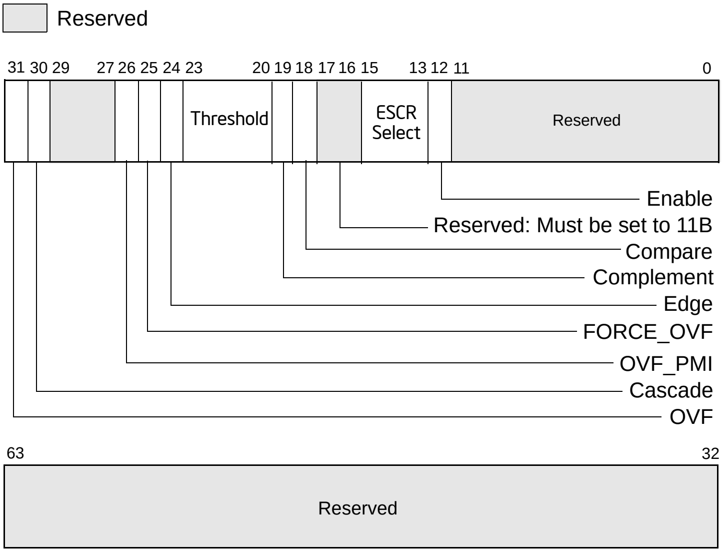

Figure 18-26. Counter Configuration Control Register (CCCR)

- **FORCE_OVF flag, bit 25 —** When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI flag, bit 26 —** When set, causes a performance monitor interrupt (PMI) to be generated when the counter overflows occurs; when clear, disables PMI generation. Note that the PMI is generated on the next event count after the counter has overflowed.

- **Cascade flag, bit 30 —** When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 18.16.2, “Performance Counters,” for further details); when clear, disables cascading of counters.
- **OVF flag, bit 31 —** Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

The CCCRs are initialized to all 0s on reset.

The events that an enabled performance counter actually counts are selected and filtered by the following flags and fields in the ESCR and CCCR registers and in the qualification order given:

1. The event select and event mask fields in the ESCR select a class of events to be counted and one or more event types within the class, respectively.
2. The OS and USR flags in the ESCR selected the privilege levels at which events will be counted.
3. The ESCR select field of the CCCR selects the ESCR. Since each counter has several ESCRs associated with it, one ESCR must be chosen to select the classes of events that may be counted.
4. The compare and complement flags and the threshold field of the CCCR select an optional threshold to be used in qualifying an event count.
5. The edge flag in the CCCR allows events to be counted only on rising-edge transitions.

The qualification order in the above list implies that the filtered output of one “stage” forms the input for the next. For instance, events filtered using the privilege level flags can be further qualified by the compare and complement flags and the threshold field, and an event that matched the threshold criteria, can be further qualified by edge detection.

The uses of the flags and fields in the CCCRs are discussed in greater detail in Section 18.16.6, “Programming the Performance Counters for Non-Retirement Events.”

18.16.4 Debug Store (DS) Mechanism

The debug store (DS) mechanism was introduced in the Pentium 4 and Intel Xeon processors to allow various types of information to be collected in memory-resident buffers for use in debugging and tuning programs. For the Pentium 4 and Intel Xeon processors, the DS mechanism is used to collect two types of information: branch records and precise event-based sampling (PEBS) records. The availability of the DS mechanism in a processor is indicated with the DS feature flag (bit 21) returned by the CPUID instruction.

See Section 18.6.8, “Branch Trace Store (BTS),” and Section 18.16.8, “Precise Event-Based Sampling (PEBS),” for a description of these facilities. Records collected with the DS mechanism are saved in the DS save area. See Section 18.16.5, “DS Save Area.”

18.16.5 DS Save Area

The debug store (DS) save area is a software-designated area of memory that is used to collect the following two types of information:

- **Branch records —** When the BTS flag in the MSR_DEBUGCTLA MSR is set, a branch record is stored in the BTS buffer in the DS save area whenever a taken branch, interrupt, or exception is detected.
- **PEBS records —** When a performance counter is configured for PEBS, a PEBS record is stored in the PEBS buffer in the DS save area after the counter overflow occurs. This record contains the architectural state of the processor (state of the 8 general purpose registers, EIP register, and EFLAGS register) at the next occurrence of the PEBS event that caused the counter to overflow. When the state information has been logged, the counter is automatically reset to a preselected value, and event counting begins again. This feature is available only for a subset of the Pentium 4 and Intel Xeon processors' performance events.

NOTES

DS save area and recording mechanism is not available in the SMM. The feature is disabled on transition to the SMM mode. Similarly DS recording is disabled on the generation of a machine check exception and is cleared on processor RESET and INIT. DS recording is available in real address mode.

The BTS and PEBS facilities may not be available on all processors. The availability of these facilities is indicated by the BTS_UNAVAILABLE and PEBS_UNAVAILABLE flags, respectively, in the IA32_MISC_ENABLE MSR (see Appendix B).

The DS save area is divided into three parts (see Figure 18-27): buffer management area, branch trace store (BTS) buffer, and PEBS buffer. The buffer management area is used to define the location and size of the BTS and PEBS buffers. The processor then uses the buffer management area to keep track of the branch and/or PEBS records in their respective buffers and to record the performance counter reset value. The linear address of the first byte of the DS buffer management area is specified with the IA32_DS_AREA MSR.

The fields in the buffer management area are as follows:

- **BTS buffer base —** Linear address of the first byte of the BTS buffer. This address should point to a natural doubleword boundary.
- **BTS index —** Linear address of the first byte of the next BTS record to be written to. Initially, this address should be the same as the address in the BTS buffer base field.
- **BTS absolute maximum —** Linear address of the next byte past the end of the BTS buffer. This address should be a multiple of the BTS record size (12 bytes) plus 1.

- **BTS interrupt threshold —** Linear address of the BTS record on which an interrupt is to be generated. This address must point to an offset from the BTS buffer base that is a multiple of the BTS record size. Also, it must be several records short of the BTS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the BTS absolute maximum record.
- **PEBS buffer base —** Linear address of the first byte of the PEBS buffer. This address should point to a natural doubleword boundary.
- **PEBS index —** Linear address of the first byte of the next PEBS record to be written to. Initially, this address should be the same as the address in the PEBS buffer base field.

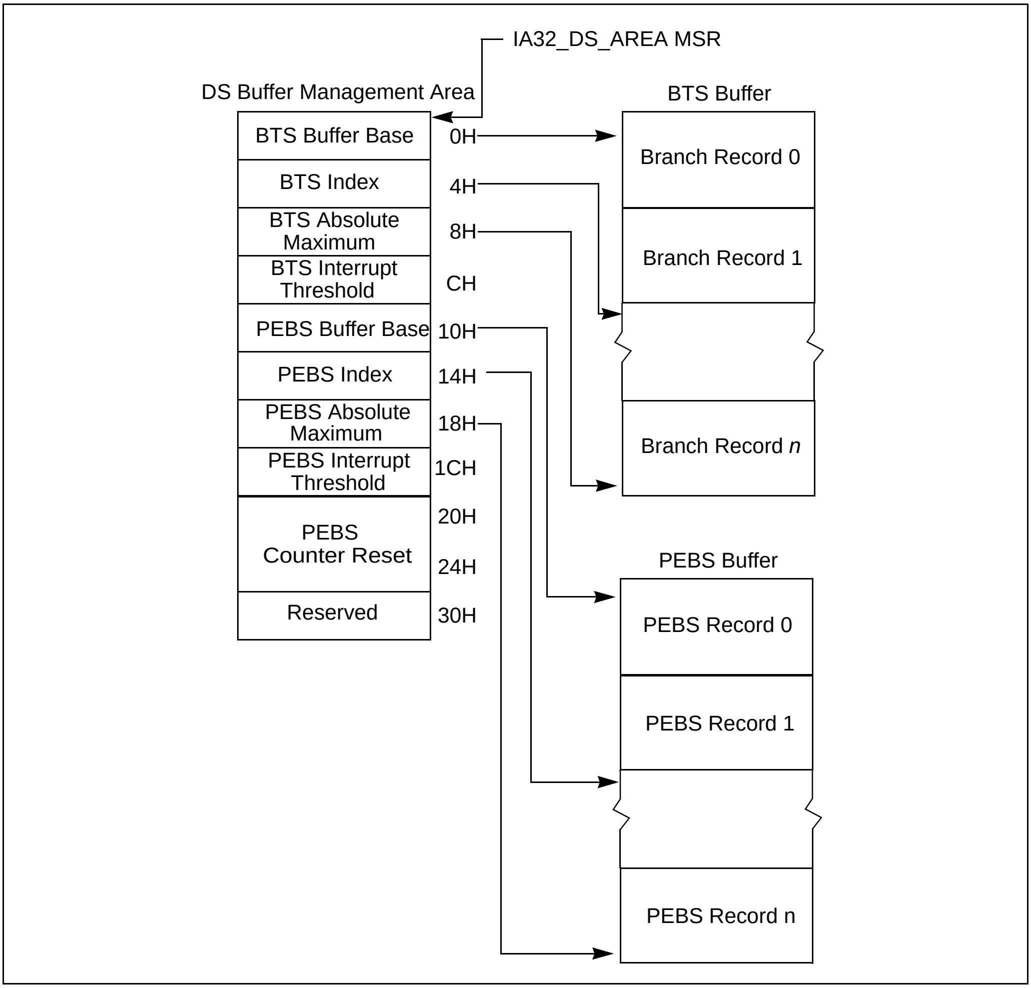

Figure 18-27. DS Save Area

- **PEBS absolute maximum —** Linear address of the next byte past the end of the PEBS buffer. This address should be a multiple of the PEBS record size (40 bytes) plus 1.
- **PEBS interrupt threshold —** Linear address of the PEBS record on which an interrupt is to be generated. This address must point to an offset from the PEBS buffer base that is a multiple of the PEBS record size. Also, it must be several records short of the PEBS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the PEBS absolute maximum record.
- **PEBS counter reset value —** A 40-bit value that the counter is to be reset to after state information has collected following counter overflow. This value allows state information to be collected after a preset number of events have been counted.

Figures 18-28 shows the structure of a 12-byte branch record in the BTS buffer. The fields in each record are as follows:

- **Last branch from —** Linear address of the instruction from which the branch, interrupt, or exception was taken.
- **Last branch to —** Linear address of the branch target or the first instruction in the interrupt or exception service routine.
- **Branch predicted —** Bit 4 of field indicates whether the branch that was taken was predicted (set) or not predicted (clear).

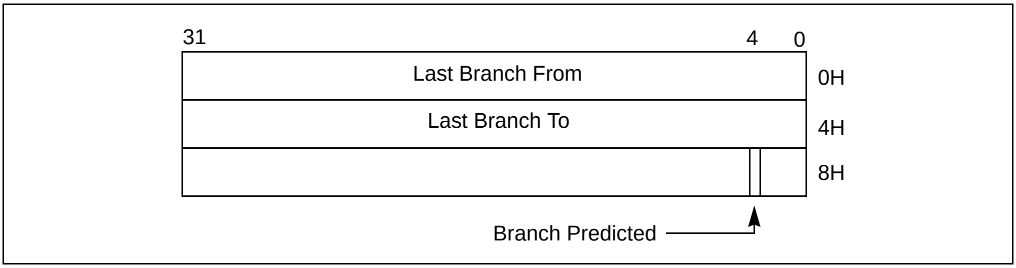

Figure 18-28. 32-bit Branch Trace Record Format

Figures 18-29 shows the structure of the 40-byte PEBS records. Nominally the register values are those at the beginning of the instruction that caused the event. However, there are cases where the registers may be logged in a partially modified state. The linear IP field shows the value in the EIP register translated from an offset into the current code segment to a linear address.

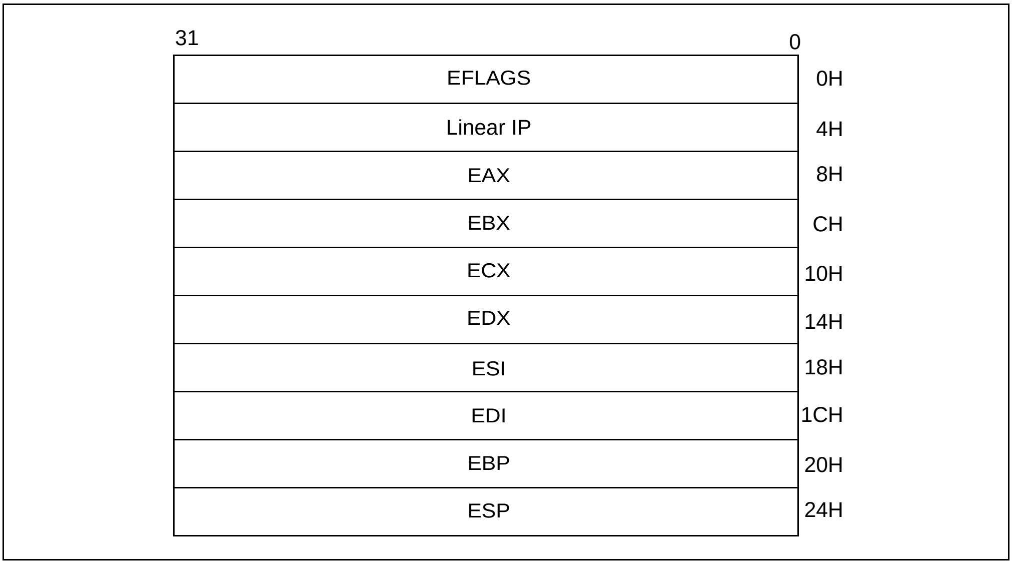

Figure 18-29. PEBS Record Format

18.16.5.1 DS Save Area and IA-32e Mode Operation

When IA-32e mode is active (IA32_EFER.LMA = 1), the structure of the DS save area is shown in Figure 18-30. The organization of each field in IA-32e mode operation is similar to that of non-IA-32e mode operation. However, each field now stores a 64-bit address. The IA32_DS_AREA MSR holds the 64-bit linear address of the first byte of the DS buffer management area.

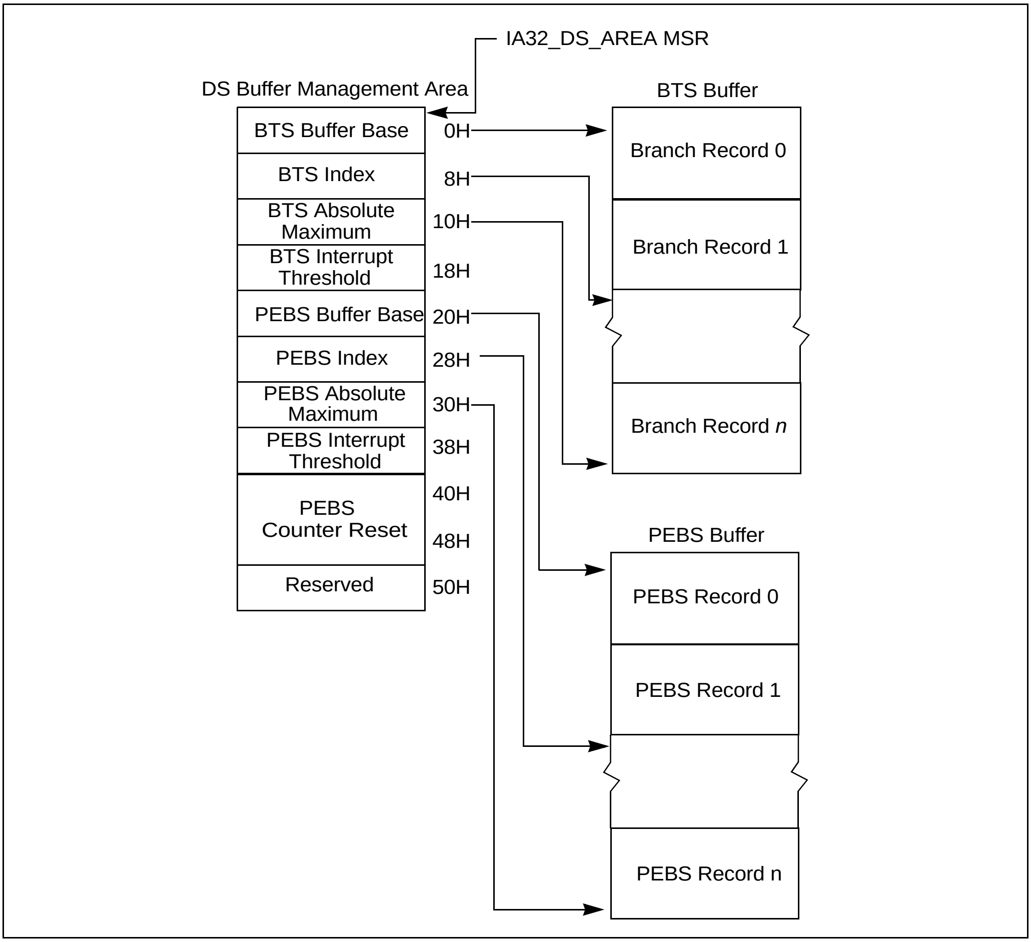

Figure 18-30. IA-32e Mode DS Save Area

When IA-32e mode is active, the structure of a branch trace record is similar to that shown in Figure 18-28, but each field is 8 bytes in length. This makes each BTS record 24 bytes (see Figure 18-31). The structure of a PEBS record is similar to that shown in Figure 18-29, but each field is 8 bytes in length and architectural states include register R8 through R15. This makes the size of a PEBS record in 64-bit mode 144 bytes (see Figure 18-32).

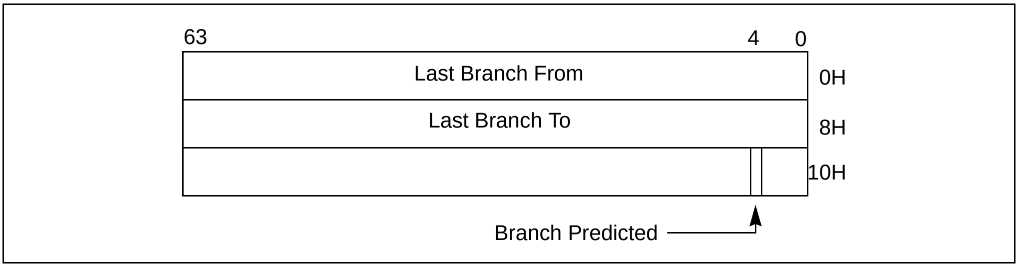

Figure 18-31. 64-bit Branch Trace Record Format

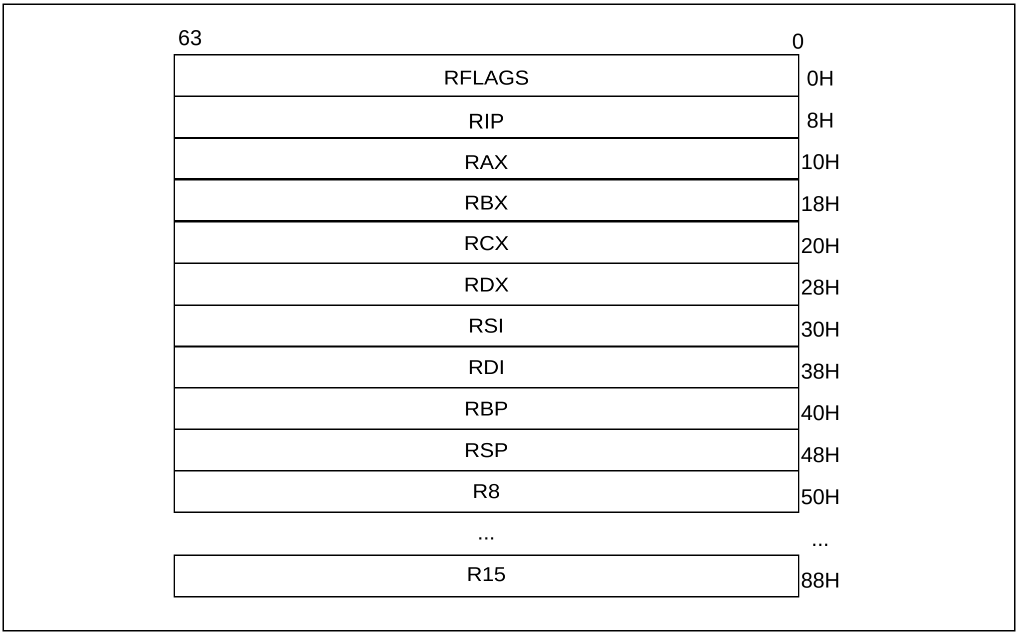

Figure 18-32. 64-bit PEBS Record Format

18.16.6 Programming the Performance Counters for Non-Retirement Events

The basic steps to program a performance counter and to count events include the following:

1. Select the event or events to be counted.
2. For each event, select an ESCR that supports the event using the values in the ESCR restrictions row in Table A-7, Appendix A.

3. Match the CCCR Select value and ESCR name in Table A-7 to a value listed in Table 18-18; select a CCCR and performance counter.
4. Set up an ESCR for the specific event or events to be counted and the privilege levels at which the are to be counted.
5. Set up the CCCR for the performance counter by selecting the ESCR and the desired event filters.
6. Set up the CCCR for optional cascading of event counts, so that when the selected counter overflows its alternate counter starts.
7. Set up the CCCR to generate an optional performance monitor interrupt (PMI) when the counter overflows. If PMI generation is enabled, the local APIC must be set up to deliver the interrupt to the processor and a handler for the interrupt must be in place.
8. Enable the counter to begin counting.

18.16.6.1 Selecting Events to Count

Table A-8 in Appendix A lists a set of at-retirement events for the Pentium 4 and Intel Xeon processors. For each event listed in Table A-8, setup information is provided. Table 18-19 gives an example of one of the events.

Table 18-19. Event Example

Event Name	Event Parameters	Parameter Value	Description
branch_retired			Counts the retirement of a branch. Specify one or more mask bits to select any combination of branch taken, not-taken, predicted and mispredicted.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	See Table 15-3 for the addresses of the ESCR MSRs
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	The counter numbers associated with each ESCR are provided. The performance counters and corresponding CCCRs can be obtained from Table 15-3.
	ESCR Event Select	06H	ESCR[31:25]
	ESCR Event Mask	 Bit 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9], Branch Not-taken Predicted, Branch Not-taken Mispredicted, Branch Taken Predicted, Branch Taken Mispredicted.
	CCCR Select	05H	CCCR[15:13]

Table 18-19. Event Example (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		P6: EMON_BR_INST_RETIRED
	Can Support PEBS	No	
	Requires Additional MSRs for Tagging	No	

For Table A-7 and Table A-8, Appendix A, the name of the event is listed in the Event Name column and parameters that define the event and other information are listed in the Event Parameters column. The Parameter Value and Description columns give specific parameters for the event and additional description information. Entries in the Event Parameters column are described below.

- **ESCR restrictions —** Lists the ESCRs that can be used to program the event. Typically only one ESCR is needed to count an event.
- **Counter numbers per ESCR —** Lists which performance counters are associated with each ESCR. Table 18-18 gives the name of the counter and CCCR for each counter number. Typically only one counter is needed to count the event.
- **ESCR event select —** Gives the value to be placed in the event select field of the ESCR to select the event.
- **ESCR event mask —** Gives the value to be placed in the Event Mask field of the ESCR to select sub-events to be counted. The parameter value column defines the documented bits with relative bit position offset starting from 0, where the absolute bit position of relative offset 0 is bit 9 of the ESCR. All undocumented bits are reserved and should be set to 0.
- **CCCR select —** Gives the value to be placed in the ESCR select field of the CCCR associated with the counter to select the ESCR to be used to define the event. This value is not the address of the ESCR; it is the number of the ESCR from the Number column in Table 18-18.
- **Event specific notes —** Gives additional information about the event, such as the name of the same or a similar event defined for the P6 family processors.
- **Can support PEBS —** Indicates if PEBS is supported for the event (only supplied for at-retirement events listed in Table A-8.)
- **Requires additional MSR for tagging —** Indicates which if any additional MSRs must be programmed to count the events (only supplied for the at-retirement events listed in Table A-8.)

NOTE

The performance-monitoring events listed in Appendix A, “Performance-Monitoring Events,” are intended to be used as guides for performance tuning. The counter values reported are not guaranteed to be absolutely accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The following procedure shows how to set up a performance counter for basic counting; that is, the counter is set up to count a specified event indefinitely, wrapping around whenever it reaches its maximum count. This procedure is continued through the following four sections.

Using information in Table A-7, Appendix A, an event to be counted can be selected as follows:

1. Select the event to be counted.
2. Select the ESCR to be used to select events to be counted from the ESCRs field.
3. Select the number of the counter to be used to count the event from the Counter Numbers Per ESCR field.
4. Determine the name of the counter and the CCCR associated with the counter, and determine the MSR addresses of the counter, CCCR, and ESCR from Table 18-18.
5. Use the WRMSR instruction to write the ESCR Event Select and ESCR Event Mask values into the appropriate fields in the ESCR. At the same time set or clear the USR and OS flags in the ESCR as desired.
6. Use the WRMSR instruction to write the CCCR Select value into the appropriate field in the CCCR.

NOTE

Typically all the fields and flags of the CCCR will be written with one WRMSR instruction; however, in this procedure, several WRMSR writes are used to more clearly demonstrate the uses of the various CCCR fields and flags.

This setup procedure is continued in the next section, Section 18.16.6.2, “Filtering Events.”

18.16.6.2 Filtering Events

Each counter receives up to 4 input lines from the processor hardware from which it is counting events. The counter treats these inputs as binary inputs (input 0 has a value of 1, input 1 has a value of 2, input 3 has a value of 4, and input 3 has a value of 8). When a counter is enabled, it adds this binary input value to the counter value on each clock cycle. For each clock cycle, the value added to the counter can then range from 0 (no event) to 15.

For many events, only the 0 input line is active, so the counter is merely counting the clock cycles during which the 0 input is asserted. However, for some events two or more input lines are used. Here, the counters threshold setting can be used to filter events. The compare, complement, threshold, and edge fields control the filtering of counter increments by input value.

If the compare flag is set, then a "greater than" or a "less than or equal to" comparison of the input value vs. a threshold value can be made. The complement flag selects "less than or equal to" (flag set) or "greater than" (flag clear). The threshold field selects a threshold value of from 0 to 15. For example, if the complement flag is cleared and the threshold field is set to 6, than any input value of 7 or greater on the 4 inputs to the counter will cause the counter to be incremented by 1, and any value less than 7 will cause an increment of 0 (or no increment) of the counter. Conversely, if the complement flag is set, any value from 0 to 6 will increment the counter and any value from 7 to 15 will not increment the counter. Note that when a threshold condition has been satisfied, the input to the counter is always 1, not the input value that is presented to the threshold filter.

The edge flag provides further filtering of the counter inputs when a threshold comparison is being made. The edge flag is only active when the compare flag is set. When the edge flag is set, the resulting output from the threshold filter (a value of 0 or 1) is used as an input to the edge filter. Each clock cycle, the edge filter examines the last and current input values and sends a count to the counter only when it detects a "rising edge" event; that is, a false-to-true transition. Figure 18-33 illustrates rising edge filtering.

The following procedure shows how to configure a CCCR to filter events using the threshold filter and the edge filter. This procedure is a continuation of the setup procedure introduced in Section 18.16.6.1, "Selecting Events to Count."

7. (Optional) To set up the counter for threshold filtering, use the WRMSR instruction to write values in the CCCR compare and complement flags and the threshold field:
   - Set the compare flag.
   - Set or clear the complement flag for less than or equal to or greater than comparisons, respectively.
   - Enter a value from 0 to 15 in the threshold field.
8. (Optional) Select rising edge filtering by setting the CCCR edge flag.

This setup procedure is continued in the next section, Section 18.16.6.3, "Starting Event Counting."

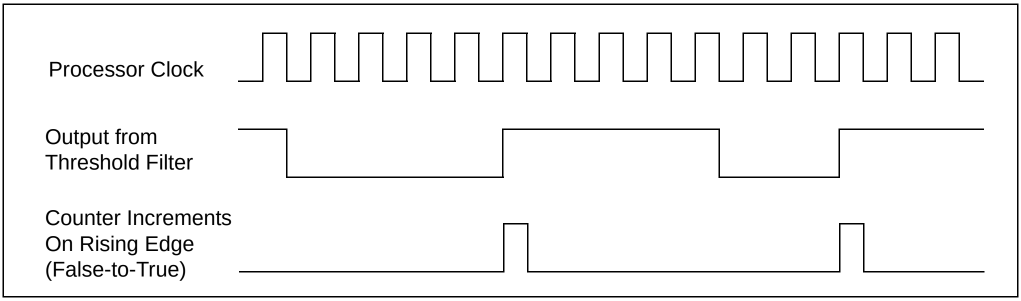
Processor Clock
Output from Threshold Filter
Counter Increments On Rising Edge (False-to-True)

Figure 18-33. Effects of Edge Filtering

18.16.6.3 Starting Event Counting

Event counting by a performance counter can be initiated in either of two ways. The typical way is to set the enable flag in the counter's CCCR. Following the instruction to set the enable flag, event counting begins and continues until it is stopped (see Section 18.16.6.5, "Halting Event Counting").

The following procedural step shows how to start event counting. This step is a continuation of the setup procedure introduced in Section 18.16.6.2, "Filtering Events."

9. To start event counting, use the WRMSR instruction to set the CCCR enable flag for the performance counter.

This setup procedure is continued in the next section, Section 18.16.6.4, "Reading a Performance Counter's Count."

The second way that a counter can be started by using the cascade feature. Here, the overflow of one counter automatically starts its alternate counter (see Section 18.16.6.6, "Cascading Counters").

18.16.6.4 Reading a Performance Counter's Count

The Pentium 4 and Intel Xeon processors' performance counters can be read using either the RDPMC or RDMSR instructions. The enhanced functions of the RDPMC instruction (including fast read) are described in Section 18.16.2, "Performance Counters." These instructions can be used to read a performance counter while it is counting or when it is stopped.

The following procedural step shows how to read the event counter. This step is a continuation of the setup procedure introduced in Section 18.16.6.3, "Starting Event Counting."

10. To read a performance counters current event count, execute the RDPMC instruction with the counter number obtained from Table 18-18 used as an operand.

This setup procedure is continued in the next section, Section 18.16.6.5, “Halting Event Counting.”

18.16.6.5 Halting Event Counting

After a performance counter has been started (enabled), it continues counting indefinitely. If the counter overflows (goes one count past its maximum count), it wraps around and continues counting. When the counter wraps around, it sets its OVF flag to indicate that the counter has overflowed. The OVF flag is a sticky flag that indicates that the counter has overflowed at least once since the OVF bit was last cleared.

To halt counting, the CCCR enable flag for the counter must be cleared.

The following procedural step shows how to stop event counting. This step is a continuation of the setup procedure introduced in Section 18.16.6.4, “Reading a Performance Counter’s Count.”

11. To stop event counting, execute a WRMSR instruction to clear the CCCR enable flag for the performance counter.

To halt a cascaded counter (a counter that was started when its alternate counter overflowed), either clear the Cascade flag in the cascaded counter’s CCCR MSR or clear the OVF flag in the alternate counter’s CCCR MSR.

18.16.6.6 Cascading Counters

As described in Section 18.16.2, “Performance Counters,” eighteen performance counters are implemented in pairs. Nine pairs of counters and associated CCCRs are further organized as four blocks: BPU, MS, FLAME, and IQ (see Table 18-18). The first three blocks contain two pairs each. The IQ block contains three pairs of counters (12 through 17) with associated CCCRs (MSR_IQ_CCCR0 through MSR_IQ_CCCR5).

The first 8 counter pairs (0 through 15) can be programmed using ESCRs to detect performance monitoring events. Pairs of ESCRs in each of the four blocks allow many different types of events to be counted. The cascade flag in the CCCR MSR allows nested monitoring of events to be performed by cascading one counter to a second counter located in another pair in the same block (see Figure 18-26 for the location of the flag).

Counters 0 and 1 form the first pair in the BPU block. Either counter 0 or 1 can be programmed to detect an event via MSR_MO B_ESCR0. Counters 0 and 2 can be cascaded in any order, as can counters 1 and 3. It’s possible to set up 4 counters in the same block to cascade on two pairs of independent events. The pairing described also applies to subsequent blocks. Since the IQ PUB has two extra counters, cascading operates somewhat differently if 16 and 17 are involved. In the IQ block, counter 16 can only be cascaded from counter 14 (not from 12); counter 14 cannot be cascaded from counter 16 using the CCCR cascade bit mechanism. Similar restrictions apply to counter 17.

Example 18-1. Counting Events

Assume a scenario where counter X is set up to count 200 occurrences of event A; then counter Y is set up to count 400 occurrences of event B. Each counter is set up to count a specific event and overflow to the next counter. In the above example, counter X is preset for a count of -200 and counter Y for a count of -400; this setup causes the counters to overflow on the 200th and 400th counts respectively.

Continuing this scenario, counter X is set up to count indefinitely and wraparound on overflow. This is described in the basic performance counter setup procedure that begins in Section 18.16.6.1, “Selecting Events to Count.” Counter Y is set up with the cascade flag in its associated CCCR MSR set to 1 and its enable flag set to 0.

To begin the nested counting, the enable bit for the counter X is set. Once enabled, counter X counts until it overflows. At this point, counter Y is automatically enabled and begins counting. Thus counter X overflows after 200 occurrences of event A. Counter Y then starts, counting 400 occurrences of event B before overflowing. When performance counters are cascaded, the counter Y would typically be set up to generate an interrupt on overflow. This is described in Section 18.16.6.8, “Generating an Interrupt on Overflow.”

The cascading counters mechanism can be used to count a single event. The counting begins on one counter then continues on the second counter after the first counter overflows. This technique doubles the number of event counts that can be recorded, since the contents of the two counters can be added together.

18.16.6.7 EXTENDED CASCADING

Extended cascading is a model-specific feature in the Intel NetBurst microarchitecture. The feature is available to Pentium 4 and Xeon processor family with family encoding of 15 and model encoding greater than or equal to 2. This feature uses bit 11 in CCCRs associated with the IQ block. See Table 18-20.

Table 18-20. CCR Names and Bit Positions

CCCR Name:Bit Position	Bit Name	Description
MSR_IQ_CCCR1|2:11	Reserved	
MSR_IQ_CCCR0:11	CASCNT4INTO0	Allow counter 4 to cascade into counter 0
MSR_IQ_CCCR3:11	CASCNT5INTO3	Allow counter 5 to cascade into counter 3
MSR_IQ_CCCR4:11	CASCNT5INTO4	Allow counter 5 to cascade into counter 4
MSR_IQ_CCCR5:11	CASCNT4INTO5	Allow counter 4 to cascade into counter 5

The extended cascading feature can be adapted to the sampling usage model for performance monitoring. However, it is known that performance counters do not generate PMI in cascade mode or extended cascade mode due to an erratum. This erratum applies to Pentium 4 and Intel Xeon processors with model encoding of 2. For Pentium 4 and Intel Xeon processors with model encoding of 0 and 1, the erratum applies to processors with stepping encoding greater than 09H.

Counters 16 and 17 in the IQ block are frequently used in precise event-based sampling or at-retirement counting of events indicating a stalled condition in the pipeline. Neither counter 16 or 17 can initiate the cascading of counter pairs using the cascade bit in a CCCR.

Extended cascading permits performance monitoring tools to use counters 16 and 17 to initiate cascading of two counters in the IQ block. Extended cascading from counter 16 and 17 is conceptually similar to cascading other counters, but instead of using CASCADE bit of a CCCR, one of the four CASCNTxINTOy bits is used.

Example 18-2. Scenario for Extended Cascading

A usage scenario for extended cascading is to sample instructions retired on logical processor 1 after the first 4096 instructions retired on logical processor 0. A procedure to program extended cascading in this scenario is outlined below:

1. Write the value 0 to counter 12.
2. Write the value 04000603H to MSR_CRU_ESCR0 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 1).
3. Write the value 04038800H to MSR_IQ_CCCR0. This enables CASCNT4INTO0 and OVF_PMI. An ISR can sample on instruction addresses in this case (do not set ENABLE, or CASCADE).
4. Write the value FFFFF000H into counter 16.1.
5. Write the value 0400060CH to MSR_CRU_ESCR2 (corresponding to selecting the NBOGNTAG and NBOGTAG event masks with qualification restricted to logical processor 0).
6. Write the value 00039000H to MSR_IQ_CCCR4 (set ENABLE bit, but not OVF_PMI).

Another use for cascading is to locate stalled execution in a multithreaded application. Assume MOB replays in thread B cause thread A to stall. Getting a sample of the stalled execution in this scenario could be accomplished by:

1. Set up counter B to count MOB replays on thread B.
2. Set up counter A to count resource stalls on thread A; set its force overflow bit and the appropriate CASCNTxINTOy bit.
3. Use the performance monitoring interrupt to capture the program execution data of the stalled thread.

18.16.6.8 Generating an Interrupt on Overflow

Any performance counter can be configured to generate a performance monitor interrupt (PMI) if the counter overflows. The PMI interrupt service routine can then collect information about the state of the processor or program when overflow occurred. This information can then be used with a tool like the Intel® VTune™ Performance Analyzer to analyze and tune program performance.

To enable an interrupt on counter overflow, the OVR_PMI flag in the counter's associated CCCR MSR must be set. When overflow occurs, a PMI is generated through the local APIC. (Here, the performance counter entry in the local vector table [LVT] is set up to deliver the interrupt generated by the PMI to the processor.)

The PMI service routine can use the OVF flag to determine which counter overflowed when multiple counters have been configured to generate PMIs. Also, note that these processors mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

When generating interrupts on overflow, the performance counter being used should be preset to value that will cause an overflow after a specified number of events are counted plus 1. The simplest way to select the preset value is to write a negative number into the counter, as described in Section 18.16.6.6, "Cascading Counters." Here, however, if an interrupt is to be generated after 100 event counts, the counter should be preset to minus 100 plus 1 (-100 + 1), or -99. The counter will then overflow after it counts 99 events and generate an interrupt on the next (100th) event counted. The difference of 1 for this count enables the interrupt to be generated immediately after the selected event count has been reached, instead of waiting for the overflow to be propagation through the counter.

Because of latency in the microarchitecture between the generation of events and the generation of interrupts on overflow, it is sometimes difficult to generate an interrupt close to an event that caused it. In these situations, the FORCE_OVF flag in the CCCR can be used to improve reporting. Setting this flag causes the counter to overflow on every counter increment, which in turn triggers an interrupt after every counter increment.

18.16.6.9 Counter Usage Guideline

There are some instances where the user must take care to configure counting logic properly, so that it is not powered down. To use any ESCR, even when it is being used just for tagging, (any) one of the counters that the particular ESCR (or its paired ESCR) can be connected to should be enabled. If this is not done, 0 counts may result. Likewise, to use any counter, there must be some event selected in a corresponding ESCR (other than no_event, which generally has a select value of 0).

18.16.7 At-Retirement Counting

At-retirement counting provides a means counting only events that represent work committed to architectural state and ignoring work that was performed speculatively and later discarded.

The Intel NetBurst microarchitecture used in the Pentium 4 and Intel Xeon processors performs many speculative activities in an attempt to increase effective processing speeds. One example of this speculative activity is branch prediction. The Pentium 4 and Intel Xeon processors typically predict the direction of branches and then decode and execute instructions down the predicted path in anticipation of the actual branch decision. When a branch misprediction occurs, the results of instructions that were decoded and executed down the mispredicted path are canceled. If a performance counter was set up to count all executed instructions, the count would include instructions whose results were canceled as well as those whose results committed to architectural state.

To provide finer granularity in event counting in these situations, the performance monitoring facilities provided in the Pentium 4 and Intel Xeon processors provide a mechanism for tagging events and then counting only those tagged events that represent committed results. This mechanism is called “at-retirement counting.”

Tables A-8 through A-12 list predefined at-retirement events and event metrics that can be used to for tagging events when using at retirement counting. The following terminology is used in describing at-retirement counting:

- **Bogus, non-bogus, retire —** In at-retirement event descriptions, the term “bogus” refers to instructions or μops that must be canceled because they are on a path taken from a mispredicted branch. The terms “retired” and “non-bogus” refer to instructions or μops along the path that results in committed architectural state changes as required by the program being executed. Thus instructions and μops are either bogus or non-bogus, but not both. Several of the Pentium 4 and Intel Xeon processors’ performance monitoring events (such as, Instruction_Retired and Uops_Retired in Table A-8) can count instructions or μops that are retired based on the characterization of bogus” versus non-bogus.
- **Tagging —** Tagging is a means of marking μops that have encountered a particular performance event so they can be counted at retirement. During the course of execution, the same event can happen more than once per μop and a direct count of the event would not provide an indication of how many μops encountered that event.

  The tagging mechanisms allow a μop to be tagged once during its lifetime and thus counted once at retirement. The retired suffix is used for performance metrics that increment a count once per μop, rather than once per event. For example, a μop may encounter a cache miss more than once during its life time, but a “Miss Retired” metric (that counts the number of retired μops that encountered a cache miss) will increment only once for that μop. A “Miss Retired” metric would be useful for characterizing the performance of the cache hierarchy for a particular instruction sequence. Details of various performance metrics and how these can be constructed using the Pentium 4 and Intel Xeon processors performance events are provided in the *Intel Pentium 4 Processor Optimization Reference Manual* (see Section 1.4, “Related Literature”).

- **Replay —** To maximize performance for the common case, the Intel NetBurst microarchitecture aggressively schedules μops for execution before all the conditions for correct execution are guaranteed to be satisfied. In the event that all of these conditions are not satisfied, μops must be reissued. The mechanism that the Pentium 4 and Intel Xeon processors use for this reissuing of μops is called replay. Some examples of replay causes are cache misses, dependence violations, and unforeseen resource constraints. In normal operation, some number of replays is common and unavoidable. An excessive number of replays is an indication of a performance problem.
- **Assist —** When the hardware needs the assistance of microcode to deal with some event, the machine takes an assist. One example of this is an underflow condition in the input operands of a floating-point operation. The hardware must internally modify the format of the operands in order to perform the computation. Assists clear the entire machine of μops before they begin and are costly.

18.16.7.1 Using At-Retirement Counting

The Pentium 4 and Intel Xeon processors allow counting both events and μops that encountered a specified event. For a subset of the at-retirement events listed in Table A-8, a μop may be tagged when it encounters that event. The tagging mechanisms can be used in non-precise event-based sampling, and a subset of these mechanisms can be used in PEBS. There are four independent tagging mechanisms, and each mechanism uses a different event to count μops tagged with that mechanism:

- **Front-end tagging —** This mechanism pertains to the tagging of μops that encountered front-end events (for example, trace cache and instruction counts) and are counted with the Front_end_event event
- **Execution tagging —** This mechanism pertains to the tagging of μops that encountered execution events (for example, instruction types) and are counted with the Execution_Event event.
- **Replay tagging —** This mechanism pertains to tagging of μops whose retirement is replayed (for example, a cache miss) and are counted with the Replay_event event. Branch mispredictions are also tagged with this mechanism.
- **No tags —** This mechanism does not use tags. It uses the Instr_retired and the Uops_ retired events.

Each tagging mechanism is independent from all others; that is, a μop that has been tagged using one mechanism will not be detected with another mechanism’s tagged-μop detector. For example, if μops are tagged using the front-end tagging mechanisms, the Replay_event will not count those as tagged μops unless they are also tagged using the replay tagging mechanism. However, execution tags allow up to four different types of μops to be counted at retirement through execution tagging.

The independence of tagging mechanisms does not hold when using PEBS. When using PEBS, only one tagging mechanism should be used at a time.

Certain kinds of μops that cannot be tagged, including I/O, uncacheable and locked accesses, returns, and far transfers.

Table A-8 lists the performance monitoring events that support at-retirement counting: specifically the Front_end_event, Execution_event, Replay_event, Inst_retired and Uops_retired events. The following sections describe the tagging mechanisms for using these events to tag μop and count tagged μops.

18.16.7.2 Tagging Mechanism for Front_end_event

The Front_end_event counts μops that have been tagged as encountering any of the following events:

- **μop decode events —** Tagging μops for μop decode events requires specifying bits in the ESCR associated with the performance-monitoring event, Uop_type.
- **Trace cache events —** Tagging μops for trace cache events may require specifying certain bits in the MSR_TC_PRECISE_EVENT MSR (see Table A-10).

Table A-8 describes the Front_end_event and Table A-10 describes metrics that are used to set up a Front_end_event count.

The MSRs specified in the Table A-8 that are supported by the front-end tagging mechanism must be set and one or both of the NBOGUS and BOGUS bits in the Front_end_event event mask must be set to count events. None of the events currently supported requires the use of the MSR_TC_PRECISE_EVENT MSR.

18.16.7.3 Tagging Mechanism For Execution_event

Table A-8 describes the Execution_event and Table A-11 describes metrics that are used to set up an Execution_event count.

The execution tagging mechanism differs from other tagging mechanisms in how it causes tagging. One *upstream* ESCR is used to specify an event to detect and to specify a tag value (bits 5 through 8) to identify that event. A second *downstream* ESCR is used to detect μops that have been tagged with that tag value identifier using Execution_event for the event selection.

The upstream ESCR that counts the event must have its tag enable flag (bit 4) set and must have an appropriate tag value mask entered in its tag value field. The 4-bit tag value mask specifies which of tag bits should be set for a particular μop. The value selected for the tag value should coincide with the event mask selected in the downstream ESCR. For example, if a tag value of 1 is set, then the event mask of NBOGUS0 should be enabled, correspondingly in the downstream ESCR. The downstream ESCR detects and counts tagged μops. The normal (not tag value) mask bits in the downstream ESCR specify which tag bits to count. If any one of the tag bits selected by the mask is set, the related counter is incremented by one. This mechanism is summarized in the Table A-11 metrics that are supported by the execution tagging mechanism. The tag enable and tag value bits are irrelevant for the downstream ESCR used to select the Execution_event.

The four separate tag bits allow the user to simultaneously but distinctly count up to four execution events at retirement. (This applies for non-precise event-based sampling. There are additional restrictions for PEBS as noted in Section 18.16.8.3, “Setting Up the PEBS Buffer.”) It is also possible to detect or count combinations of events by setting multiple tag value bits in the upstream ESCR or multiple mask bits in the downstream ESCR. For example, use a tag value of 3H in the upstream ESCR and use NBOGUS0/NBOGUS1 in the downstream ESCR event mask.

18.16.7.4 Tagging Mechanism for Replay_event

Table A-8 describes the Replay_event and Table A-12 describes metrics that are used to set up an Replay_event count.

The replay mechanism enables tagging of μops for a subset of all replays before retirement. Use of the replay mechanism requires selecting the type of μop that may experience the replay in the MSR_PEBS_MATRIX_VERT MSR and selecting the type of event in the MSR_PEBS_ENABLE MSR. Replay tagging must also be enabled with the UOP_Tag flag (bit 24) in the MSR_PEBS_ENABLE MSR.

The Table A-12 lists the metrics that are support the replay tagging mechanism and the at-retirement events that use the replay tagging mechanism, and specifies how the appropriate MSRs need to be configured. The replay tags defined in Table A-5 also enable Precise Event-Based Sampling (PEBS, see Section 15.9.8). Each of these replay tags can also be used in normal sampling by not setting Bit 24 nor Bit 25 in IA_32_PEBS_ENABLE_MSR. Each of these metrics requires that the Replay_Event (see Table A-8) be used to count the tagged μops.

18.16.8 Precise Event-Based Sampling (PEBS)

The debug store (DS) mechanism in processors based on Intel NetBurst microarchitecture allow two types of information to be collected for use in debugging and tuning programs: PEBS records and BTS records. See Section 18.6.8, “Branch Trace Store (BTS),” for a description of the BTS mechanism.

PEBS permits the saving of precise architectural information associated with one or more performance events in the precise event records buffer, which is part of the DS save area (see Section 18.16.5, “DS Save Area”). To use this mechanism, a counter is configured to overflow after it has counted a preset number of events. After the counter overflows, the processor copies the current state of the general-purpose and EFLAGS registers and instruction pointer into a record in the precise event records buffer. The processor then resets the count in the performance counter and restarts the counter. When the precise event records buffer is nearly full, an interrupt is generated, allowing the precise event records to be saved. A circular buffer is not supported for precise event records.

PEBS is supported only for a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can only be carried out using the one performance counter, the MSR_IQ_COUNTER4 MSR.

In processors based on Intel Core microarchitecture, a similar PEBS mechanism is also supported using IA32_PMC0 and IA32_PERFEVTSEL0 MSRs (See Section 18.14.4).

18.16.8.1 Detection of the Availability of the PEBS Facilities

The DS feature flag (bit 21) returned by the CPUID instruction indicates (when set) the availability of the DS mechanism in the processor, which supports the PEBS (and BTS) facilities. When this bit is set, the following PEBS facilities are available:

- The PEBS_UNAVAILABLE flag in the IA32_MISC_ENABLE MSR indicates (when clear) the availability of the PEBS facilities, including the MSR_PEBS_ENABLE MSR.
- The enable PEBS flag (bit 24) in the MSR_PEBS_ENABLE MSR allows PEBS to be enabled (set) or disabled (clear).
- The IA32_DS_AREA MSR can be programmed to point to the DS save area.

18.16.8.2 Setting Up the DS Save Area

Section 18.6.8.2, “Setting Up the DS Save Area,” describes how to set up and enable the DS save area. This procedure is common for PEBS and BTS.

18.16.8.3 Setting Up the PEBS Buffer

Only the MSR_IQ_COUNTER4 performance counter can be used for PEBS. Use the following procedure to set up the processor and this 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, and precise event interrupt threshold, and precise event counter reset fields of the DS buffer management area (see Figure 18-27) to set up the precise event records buffer in memory.
2. Enable PEBS. Set the Enable PEBS flag (bit 24) in MSR_PEBS_ENABLE MSR.
3. Set up the MSR_IQ_COUNTER4 performance counter and its associated CCCR and one or more ESCRs for PEBS as described in Tables A-8 through A-12.

18.16.8.4 Writing a PEBS Interrupt Service Routine

The PEBS facilities share the same interrupt vector and interrupt service routine (called the DS ISR) with the non-precise event-based sampling and BTS facilities. To handle PEBS interrupts, PEBS handler code must be included in the DS ISR. See Section 18.6.8.5, “Writing the DS Interrupt Service Routine,” for guidelines for writing the DS ISR.

18.16.8.5 Other DS Mechanism Implications

The DS mechanism is not available in the SMM. It is disabled on transition to the SMM mode. Similarly the DS mechanism is disabled on the generation of a machine check exception and is cleared on processor RESET and INIT.

The DS mechanism is available in real address mode.

18.16.9 Operating System Implications

The DS mechanism can be used by the operating system as a debugging extension to facilitate failure analysis. When using this facility, a 25 to 30 times slowdown can be expected due to the effects of the trace store occurring on every taken branch.

Depending upon intended usage, the instruction pointers that are part of the branch records or the PEBS records need to have an association with the corresponding process. One solution requires the ability for the DS specific operating system module to be chained to the context switch. A separate buffer can then be maintained for each process of interest and the MSR pointing to the configuration area saved and setup appropriately on each context switch.

If the BTS facility has been enabled, then it must be disabled and state stored on transition of the system to a sleep state in which processor context is lost. The state must be restored on return from the sleep state.

It is required that an interrupt gate be used for the DS interrupt as opposed to a trap gate to prevent the generation of an endless interrupt loop.

Pages that contain buffers must have mappings to the same physical address for all processes/logical processors, such that any change to CR3 will not change DS addresses. If this requirement cannot be satisfied (that is, the feature is enabled on a per thread/process basis), then the operating system must ensure that the feature is enabled/disabled appropriately in the context switch code.

18.17 PERFORMANCE MONITORING AND HYPER-THREADING TECHNOLOGY IN PROCESSORS BASED ON INTEL NETBURST MICROARCHITECTURE

The performance monitoring capability of processors based on Intel NetBurst microarchitecture and supporting Hyper-Threading Technology is similar to that described in Section 18.16. However, the capability is extended so that:

- Performance counters can be programmed to select events qualified by logical processor IDs.
- Performance monitoring interrupts can be directed to a specific logical processor within the physical processor.

The sections below describe performance counters, event qualification by logical processor ID, and special purpose bits in ESCRs/CCCRs. They also describe MSR_PEBS_ENABLE, MSR_PEBS_MATRIX_VERT, and MSR_TC_PRECISE_EVENT.

18.17.1 ESCR MSRs

Figure 18-34 shows the layout of an ESCR MSR in processors supporting Hyper-Threading Technology.

The functions of the flags and fields are as follows:

- **T1_USR flag, bit 0 —** When set, events are counted when thread 1 (logical processor 1) is executing at a current privilege level (CPL) of 1, 2, or 3. These privilege levels are generally used by application code and unprotected operating system code.

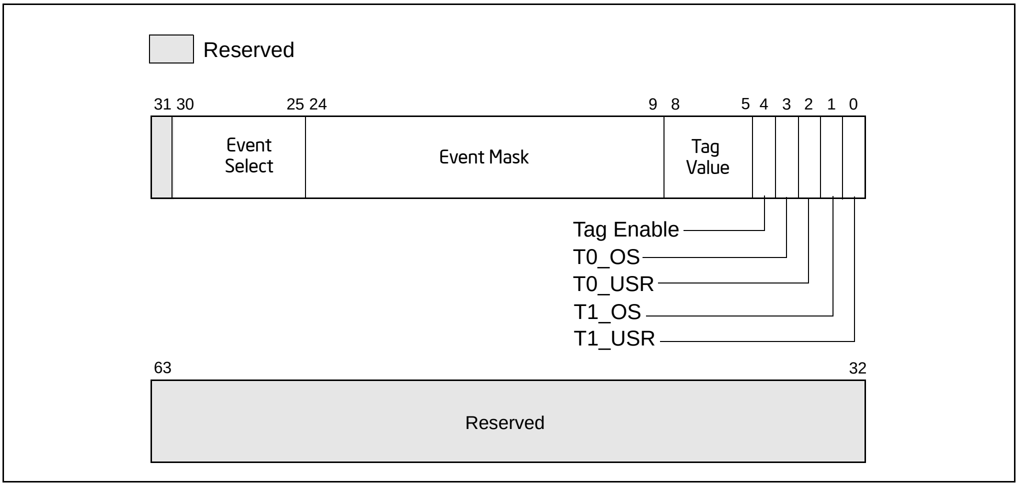

Figure 18-34. Event Selection Control Register (ESCR) for the Pentium 4 Processor, Intel Xeon Processor and Intel Xeon Processor MP Supporting Hyper-Threading Technology

- **T1_OS flag, bit 1 —** When set, events are counted when thread 1 (logical processor 1) is executing at CPL of 0. This privilege level is generally reserved for protected operating system code. (When both the T1_OS and T1_USR flags are set, thread 1 events are counted at all privilege levels.)
- **T0_USR flag, bit 2 —** When set, events are counted when thread 0 (logical processor 0) is executing at a CPL of 1, 2, or 3.
- **T0_OS flag, bit 3 —** When set, events are counted when thread 0 (logical processor 0) is executing at CPL of 0. (When both the T0_OS and T0_USR flags are set, thread 0 events are counted at all privilege levels.)

- **Tag enable, bit 4 —** When set, enables tagging of μops to assist in at-retirement event counting; when clear, disables tagging. See Section 18.16.7, “At-Retirement Counting.”
- **Tag value field, bits 5 through 8 —** Selects a tag value to associate with a μop to assist in at-retirement event counting.
- **Event mask field, bits 9 through 24 —** Selects events to be counted from the event class selected with the event select field.
- **Event select field, bits 25 through 30) —** Selects a class of events to be counted. The events within this class that are counted are selected with the event mask field.

The T0_OS and T0_USR flags and the T1_OS and T1_USR flags allow event counting and sampling to be specified for a specific logical processor (0 or 1) within an Intel Xeon processor MP (See also: Section 7.5.5, “Identifying Logical Processors in an MP System,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*).

Not all performance monitoring events can be detected within an Intel Xeon processor MP on a per logical processor basis (see Section 18.17.4, “Performance Monitoring Events”). Some sub-events (specified by an event mask bits) are counted or sampled without regard to which logical processor is associated with the detected event.

18.17.2 CCCR MSRs

Figure 18-35 shows the layout of a CCCR MSR in processors supporting Hyper-Threading Technology. The functions of the flags and fields are as follows:

- **Enable flag, bit 12 —** When set, enables counting; when clear, the counter is disabled. This flag is cleared on reset
- **ESCR select field, bits 13 through 15 —** Identifies the ESCR to be used to select events to be counted with the counter associated with the CCCR.
- **Active thread field, bits 16 and 17 —** Enables counting depending on which logical processors are active (executing a thread). This field enables filtering of events based on the state (active or inactive) of the logical processors. The encodings of this field are as follows:

  00 — None. Count only when neither logical processor is active.

  01 — Single. Count only when one logical processor is active (either 0 or 1).

  10 — Both. Count only when both logical processors are active.

  11 — Any. Count when either logical processor is active.

  A halted logical processor or a logical processor in the “wait for SIPI” state is considered inactive.

- **Compare flag, bit 18 —** When set, enables filtering of the event count; when clear, disables filtering. The filtering method is selected with the threshold, complement, and edge flags.

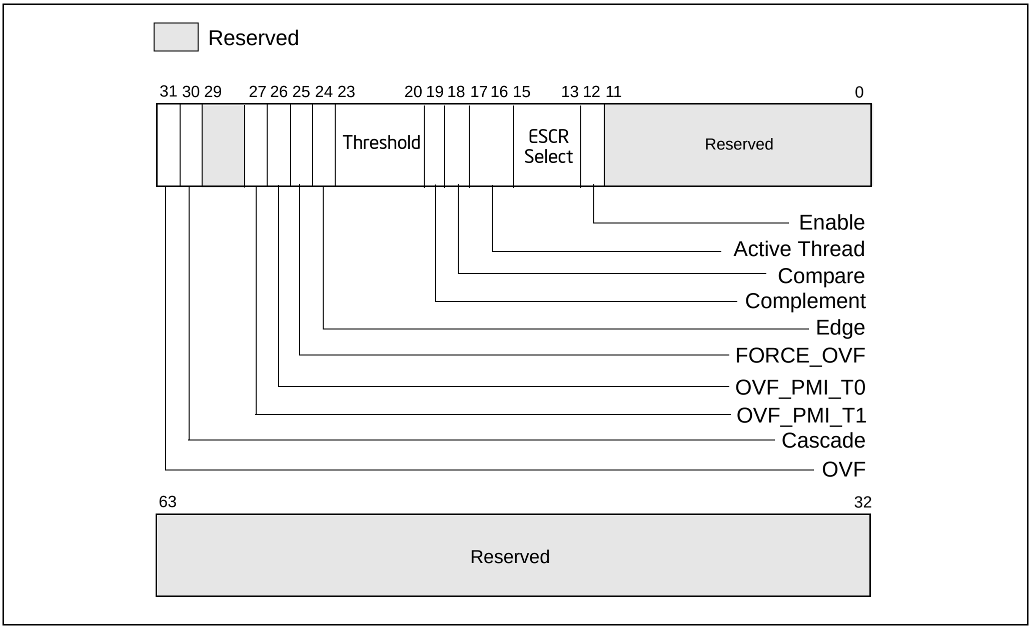

Figure 18-35. Counter Configuration Control Register (CCCR)

- **Complement flag, bit 19 —** Selects how the incoming event count is compared with the threshold value. When set, event counts that are less than or equal to the threshold value result in a single count being delivered to the performance counter; when clear, counts greater than the threshold value result in a count being delivered to the performance counter (see Section 18.16.6.2, "Filtering Events"). The compare flag is not active unless the compare flag is set.
- **Threshold field, bits 20 through 23 —** Selects the threshold value to be used for comparisons. The processor examines this field only when the compare flag is set, and uses the complement flag setting to determine the type of threshold comparison to be made. The useful range of values that can be entered in this field depend on the type of event being counted (see Section 18.16.6.2, "Filtering Events").
- **Edge flag, bit 24 —** When set, enables rising edge (false-to-true) edge detection of the threshold comparison output for filtering event counts; when clear, rising edge detection is disabled. This flag is active only when the compare flag is set.

- **FORCE_OVF flag, bit 25 —** When set, forces a counter overflow on every counter increment; when clear, overflow only occurs when the counter actually overflows.
- **OVF_PMI_T0 flag, bit 26 —** When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 0 when the counter overflows occurs; when clear, disables PMI generation for logical processor 0. Note that the PMI is generate on the next event count after the counter has overflowed.
- **OVF_PMI_T1 flag, bit 27 —** When set, causes a performance monitor interrupt (PMI) to be sent to logical processor 1 when the counter overflows occurs; when clear, disables PMI generation for logical processor 1. Note that the PMI is generate on the next event count after the counter has overflowed.
- **Cascade flag, bit 30 —** When set, enables counting on one counter of a counter pair when its alternate counter in the other the counter pair in the same counter group overflows (see Section 18.16.2, “Performance Counters,” for further details); when clear, disables cascading of counters.
- **OVF flag, bit 31 —** Indicates that the counter has overflowed when set. This flag is a sticky flag that must be explicitly cleared by software.

18.17.3 IA32_PEBS_ENABLE MSR

In a processor supporting Hyper-Threading Technology and based on the Intel NetBurst microarchitecture, PEBS is enabled and qualified with two bits in the MSR_PEBS_ENABLE MSR: bit 25 (ENABLE_PEBS_MY_THR) and 26 (ENABLE_PEBS_OTH_THR) respectively. These bits do not explicitly identify a specific logical processor by logic processor ID(T0 or T1); instead, they allow a software agent to enable PEBS for subsequent threads of execution on the same logical processor on which the agent is running (“my thread”) or for the other logical processor in the physical package on which the agent is not running (“other thread”).

PEBS is supported for only a subset of the at-retirement events: Execution_event, Front_end_event, and Replay_event. Also, PEBS can be carried out only with two performance counters: MSR_IQ_CCCR4 (MSR address 370H) for logical processor 0 and MSR_IQ_CCCR5 (MSR address 371H) for logical processor 1.

Performance monitoring tools should use a processor affinity mask to bind the kernel mode components that need to modify the ENABLE_PEBS_MY_THR and ENABLE_PEBS_OTH_THR bits in the MSR_PEBS_ENABLE MSR to a specific logical processor. This is to prevent these kernel mode components from migrating between different logical processors due to OS scheduling.

18.17.4 Performance Monitoring Events

All of the events listed in Table A-7 and A-8 are available in an Intel Xeon processor MP. When Hyper-Threading Technology is active, many performance monitoring events can be can be qualified by the logical processor ID, which corresponds to bit 0 of the initial APIC ID. This allows for counting an event in any or all of the logical processors. However, not all the events have this logic processor specificity, or thread specificity.

Here, each event falls into one of two categories:

- **Thread specific (TS) —** The event can be qualified as occurring on a specific logical processor.
- **Thread independent (TI) —** The event cannot be qualified as being associated with a specific logical processor.

Table A-13 gives logical processor specific information (TS or TI) for each of the events described in Tables A-7 and A-8. If for example, a TS event occurred in logical processor T0, the counting of the event (as shown in Table 18-21) depends only on the setting of the T0_USR and T0_OS flags in the ESCR being used to set up the event counter. The T1_USR and T1_OS flags have no effect on the count.

Table 18-21. Effect of Logical Processor and CPL Qualification for Logical-Processor-Specific (TS) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while T1 in USR	Counts while T1 in OS or USR	Counts while T1 in OS
T0_OS/T0_USR = 01	Counts while T0 in USR	Counts while T0 in USR or T1 in USR	Counts while (a) T0 in USR or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 11	Counts while T0 in OS or USR	Counts while (a) T0 in OS or (b) T0 in USR or (c) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) or T0 in USR or (c) T1 in OS
T0_OS/T0_USR = 10	Counts T0 in OS	Counts T0 in OS or T1 in USR	Counts while (a)T0 in Os or (b) T1 in OS or (c) T1 in USR	Counts while (a) T0 in OS or (b) T1 in OS

When a bit in the event mask field is TI, the effect of specifying bit-0-3 of the associated ESCR are described in Table 15-6. For events that are marked as TI in Appendix A, the effect of selectively specifying T0_USR, T0_OS, T1_USR, T1_OS bits is shown in Table 18-22.

Table 18-22. Effect of Logical Processor and CPL Qualification for Non-logical-Processor-specific (TI) Events

	T1_OS/T1_USR = 00	T1_OS/T1_USR = 01	T1_OS/T1_USR = 11	T1_OS/T1_USR = 10
T0_OS/T0_USR = 00	Zero count	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS
T0_OS/T0_USR = 01	Counts while (a) T0 in USR or (b) T1 in USR	Counts while (a) T0 in USR or (b) T1 in USR	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 11	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1
T0_OS/T0_USR = 0	Counts while (a) T0 in OS or (b) T1 in OS	Counts irrespective of CPL, T0, T1	Counts irrespective of CPL, T0, T1	Counts while (a) T0 in OS or (b) T1 in OS

18.18 COUNTING CLOCKS

The count of cycles, also known as clockticks, forms a the basis for measuring how long a program takes to execute. Clockticks are also used as part of efficiency ratios like cycles per instruction (CPI). Processor clocks may stop ticking under circumstances like the following:

- The processor is halted when there is nothing for the CPU to do. For example, the processor may halt to save power while the computer is servicing an I/O request. When Hyper-Threading Technology is enabled, both logical processors must be halted for performance-monitoring counters to be powered down.
- The processor is asleep as a result of being halted or because of a power-management scheme. There are different levels of sleep. In the some deep sleep levels, the time-stamp counter stops counting.

In addition, processor core clocks may undergo transitions at different ratios relative to the processor's bus clock frequency. Some of the situations that can cause processor core clock to undergo frequency transitions include:

- TM2 transitions
- Enhanced Intel SpeedStep Technology transitions (P-state transitions)

For Intel processors that support Intel Dynamic Acceleration or XE operation, the processor core clocks may operate at a frequency that differs from the maximum qualified frequency (as indicated by brand string information reported by CPUID instruction). See Section 18.18.5 for more detail.

There are several ways to count processor clock cycles to monitor performance. These are:

- **Non-halted clockticks —** Measures clock cycles in which the specified logical processor is not halted and is not in any power-saving state. When Hyper-Threading Technology is enabled, ticks can be measured on a per-logical-processor basis. There are also performance events on dual-core processors that measure clockticks per logical processor when the processor is not halted.
- **Non-sleep clockticks —** Measures clock cycles in which the specified physical processor is not in a sleep mode or in a power-saving state. These ticks can**n**ot be measured on a logical-processor basis.
- **Time-stamp counter —** Measures clock cycles in which the physical processor is not in deep sleep. These ticks cannot be measured on a logical-processor basis.
- **Reference clockticks —** TM2 or Enhanced Intel SpeedStep technology are two examples of processor features that can cause processor core clockticks to represent non-uniform tick intervals due to change of bus ratios. Performance events that counts clockticks of a constant reference frequency was introduced Intel Core Duo and Intel Core Solo processors. The mechanism is further enhanced on processors based on Intel Core microarchitecture.

Some processor models permit clock cycles to be measured when the physical processor is not in deep sleep (by using the time-stamp counter and the RDTSC instruction). Note that such ticks cannot be measured on a per-logical-processor basis. See Section 18.10, “Time-Stamp Counter,” for detail on processor capabilities.

The first two methods use performance counters and can be set up to cause an interrupt upon overflow (for sampling). They may also be useful where it is easier for a tool to read a performance counter than to use a time stamp counter (the timestamp counter is accessed using the RDTSC instruction).

For applications with a significant amount of I/O, there are two ratios of interest:

- **Non-halted CPI —** Non-halted clockticks/instructions retired measures the CPI for phases where the CPU was being used. This ratio can be measured on a logical-processor basis when Hyper-Threading Technology is enabled.
- **Nominal CPI —** Time-stamp counter ticks/instructions retired measures the CPI over the duration of a program, including those periods when the machine halts while waiting for I/O.

18.18.1 Non-Halted Clockticks

Use the following procedure to program ESCRs and CCCRs to obtain non-halted clockticks on processors based on Intel NetBurst microarchitecture:

1. Select an ESCR for the global_power_events and specify the RUNNING sub-event mask and the desired T0_OS/T0_USR/T1_OS/T1_USR bits for the targeted processor.

2. Select an appropriate counter.
3. Enable counting in the CCCR for that counter by setting the enable bit.

18.18.2 Non-Sleep Clockticks

Performance monitoring counters can be configured to count clockticks whenever the performance monitoring hardware is not powered-down. To count Non-sleep Clockticks with a performance-monitoring counter, do the following:

1. Select one of the 18 counters.
2. Select any of the ESCRs whose events the selected counter can count. Set its event select to anything other than no_event. This may not seem necessary, but the counter may be disabled if this is not done.
3. Turn threshold comparison on in the CCCR by setting the compare bit to 1.
4. Set the threshold to 15 and the complement to 1 in the CCCR. Since no event can exceed this threshold, the threshold condition is met every cycle and the counter counts every cycle. Note that this overrides any qualification (e.g. by CPL) specified in the ESCR.
5. Enable counting in the CCCR for the counter by setting the enable bit.

In most cases, the counts produced by the non-halted and non-sleep metrics are equivalent if the physical package supports one logical processor and is not placed in a power-saving state. Operating systems may execute an HLT instruction and place a physical processor in a power-saving state.

On processors that support Hyper-Threading Technology (HT), each physical package can support two or more logical processors. Current implementation of HT provides two logical processors for each physical processor. While both logical processors can execute two threads simultaneously, one logical processor may halt to allow the other logical processor to execute without sharing execution resources between two logical processors.

Non-halted Clockticks can be set up to count the number of processor clock cycles for each logical processor whenever the logical processor is not halted (the count may include some portion of the clock cycles for that logical processor to complete a transition to a halted state). Physical processors that support HT enter into a power-saving state if all logical processors halt.

The Non-sleep Clockticks mechanism uses a filtering mechanism in CCCRs. The mechanism will continue to increment as long as one logical processor is not halted or in a power-saving state. Applications may cause a processor to enter into a power-saving state by using an OS service that transfers control to an OS’s idle loop. The idle loop then may place the processor into a power-saving state after an implementation-dependent period if there is no work for the processor.

18.18.3 Incrementing the Time-Stamp Counter

The time-stamp counter increments when the clock signal on the system bus is active and when the sleep pin is not asserted. The counter value can be read with the RDTSC instruction.

The time-stamp counter and the non-sleep clockticks count may not agree in all cases and for all processors. See Section 18.10, “Time-Stamp Counter,” for more information on counter operation.

18.18.4 Non-Halted Reference Clockticks

Software can use either processor-specific performance monitor events (for example: CPU_CLK_UNHALTED.BUS on processors based on the Intel Core microarchitecture, and equivalent event specifications on the Intel Core Duo and Intel Core Solo processors) to count non-halted reference clockticks.

These events count reference clock cycles whenever the specified processor is not halted. The counter counts reference cycles associated with a fixed-frequency clock source irrespective of P-state, TM2, or frequency transitions that may occur to the processor.

18.18.5 Cycle Counting and Opportunistic Processor Operation

As a result of the state transitions due to opportunistic processor performance operation (see Chapter 13, “Power and Thermal Management”), a logical processor or a processor core can operate at frequency different from that indicated by the processor’s maximum qualified frequency.

The following items are expected to hold true irrespective of when opportunistic processor operation causes state transitions:

- The time stamp counter operates at a fixed-rate frequency of the processor.
- The IA32_MPERF counter increments at the same TSC frequency irrespective of any transitions caused by opportunistic processor operation.
- The IA32_FIXED_CTR2 counter increments at the same TSC frequency irrespective of any transitions caused by opportunistic processor operation.
- The Local APIC timer operation is unaffected by opportunistic processor operation.
- The TSC, IA32_MPERF, and IA32_FIXED_CTR2 operate at the same, maximum-resolved frequency of the platform, which is equal to the product of scalable bus frequency and maximum resolved bus ratio.

For processors based on Intel Core microarchitecture, the scalable bus frequency is encoded in the bit field MSR_FSB_FREQ[2:0] at (0CDH), see Appendix B, “Model-

Specific Registers (MSRs)". The maximum resolved bus ratio can be read from the following bit field:

- If XE operation is disabled, the maximum resolved bus ratio can be read in MSR_PLATFORM_ID[12:8]. It corresponds to the maximum qualified frequency.
- IF XE operation is enabled, the maximum resolved bus ratio is given in MSR_PERF_STAT[44:40], it corresponds to the maximum XE operation frequency configured by BIOS.

XE operation of an Intel 64 processor is implementation specific. XE operation can be enabled only by BIOS. If MSR_PERF_STAT[31] is set, XE operation is enabled. The MSR_PERF_STAT[31] field is read-only.

18.19 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 and Intel® Atom™ 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, 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.

18.20 PERFORMANCE MONITORING AND DUAL-CORE TECHNOLOGY

The performance monitoring capability of dual-core processors duplicates the microarchitectural resources of a single-core processor implementation. Each processor core has dedicated performance monitoring resources.

In the case of Pentium D processor, each logical processor is associated with dedicated resources for performance monitoring. In the case of Pentium processor Extreme edition, each processor core has dedicated resources, but two logical processors in the same core share performance monitoring resources (see Section 18.17, “Performance Monitoring and Hyper-Threading Technology in Processors Based on Intel NetBurst Microarchitecture”).

18.21 PERFORMANCE MONITORING ON 64-BIT INTEL XEON PROCESSOR MP WITH UP TO 8-MBYTE L3 CACHE

The 64-bit Intel Xeon processor MP with up to 8-MByte L3 cache has a CPUID signature of family [0FH], model [03H or 04H]. Performance monitoring capabilities available to Pentium 4 and Intel Xeon processors with the same values (see Section 18.11 and Section 18.17) apply to the 64-bit Intel Xeon processor MP with an L3 cache.

The level 3 cache is connected between the system bus and IOQ through additional control logic. See Figure 18-36.

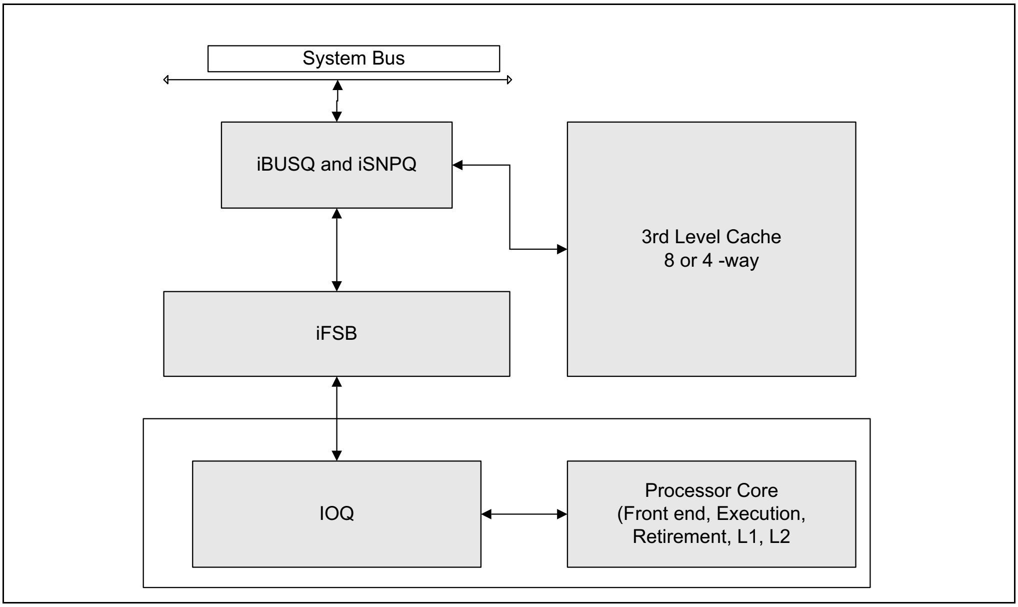

Figure 18-36. Block Diagram of 64-bit Intel Xeon Processor MP with 8-MByte L3

Additional performance monitoring capabilities and facilities unique to 64-bit Intel Xeon processor MP with an L3 cache are described in this section. The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs), each dedicated to a specific event. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values.

The lower 32-bits of the MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers. These performance counters can be accessed using RDPMC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

The performance monitoring capabilities consist of four events. These are:

- **IBUSQ event —** This event detects the occurrence of micro-architectural conditions related to the iBUSQ unit. It provides two MSRs: MSR_IFSB_IBUSQ0 and MSR_IFSB_IBUSQ1. Configure sub-event qualification and enable/disable functions using the high 32 bits of these MSRs. The low 32 bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32 bits. See Figure 18-37.

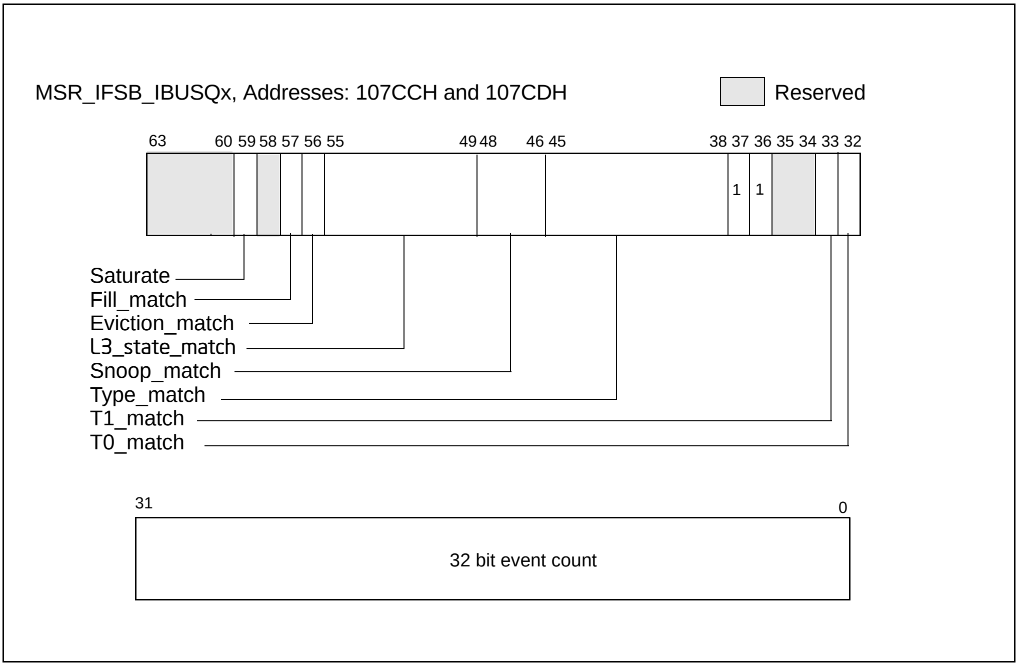

Figure 18-37. MSR_IFSB_IBUSQx, Addresses: 107CCH and 107CDH

- **ISNPQ event —** This event detects the occurrence of microarchitectural conditions related to the iSNPQ unit. It provides two MSRs: MSR_IFSB_ISNPQ0 and MSR_IFSB_ISNPQ1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the MSRs. The low 32-bits act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the upper 32-bits. See Figure 18-38.

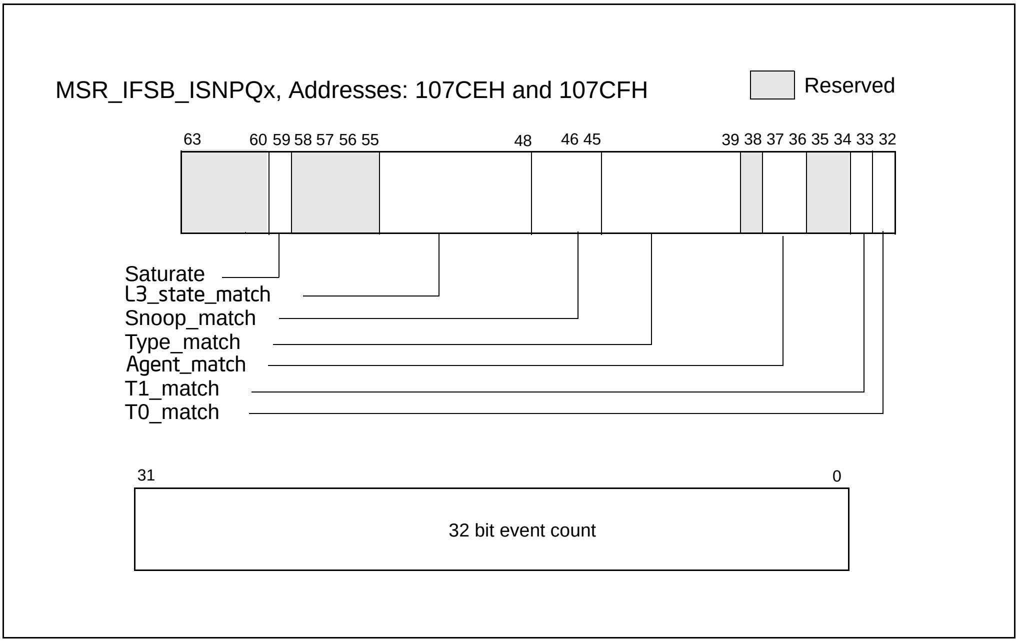

Figure 18-38. MSR_IFSB_ISNPQx, Addresses: 107CEH and 107CFH

- **EFSB event —** This event can detect the occurrence of micro-architectural conditions related to the iFSB unit or system bus. It provides two MSRs: MSR_EFSB_DRDY0 and MSR_EFSB_DRDY1. Configure sub-event qualifications and enable/disable functions using the high 32 bits of the 64-bit MSR. The low 32-bit act as a 32-bit event counter. Counting starts after software writes a non-zero value to one or more of the qualification bits in the upper 32-bits of the MSR. See Figure 18-39.

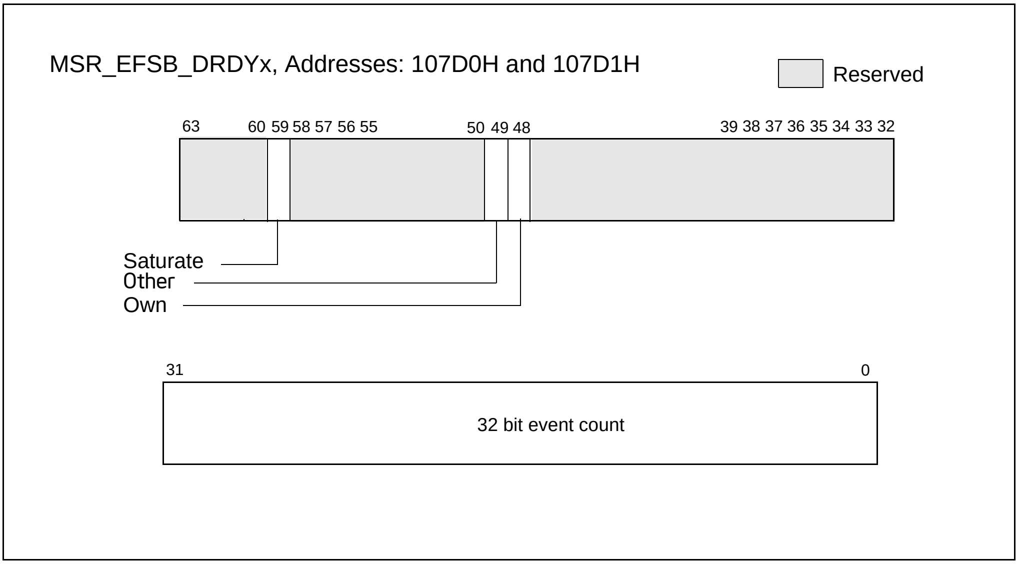

Figure 18-39. MSR_EFSB_DRDYx, Addresses: 107D0H and 107D1H

- **IBUSQ Latency event —** This event accumulates weighted cycle counts for latency measurement of transactions in the iBUSQ unit. The count is enabled by setting MSR_IFSB_CTRL6[bit 26] to 1; the count freezes after software sets MSR_IFSB_CTRL6[bit 26] to 0. MSR_IFSB_CNTR7 acts as a 64-bit event counter for this event. See Figure 18-40.

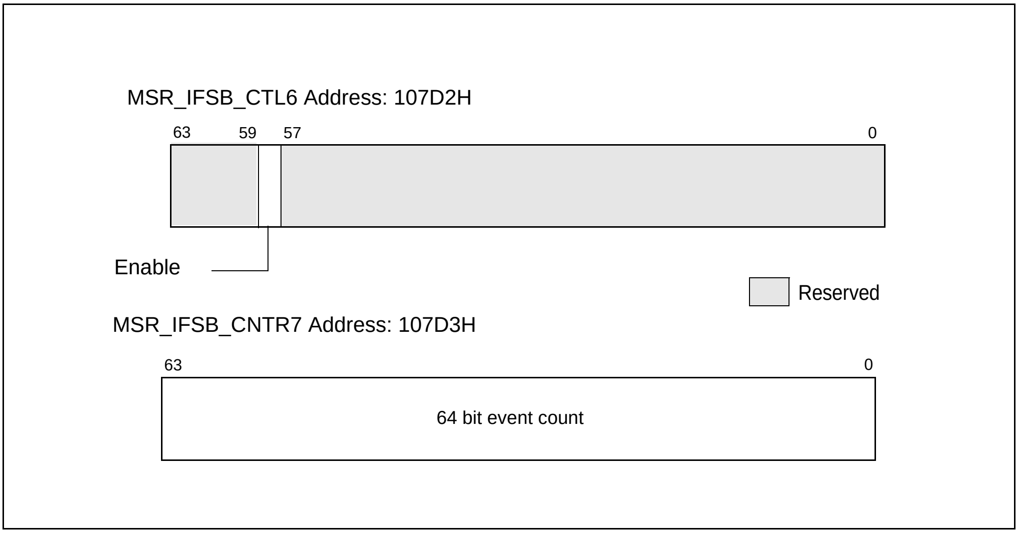

Figure 18-40. MSR_IFSB_CTL6, Address: 107D2H; MSR_IFSB_CNTR7, Address: 107D3H

18.22 PERFORMANCE MONITORING ON DUAL-CORE INTEL XEON PROCESSOR 7100 SERIES

The Dual-Core Intel Xeon processor 7100 Series have a CPUID signature of family [0FH], model [06H] and a unified L3 cache shared between two cores. Each core in an Intel Xeon processor 7100 series supports Intel Hyper-Threading Technology, providing two logical processors per core.

Intel Xeon processor 7100 series are based on Intel NetBurst microarchitecture, but the IOQ logic in each processor core is replaced with a Simple Direct Interface (SDI) logic. The L3 cache is connected between the system bus and the SDI through additional control logic. See Figure 18-41.

Almost all of the performance monitoring capabilities available to processors with the same CPUID signatures (see Section 18.11 and Section 18.17) apply to the Intel Xeon processor 7100 series. The IOQ_allocation and IOQ_active_entries events are not supported. Additional performance monitoring capabilities available to Intel Xeon processor 7100 series are described in this section.

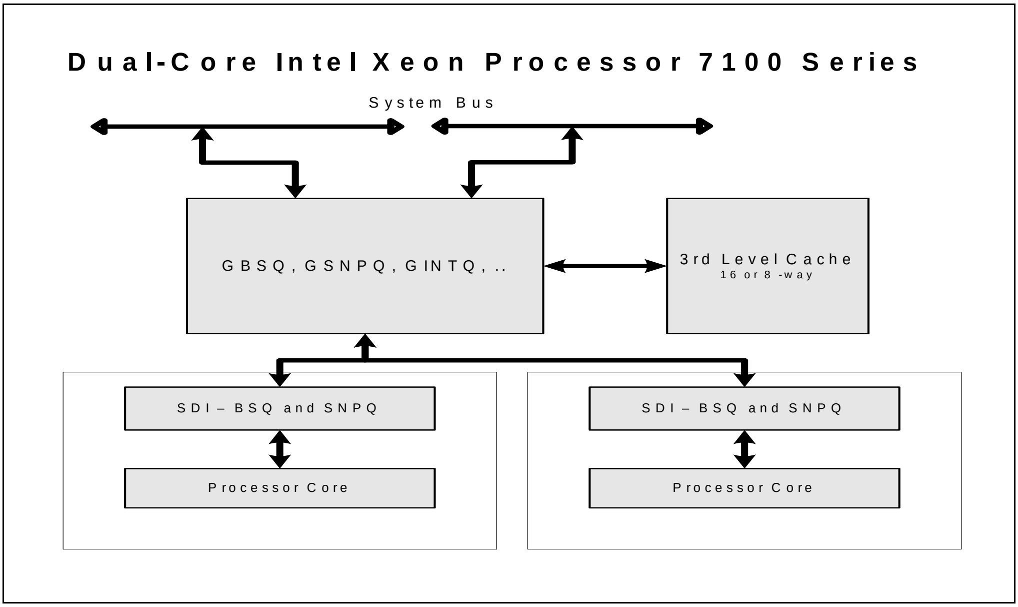

Figure 18-41. Block Diagram of Intel Xeon Processor 7100 Series

The facility for monitoring events consists of a set of dedicated model-specific registers (MSRs). There are eight event select/counting MSRs that are dedicated to counting events associated with specified microarchitectural conditions. Programming of these MSRs requires using RDMSR/WRMSR instructions with 64-bit values. In addition, an MSR MSR_EMON_L3_GL_CTL provides simplified interface to control freezing, resetting, re-enabling operation of any combination of these event select/counting MSRs.

The eight MSRs dedicated to count occurrences of specific conditions are further divided to count three sub-classes of microarchitectural conditions:

- Two MSRs (MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1) are dedicated to counting GBSQ events. Up to two GBSQ events can be programmed and counted simultaneously.
- Two MSRs (MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3) are dedicated to counting GSNPQ events. Up to two GBSQ events can be programmed and counted simultaneously.
- Four MSRs (MSR_EMON_L3_CTR_CTL4, MSR_EMON_L3_CTR_CTL5, MSR_EMON_L3_CTR_CTL6, and MSR_EMON_L3_CTR_CTL7) are dedicated to counting external bus operations.

The bit fields in each of eight MSRs share the following common characteristics:

- Bits 63:32 is the event control field that includes an event mask and other bit fields that control counter operation. The event mask field specifies details of the microarchitectural condition, and its definition differs across GBSQ, GSNPQ, FSB.
- Bits 31:0 is the event count field. If the specified condition is met during each relevant clock domain of the event logic, the matched condition signals the counter logic to increment the associated event count field. The lower 32-bits of these 8 MSRs at addresses 107CC through 107D3 are treated as 32 bit performance counter registers. These performance counters can be accessed using RDPMC instruction with the index starting from 18 through 25. The EDX register returns zero when reading these 8 PMCs.

18.22.1 GBSQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL0 and MSR_EMON_L3_CTR_CTL1 is given in Figure 18-42. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following eight attributes:

- Agent_Select (bits 35:32): Each bit specifies a logical processor in the physical package. The lower two bits corresponds to two logical processors in the first processor core, the upper two bits corresponds to two logical processors in the second processor core. 0FH encoding matches transactions from any logical processor.

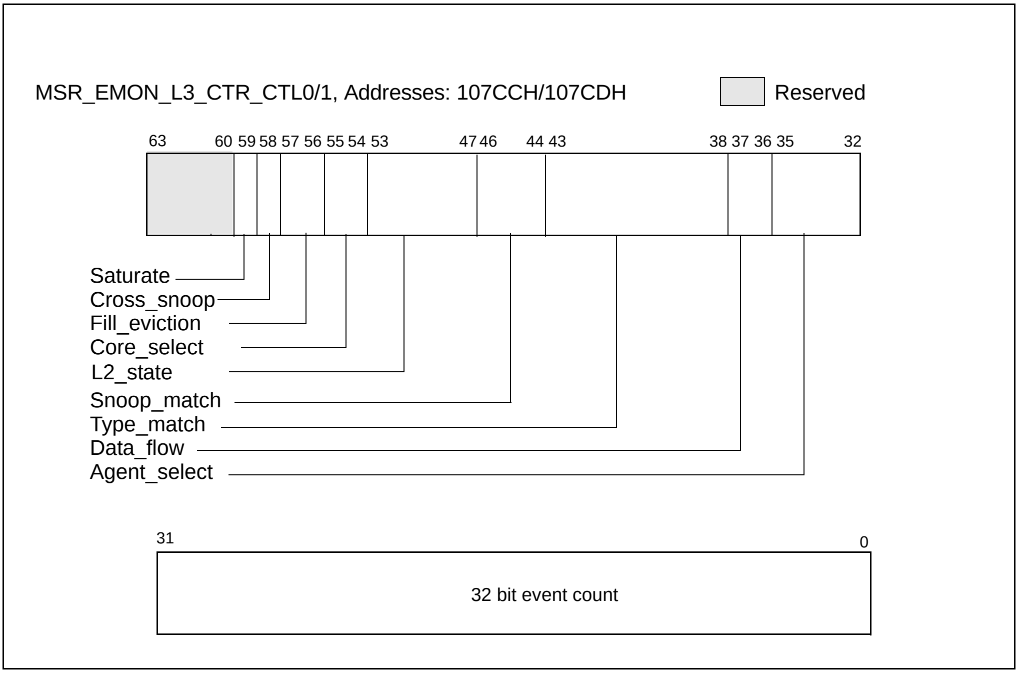

Figure 18-42. MSR_EMON_L3_CTR_CTL0/1, Addresses: 107CCH/107CDH

- Data_Flow (bits 37:36): Bit 36 specifies demand transactions, bit 37 specifies prefetch transactions.
- Type_Match (bits 43:38): Specifies transaction types. If all six bits are set, event count will include all transaction types.
- Snoop_Match: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- L2_State (bits 53:47): Each bit specifies an L2 coherency state.
- Core_Select (bits 55:54): The valid encodings are
  - 00B: Match transactions from any core in the physical package
  - 01B: Match transactions from this core only
  - 10B: Match transactions from the other core in the physical package
  - 11B: Match transaction from both cores in the physical package
- Fill_Eviction (bits 57:56): The valid encodings are
  - 00B: Match any transactions
  - 01B: Match transactions that fill L2

  - 10B: Match transactions that fill L2 without an eviction
  - 11B: Match transaction fill L2 with an eviction
- Cross_Snoop (bit 58): The encodings are \
  - 0B: Match any transactions
  - 1B: Match cross snoop transactions

For each counting clock domain, if all eight attributes match, event logic signals to increment the event count field.

18.22.2 GSNPQ Event Interface

The layout of MSR_EMON_L3_CTR_CTL2 and MSR_EMON_L3_CTR_CTL3 is given in Figure 18-43. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) consists of the following six attributes:

- Agent_Select (bits 37:32): Each of the lowest 4 bits specifies a logical processor in the physical package. The lowest two bits corresponds to two logical processors in the first processor core, the next two bits corresponds to two logical processors in the second processor core. Bit 36 specifies other symmetric agent transactions. Bit 37 specifies central agent transactions. 3FH encoding matches transactions from any logical processor.
- Type_Match (bits 43:38): Specifies transaction types. If all six bits are set, event count will include any transaction types.
- Snoop_Match: (bits 46:44): The three bits specify (in ascending bit position) clean snoop result, HIT snoop result, and HITM snoop results respectively.
- L2_State (bits 53:47): Each bit specifies an L2 coherency state.
- Core_Select (bits 56:54): Bit 56 enables Core_Select matching. If bit 56 is clear, Core_select encoding is ignored. If bit 56 is set, the valid encodings for the lower two bits (bit 55, 54) are
  - 00B: Match transactions from only one core (irrespective which core) in the physical package
  - 01B: Match transactions from this core and not the other core
  - 10B: Match transactions from the other core in the physical package, but not this core
  - 11B: Match transaction from both cores in the physical package
- Block_Snoop (bit 57): specifies blocked snoop.

For each counting clock domain, if all six attributes match, event logic signals to increment the event count field.

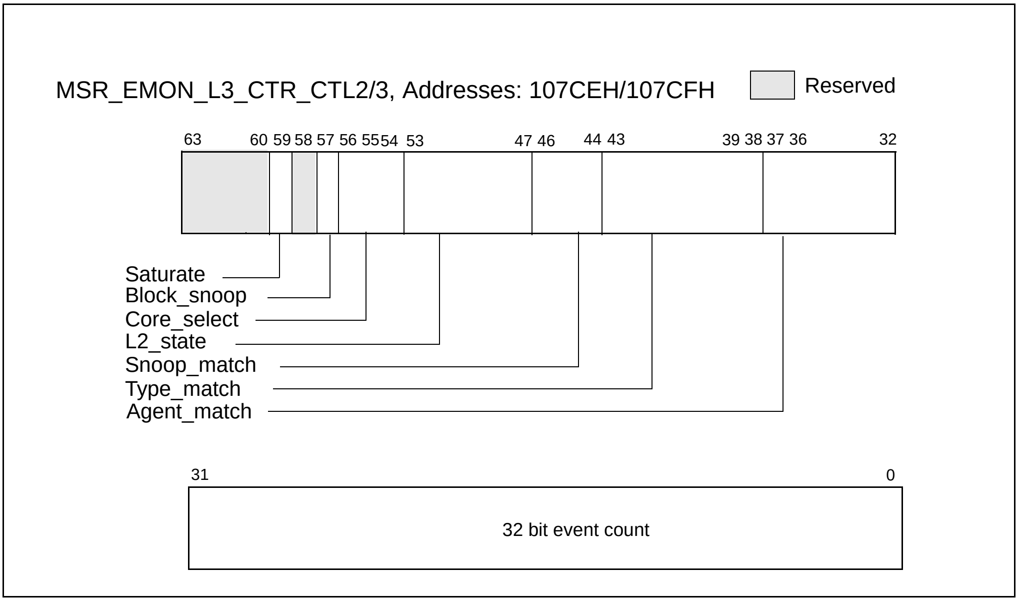

Figure 18-43. MSR_EMON_L3_CTR_CTL2/3, Addresses: 107CEH/107CFH

18.22.3 FSB Event Interface

The layout of MSR_EMON_L3_CTR_CTL4 through MSR_EMON_L3_CTR_CTL7 is given in Figure 18-44. Counting starts after software writes a non-zero value to one or more of the upper 32 bits.

The event mask field (bits 58:32) is organized as follows:

- Bit 58: must set to 1.
- FSB_Submask (bits 57:32): Specifies FSB-specific sub-event mask.

The FSB sub-event mask defines a set of independent attributes. The event logic signals to increment the associated event count field if one of the attribute matches. Some of the sub-event mask bit counts durations. A duration event increments at most once per cycle.

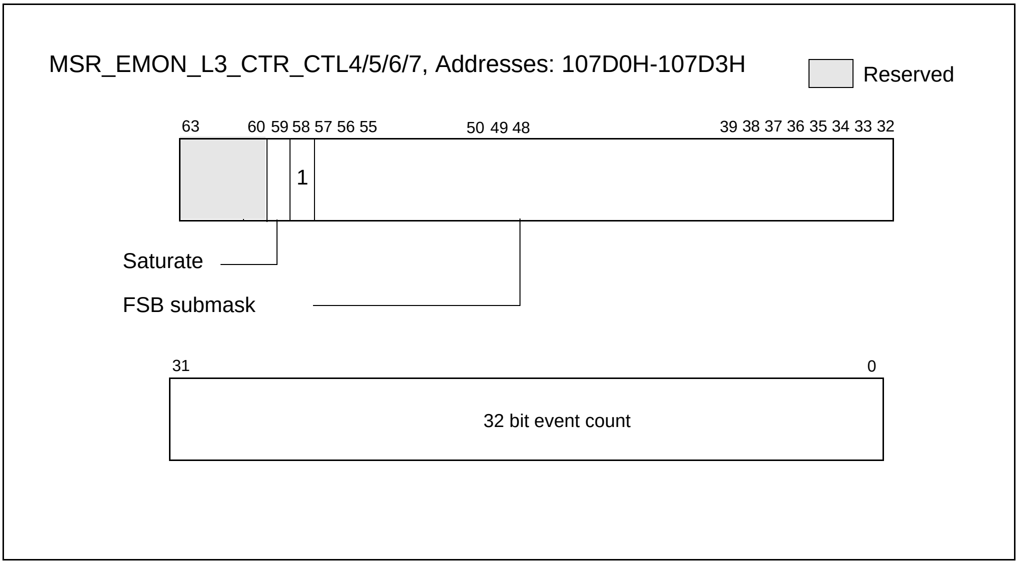

Figure 18-44. MSR_EMON_L3_CTR_CTL4/5/6/7, Addresses: 107D0H-107D3H

18.22.3.1 FSB Sub-Event Mask Interface

- FSB_type (bit 37:32): Specifies different FSB transaction types originated from this physical package
- FSB_L_clear (bit 38): Count clean snoop results from any source for transaction originated from this physical package
- FSB_L_hit (bit 39): Count HIT snoop results from any source for transaction originated from this physical package
- FSB_L_hitm (bit 40): Count HITM snoop results from any source for transaction originated from this physical package
- FSB_L_defer (bit 41): Count DEFER responses to this processor’s transactions
- FSB_L_retry (bit 42): Count RETRY responses to this processor’s transactions
- FSB_L_snoop_stall (bit 43): Count snoop stalls to this processor’s transactions
- FSB_DBSY (bit 44): Count DBSY assertions by this processor (without a concurrent DRDY)
- FSB_DRDY (bit 45): Count DRDY assertions by this processor
- FSB_BNR (bit 46): Count BNR assertions by this processor
- FSB_IOQ_empty (bit 47): Counts each bus clocks when the IOQ is empty
- FSB_IOQ_full (bit 48): Counts each bus clocks when the IOQ is full
- FSB_IOQ_active (bit 49): Counts each bus clocks when there is at least one entry in the IOQ

- FSB_WW_data (bit 50): Counts back-to-back write transaction's data phase.
- FSB_WW_issue (bit 51): Counts back-to-back write transaction request pairs issued by this processor.
- FSB_WR_issue (bit 52): Counts back-to-back write-read transaction request pairs issued by this processor.
- FSB_RW_issue (bit 53): Counts back-to-back read-write transaction request pairs issued by this processor.
- FSB_other_DBSY (bit 54): Count DBSY assertions by another agent (without a concurrent DRDY)
- FSB_other_DRDY (bit 55): Count DRDY assertions by another agent
- FSB_other_snoop_stall (bit 56): Count snoop stalls on the FSB due to another agent
- FSB_other_BNR (bit 57): Count BNR assertions from another agent

18.22.4 Common Event Control Interface

The MSR_EMON_L3_GL_CTL MSR provides simplified access to query overflow status of the GBSQ, GSNPQ, FSB event counters. It also provides control bit fields to freeze, unfreeze, or reset those counters. The following bit fields are supported:

- GL_freeze_cmd (bit 0): Freeze the event counters specified by the GL_event_select field.
- GL_unfreeze_cmd (bit 1): Unfreeze the event counters specified by the GL_event_select field.
- GL_reset_cmd (bit 2): Clear the event count field of the event counters specified by the GL_event_select field. The event select field is not affected.
- GL_event_select (bit 23:16): Selects one or more event counters to subject to specified command operations indicated by bits 2:0. Bit 16 corresponds to MSR_EMON_L3_CTR_CTL0, bit 23 corresponds to MSR_EMON_L3_CTR_CTL7.
- GL_event_status (bit 55:48): Indicates the overflow status of each event counters. Bit 48 corresponds to MSR_EMON_L3_CTR_CTL0, bit 55 corresponds to MSR_EMON_L3_CTR_CTL7.

In the event control field (bits 63:32) of each MSR, if the saturate control (bit 59, see Figure 18-42 for example) is set, the event logic forces the value FFFF_FFFFH into the event count field instead of incrementing it.

18.23 PERFORMANCE MONITORING (P6 FAMILY PROCESSOR)

The P6 family processors provide two 40-bit performance counters, allowing two types of events to be monitored simultaneously. These can either count events or measure duration. When counting events, a counter increments each time a specified event takes place or a specified number of events takes place. When measuring duration, it counts the number of processor clocks that occur while a specified condition is true. The counters can count events or measure durations that occur at any privilege level.

Table A-16, Appendix A, lists the events that can be counted with the P6 family performance monitoring counters.

NOTE

> The performance-monitoring events listed in Appendix A are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

The performance-monitoring counters are supported by four MSRs: the performance event select MSRs (PerfEvtSel0 and PerfEvtSel1) and the performance counter MSRs (PerfCtr0 and PerfCtr1). These registers can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0. The PerfCtr0 and PerfCtr1 MSRs can be read from any privilege level using the RDPMC (read performance-monitoring counters) instruction.

NOTE

> The PerfEvtSel0, PerfEvtSel1, PerfCtr0, and PerfCtr1 MSRs and the events listed in Table A-16 are model-specific for P6 family processors. They are not guaranteed to be available in other IA-32 processors.

18.23.1 PerfEvtSel0 and PerfEvtSel1 MSRs

The PerfEvtSel0 and PerfEvtSel1 MSRs control the operation of the performance-monitoring counters, with one register used to set up each counter. They specify the events to be counted, how they should be counted, and the privilege levels at which counting should take place. Figure 18-45 shows the flags and fields in these MSRs.

The functions of the flags and fields in the PerfEvtSel0 and PerfEvtSel1 MSRs are as follows:

- **Event select field (bits 0 through 7)** — Selects the event logic unit to detect certain microarchitectural conditions (see Table A-16, for a list of events and their 8-bit codes).
- **Unit mask (UMASK) field (bits 8 through 15)** — Further qualifies the event logic unit selected in the event select field to detect a specific microarchitectural condition. For example, for some cache events, the mask is used as a MESI-protocol qualifier of cache states (see Table A-16).

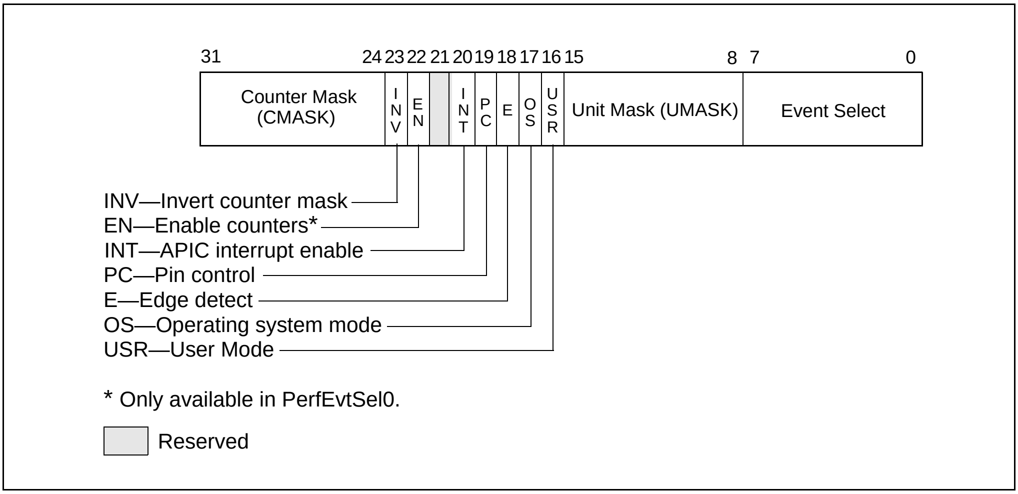

Figure 18-45. PerfEvtSel0 and PerfEvtSel1 MSRs

- **USR (user mode) flag (bit 16)** — Specifies that events are counted only when the processor is operating at privilege levels 1, 2 or 3. This flag can be used in conjunction with the OS flag.
- **OS (operating system mode) flag (bit 17)** — Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USR flag.
- **E (edge detect) flag (bit 18)** — Enables (when set) edge detection of events. The processor counts the number of deasserted to asserted transitions of any condition that can be expressed by the other fields. The mechanism is limited in that it does not permit back-to-back assertions to be distinguished. This mechanism allows software to measure not only the fraction of time spent in a particular state, but also the average length of time spent in such a state (for example, the time spent waiting for an interrupt to be serviced).
- **PC (pin control) flag (bit 19)** — When set, the processor toggles the PM*i* pins and increments the counter when performance-monitoring events occur; when clear, the processor toggles the PM*i* pins when the counter overflows. The toggling of a pin is defined as assertion of the pin for a single bus clock followed by deassertion.
- **INT (APIC interrupt enable) flag (bit 20)** — When set, the processor generates an exception through its local APIC on counter overflow.
- **EN (Enable Counters) Flag (bit 22)** — This flag is only present in the PerfEvtSel0 MSR. When set, performance counting is enabled in both performance-monitoring counters; when clear, both counters are disabled.
- **INV (invert) flag (bit 23)** — Inverts the result of the counter-mask comparison when set, so that both greater than and less than comparisons can be made.

- **Counter mask (CMASK) field (bits 24 through 31) —** When nonzero, the processor compares this mask to the number of events counted during a single cycle. If the event count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented. This mask can be used to count events only if multiple occurrences happen per clock (for example, two or more instructions retired per clock). If the counter-mask field is 0, then the counter is incremented each cycle by the number of events that occurred that cycle.

18.23.2 PerfCtr0 and PerfCtr1 MSRs

The performance-counter MSRs (PerfCtr0 and PerfCtr1) contain the event or duration counts for the selected events being counted. The RDPMC instruction can be used by programs or procedures running at any privilege level and in virtual-8086 mode to read these counters. The PCE flag in control register CR4 (bit 8) allows the use of this instruction to be restricted to only programs and procedures running at privilege level 0.

The RDPMC instruction is not serializing or ordered with other instructions. Thus, it does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDPMC instruction operation is performed.

Only the operating system, executing at privilege level 0, can directly manipulate the performance counters, using the RDMSR and WRMSR instructions. A secure operating system would clear the PCE flag during system initialization to disable direct user access to the performance-monitoring counters, but provide a user-accessible programming interface that emulates the RDPMC instruction.

The WRMSR instruction cannot arbitrarily write to the performance-monitoring counter MSRs (PerfCtr0 and PerfCtr1). Instead, the lower-order 32 bits of each MSR may be written with any value, and the high-order 8 bits are sign-extended according to the value of bit 31. This operation allows writing both positive and negative values to the performance counters.

18.23.3 Starting and Stopping the Performance-Monitoring Counters

The performance-monitoring counters are started by writing valid setup information in the PerfEvtSel0 and/or PerfEvtSel1 MSRs and setting the enable counters flag in the PerfEvtSel0 MSR. If the setup is valid, the counters begin counting following the execution of a WRMSR instruction that sets the enable counter flag. The counters can be stopped by clearing the enable counters flag or by clearing all the bits in the PerfEvtSel0 and PerfEvtSel1 MSRs. Counter 1 alone can be stopped by clearing the PerfEvtSel1 MSR.

18.23.4 Event and Time-Stamp Monitoring Software

To use the performance-monitoring counters and time-stamp counter, the operating system needs to provide an event-monitoring device driver. This driver should include procedures for handling the following operations:

- Feature checking
- Initialize and start counters
- Stop counters
- Read the event counters
- Read the time-stamp counter

The event monitor feature determination procedure must check whether the current processor supports the performance-monitoring counters and time-stamp counter. This procedure compares the family and model of the processor returned by the CPUID instruction with those of processors known to support performance monitoring. (The Pentium and P6 family processors support performance counters.) The procedure also checks the MSR and TSC flags returned to register EDX by the CPUID instruction to determine if the MSRs and the RDTSC instruction are supported.

The initialize and start counters procedure sets the PerfEvtSel0 and/or PerfEvtSel1 MSRs for the events to be counted and the method used to count them and initializes the counter MSRs (PerfCtr0 and PerfCtr1) to starting counts. The stop counters procedure stops the performance counters (see Section 18.23.3, “Starting and Stopping the Performance-Monitoring Counters”).

The read counters procedure reads the values in the PerfCtr0 and PerfCtr1 MSRs, and a read time-stamp counter procedure reads the time-stamp counter. These procedures would be provided in lieu of enabling the RDTSC and RDPMC instructions that allow application code to read the counters.

18.23.5 Monitoring Counter Overflow

The P6 family processors provide the option of generating a local APIC interrupt when a performance-monitoring counter overflows. This mechanism is enabled by setting the interrupt enable flag in either the PerfEvtSel0 or the PerfEvtSel1 MSR. The primary use of this option is for statistical performance sampling.

To use this option, the operating system should do the following things on the processor for which performance events are required to be monitored:

- Provide an interrupt vector for handling the counter-overflow interrupt.
- Initialize the APIC PERF local vector entry to enable handling of performance-monitor counter overflow events.
- Provide an entry in the IDT that points to a stub exception handler that returns without executing any instructions.
- Provide an event monitor driver that provides the actual interrupt handler and modifies the reserved IDT entry to point to its interrupt routine.

When interrupted by a counter overflow, the interrupt handler needs to perform the following actions:

- Save the instruction pointer (EIP register), code-segment selector, TSS segment selector, counter values and other relevant information at the time of the interrupt.
- Reset the counter to its initial setting and return from the interrupt.

An event monitor application utility or another application program can read the information collected for analysis of the performance of the profiled application.

18.24 PERFORMANCE MONITORING (PENTIUM PROCESSORS)

The Pentium processor provides two 40-bit performance counters, which can be used to count events or measure duration. The counters are supported by three MSRs: the control and event select MSR (CESR) and the performance counter MSRs (CTR0 and CTR1). These can be read from and written to using the RDMSR and WRMSR instructions, respectively. They can be accessed using these instructions only when operating at privilege level 0.

Each counter has an associated external pin (PM0/BP0 and PM1/BP1), which can be used to indicate the state of the counter to external hardware.

NOTES

The CESR, CTR0, and CTR1 MSRs and the events listed in Table A-17 are model-specific for the Pentium processor.

The performance-monitoring events listed in Appendix A are intended to be used as guides for performance tuning. Counter values reported are not guaranteed to be accurate and should be used as a relative guide for tuning. Known discrepancies are documented where applicable.

18.24.1 Control and Event Select Register (CESR)

The 32-bit control and event select MSR (CESR) controls the operation of performance-monitoring counters CTR0 and CTR1 and the associated pins (see Figure 18-46). To control each counter, the CESR register contains a 6-bit event select field (ES0 and ES1), a pin control flag (PC0 and PC1), and a 3-bit counter control field (CC0 and CC1). The functions of these fields are as follows:

- **ES0 and ES1 (event select) fields (bits 0-5, bits 16-21) —** Selects (by entering an event code in the field) up to two events to be monitored. See Table A-17 for a list of available event codes.

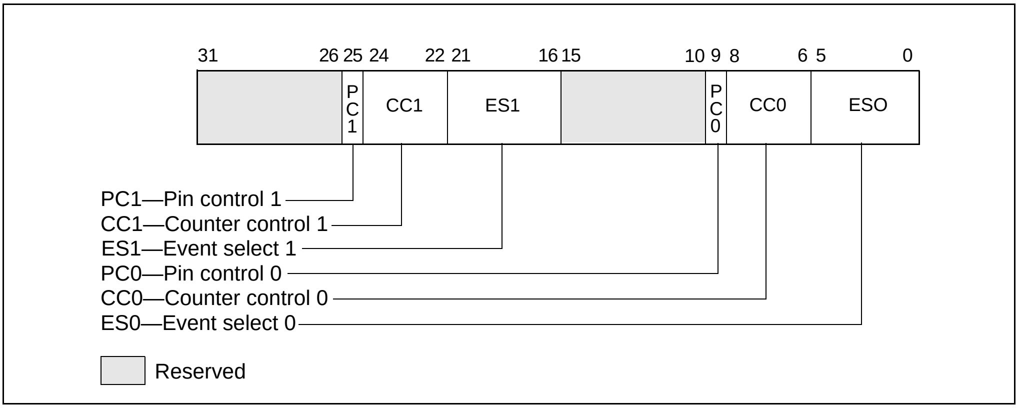

Figure 18-46. CESR MSR (Pentium Processor Only)

- **CC0 and CC1 (counter control) fields (bits 6-8, bits 22-24) —** Controls the operation of the counter. Control codes are as follows:

  000 — Count nothing (counter disabled)

  001 — Count the selected event while CPL is 0, 1, or 2

  010 — Count the selected event while CPL is 3

  011 — Count the selected event regardless of CPL

  100 — Count nothing (counter disabled)

  101 — Count clocks (duration) while CPL is 0, 1, or 2

  110 — Count clocks (duration) while CPL is 3

  111 — Count clocks (duration) regardless of CPL

  The highest order bit selects between counting events and counting clocks (duration); the middle bit enables counting when the CPL is 3; and the low-order bit enables counting when the CPL is 0, 1, or 2.

- **PC0 and PC1 (pin control) flags (bits 9, 25) —** Selects the function of the external performance-monitoring counter pin (PM0/BP0 and PM1/BP1). Setting one of these flags to 1 causes the processor to assert its associated pin when the counter has overflowed; setting the flag to 0 causes the pin to be asserted when the counter has been incremented. These flags permit the pins to be individually programmed to indicate the overflow or incremented condition. The external signalling of the event on the pins will lag the internal event by a few clocks as the signals are latched and buffered.

While a counter need not be stopped to sample its contents, it must be stopped and cleared or preset before switching to a new event. It is not possible to set one counter separately. If only one event needs to be changed, the CESR register must be read, the appropriate bits modified, and all bits must then be written back to CESR. At reset, all bits in the CESR register are cleared.

18.24.2 Use of the Performance-Monitoring Pins

When performance-monitor pins PM0/BP0 and/or PM1/BP1 are configured to indicate when the performance-monitor counter has incremented and an “occurrence event” is being counted, the associated pin is asserted (high) each time the event occurs. When a “duration event” is being counted, the associated PM pin is asserted for the entire duration of the event. When the performance-monitor pins are configured to indicate when the counter has overflowed, the associated PM pin is asserted when the counter has overflowed.

When the PM0/BP0 and/or PM1/BP1 pins are configured to signal that a counter has incremented, it should be noted that although the counters may increment by 1 or 2 in a single clock, the pins can only indicate that the event occurred. Moreover, since the internal clock frequency may be higher than the external clock frequency, a single external clock may correspond to multiple internal clocks.

A “count up to” function may be provided when the event pin is programmed to signal an overflow of the counter. Because the counters are 40 bits, a carry out of bit 39 indicates an overflow. A counter may be preset to a specific value less then $2^{40} - 1$. After the counter has been enabled and the prescribed number of events has transpired, the counter will overflow.

Approximately 5 clocks later, the overflow is indicated externally and appropriate action, such as signaling an interrupt, may then be taken.

The PM0/BP0 and PM1/BP1 pins also serve to indicate breakpoint matches during in-circuit emulation, during which time the counter increment or overflow function of these pins is not available. After RESET, the PM0/BP0 and PM1/BP1 pins are configured for performance monitoring, however a hardware debugger may reconfigure these pins to indicate breakpoint matches.

18.24.3 Events Counted

Events that performance-monitoring counters can be set to count and record (using CTR0 and CTR1) are divided in two categories: occurrence and duration:

- **Occurrence events** — Counts are incremented each time an event takes place. If PM0/BP0 or PM1/BP1 pins are used to indicate when a counter increments, the pins are asserted each clock counters increment. But if an event happens twice in one clock, the counter increments by 2 (the pins are asserted only once).
- **Duration events** — Counters increment the total number of clocks that the condition is true. When used to indicate when counters increment, PM0/BP0 and/or PM1/BP1 pins are asserted for the duration.

CHAPTER 19
INTRODUCTION TO VIRTUAL-MACHINE EXTENSIONS

19.1 OVERVIEW

This chapter describes the basics of virtual machine architecture and an overview of the virtual-machine extensions (VMX) that support virtualization of processor hardware for multiple software environments.

Information about VMX instructions is provided in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*. Other aspects of VMX and system programming considerations are described in chapters of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.

19.2 VIRTUAL MACHINE ARCHITECTURE

Virtual-machine extensions define processor-level support for virtual machines on IA-32 processors. Two principal classes of software are supported:

- **Virtual-machine monitors (VMM)** — A VMM acts as a host and has full control of the processor(s) and other platform hardware. A VMM presents guest software (see next paragraph) with an abstraction of a virtual processor and allows it to execute directly on a logical processor. A VMM is able to retain selective control of processor resources, physical memory, interrupt management, and I/O.
- **Guest software** — Each virtual machine (VM) is a guest software environment that supports a stack consisting of operating system (OS) and application software. Each operates independently of other virtual machines and uses on the same interface to processor(s), memory, storage, graphics, and I/O provided by a physical platform. The software stack acts as if it were running on a platform with no VMM. Software executing in a virtual machine must operate with reduced privilege so that the VMM can retain control of platform resources.

19.3 INTRODUCTION TO VMX OPERATION

Processor support for virtualization is provided by a form of processor operation called VMX operation. There are two kinds of VMX operation: VMX root operation and VMX non-root operation. In general, a VMM will run in VMX root operation and guest software will run in VMX non-root operation. Transitions between VMX root operation and VMX non-root operation are called VMX transitions. There are two kinds of VMX transitions. Transitions into VMX non-root operation are called VM entries. Transitions from VMX non-root operation to VMX root operation are called VM exits.

Processor behavior in VMX root operation is very much as it is outside VMX operation. The principal differences are that a set of new instructions (the VMX instructions) is available and that the values that can be loaded into certain control registers are limited (see Section 19.8).

Processor behavior in VMX non-root operation is restricted and modified to facilitate virtualization. Instead of their ordinary operation, certain instructions (including the new VMCALL instruction) and events cause VM exits to the VMM. Because these VM exits replace ordinary behavior, the functionality of software in VMX non-root operation is limited. It is this limitation that allows the VMM to retain control of processor resources.

There is no software-visible bit whose setting indicates whether a logical processor is in VMX non-root operation. This fact may allow a VMM to prevent guest software from determining that it is running in a virtual machine.

Because VMX operation places restrictions even on software running with current privilege level (CPL) 0, guest software can run at the privilege level for which it was originally designed. This capability may simplify the development of a VMM.

19.4 LIFE CYCLE OF VMM SOFTWARE

Figure 19-1 illustrates the life cycle of a VMM and its guest software as well as the interactions between them. The following items summarize that life cycle:

- Software enters VMX operation by executing a VMXON instruction.
- Using VM entries, a VMM can then enter guests into virtual machines (one at a time). The VMM effects a VM entry using instructions VMLAUNCH and VMRESUME; it regains control using VM exits.
- VM exits transfer control to an entry point specified by the VMM. The VMM can take action appropriate to the cause of the VM exit and can then return to the virtual machine using a VM entry.
- Eventually, the VMM may decide to shut itself down and leave VMX operation. It does so by executing the VMXOFF instruction.

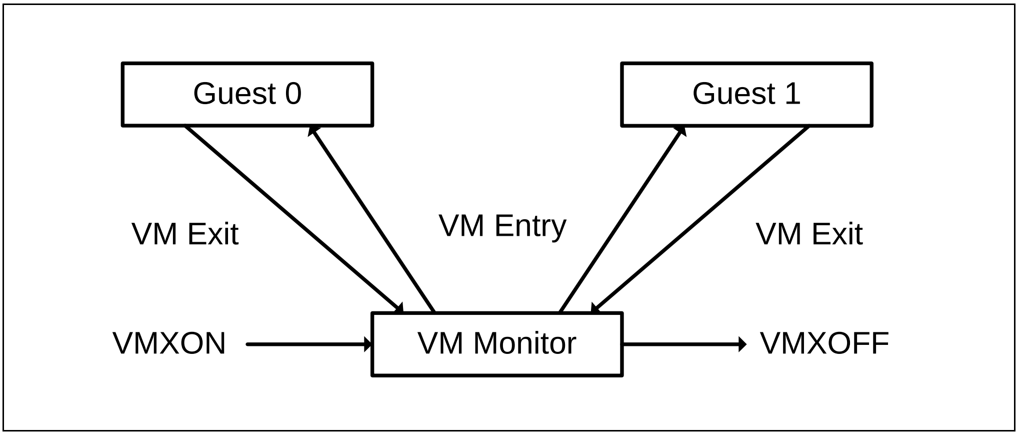

Figure 19-1. Interaction of a Virtual-Machine Monitor and Guests

19.5 VIRTUAL-MACHINE CONTROL STRUCTURE

VMX non-root operation and VMX transitions are controlled by a data structure called a virtual-machine control structure (VMCS).

Access to the VMCS is managed through a component of processor state called the VMCS pointer (one per logical processor). The value of the VMCS pointer is the 64-bit address of the VMCS. The VMCS pointer is read and written using the instructions VMPTRST and VMPTRLD. The VMM configures a VMCS using the VMREAD, VMWRITE, and VMCLEAR instructions.

A VMM could use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the VMM could use a different VMCS for each virtual processor.

19.6 DISCOVERING SUPPORT FOR VMX

Before system software enters into VMX operation, it must discover the presence of VMX support in the processor. System software can determine whether a processor supports VMX operation using CPUID. If CPUID.1:ECX.VMX[bit 5] = 1, then VMX operation is supported. See Chapter 3, “Instruction Set Reference, A-M” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*.

The VMX architecture is designed to be extensible so that future processors in VMX operation can support additional features not present in first-generation implementations of the VMX architecture. The availability of extensible VMX features is reported to software using a set of VMX capability MSRs (see Appendix G, “VMX Capability Reporting Facility”).

19.7 ENABLING AND ENTERING VMX OPERATION

Before system software can enter VMX operation, it enables VMX by setting CR4.VMXE[bit 13] = 1. VMX operation is then entered by executing the VMXON instruction. VMXON causes an invalid-opcode exception (#UD) if executed with CR4.VMXE = 0. Once in VMX operation, it is not possible to clear CR4.VMXE (see Section 19.8). System software leaves VMX operation by executing the VMXOFF instruction. CR4.VMXE can be cleared outside of VMX operation after executing of VMXOFF.

VMXON is also controlled by the IA32_FEATURE_CONTROL MSR (MSR address 3AH). This MSR is cleared to zero when a logical processor is reset. The relevant bits of the MSR are:

- **Bit 0 is the lock bit.** If this bit is clear, VMXON causes a general-protection exception. If the lock bit is set, WRMSR to this MSR causes a general-protection exception; the MSR cannot be modified until a power-up reset condition. System BIOS can use this bit to provide a setup option for BIOS to disable support for VMX. To enable VMX support in a platform, BIOS must set bit 1, bit 2, or both (see below), as well as the lock bit.
- **Bit 1 enables VMXON in SMX operation.** If this bit is clear, execution of VMXON in SMX operation causes a general-protection exception. Attempts to set this bit on logical processors that do not support both VMX operation (see Section 19.6) and SMX operation (see Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*) cause general-protection exceptions.
- **Bit 2 enables VMXON outside SMX operation.** If this bit is clear, execution of VMXON outside SMX operation causes a general-protection exception. Attempts to set this bit on logical processors that do not support VMX operation (see Section 19.6) cause general-protection exceptions.

NOTE

A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

Before executing VMXON, software should allocate a naturally aligned 4-KByte region of memory that a logical processor may use to support VMX operation.[1] This region is called the **VMXON region**. The address of the VMXON region (the VMXON pointer)

[1] Future processors may require that a different amount of memory be reserved. If so, this fact is reported to software using the VMX capability-reporting mechanism.

is provided in an operand to VMXON. Section 20.10.4, “VMXON Region,” details how software should initialize and access the VMXON region.

19.8 RESTRICTIONS ON VMX OPERATION

VMX operation places restrictions on processor operation. These are detailed below:

- In VMX operation, processors may fix certain bits in CR0 and CR4 to specific values and not support other values. VMXON fails if any of these bits contains an unsupported value (see “VMXON—Enter VMX Operation” in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*). Any attempt to set one of these bits to an unsupported value while in VMX operation (including VMX root operation) using any of the CLTS, LMSW, or MOV CR instructions causes a general-protection exception. VM entry or VM exit cannot set any of these bits to an unsupported value.[2]

NOTE

> The first processors to support VMX operation require that the following bits be 1 in VMX operation: CR0.PE, CR0.NE, CR0.PG, and CR4.VMXE. The restrictions on CR0.PE and CR0.PG imply that VMX operation is supported only in paged protected mode (including IA-32e mode). Therefore, guest software cannot be run in unpaged protected mode or in real-address mode. See Section 27.2, “Supporting Processor Operating Modes in Guest Environments,” for a discussion of how a VMM might support guest software that expects to run in unpaged protected mode or in real-address mode.

- VMXON fails if a logical processor is in A20M mode (see “VMXON—Enter VMX Operation” in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*). Once the processor is in VMX operation, A20M interrupts are blocked. Thus, it is impossible to be in A20M mode in VMX operation.
- The INIT signal is blocked whenever a logical processor is in VMX root operation. It is not blocked in VMX non-root operation. Instead, INITs cause VM exits (see Section 21.3, “Other Causes of VM Exits”).

[2] Software should consult the VMX capability MSRs IA32_VMX_CR0_FIXED0 and IA32_VMX_CR0_FIXED1 to determine how bits in CR0 are set. (see Appendix G.7). For CR4, software should consult the VMX capability MSRs IA32_VMX_CR4_FIXED0 and IA32_VMX_CR4_FIXED1 (see Appendix G.8).

CHAPTER 20
VIRTUAL-MACHINE CONTROL STRUCTURES

20.1 OVERVIEW

The virtual-machine control data structure (VMCS) is defined for VMX operation. A VMCS manages transitions in and out of VMX non-root operation (VM entries and VM exits) as well as processor behavior in VMX non-root operation. This structure is manipulated by the new instructions VMCLEAR, VMPTRLD, VMREAD, and VMWRITE.

A VMM can use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the VMM can use a different VMCS for each virtual processor.

Each logical processor associates a region in memory with each VMCS. This region is called the **VMCS region**.[1] Software references a specific VMCS by using the 64-bit physical address of the region; such an address is called a **VMCS pointer**. VMCS pointers must be aligned on a 4-KByte boundary (bits 11:0 must be zero). On processors that support Intel 64 architecture, these pointers must not set bits beyond the processor's physical-address width.[2] On processors that do not support Intel 64 architecture, they must not set any bits in the range 63:32.

A logical processor may maintain any number of active VMCSs. At any given time, one is the current VMCS:

- Software makes a VMCS **active** by executing VMPTRLD with the address of the VMCS. The processor may optimize VMX operation by maintaining the state of an active VMCS in memory, on the processor, or both. Software should not make a VMCS active on more than one logical processor (see Section 20.10.1 for how to migrate a VMCS from one logical processor to another). Software makes a VMCS inactive by executing VMCLEAR with the address of the VMCS. A logical processor does not use an inactive VMCS or maintain its state on the processor.

  If VMXOFF is executed while a VMCS is active, the VMCS data in the corresponding VMCS region are undefined after execution of VMXOFF. Software can avoid this problem by avoiding execution of VMXOFF while a VMCS is active.

- Software makes a VMCS **current** by executing VMPTRLD with the address of the VMCS; that address is loaded into the **current-VMCS pointer**. VMX instructions VMLAUNCH, VMPTRST, VMREAD, VMRESUME, and VMWRITE operate on the current VMCS. In particular, the VMPTRST instruction stores the current-VMCS

[1] The amount of memory required for a VMCS region is at most 4 KBytes. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix G.1).

[2] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

pointer into a specified memory location (it stores the value FFFFFFFF_FFFFFFFFH if there is no current VMCS). A VMCS remains current until either software executes VMPTRLD with the address of a different VMCS (which then becomes the current VMCS) or software executes VMCLEAR with the address of the current VMCS (after which there is no current VMCS).

This document frequently uses the term “the VMCS” to refer to the current VMCS.

20.2 FORMAT OF THE VMCS REGION

A VMCS region comprises up to 4-KBytes.[1] The format of a VMCS region is given in Table 20-1.

Table 20-1. Format of the VMCS Region

Byte Offset	Contents
0	VMCS revision identifier
4	VMX-abort indicator
8	VMCS data (implementation-specific format)

The first 32 bits of the VMCS region contain the **VMCS revision identifier**. Processors that maintain VMCS data in different formats (see below) use different VMCS revision identifiers. These identifiers enable software to avoid using a VMCS region formatted for one processor on a processor that uses a different format.[2]

Software should write the VMCS revision identifier to the VMCS region before using that region for a VMCS. The VMCS revision identifier is never written by the processor; VMPTRLD may fail if its operand references a VMCS region whose VMCS revision identifier differs from that used by the processor. Software can discover the VMCS revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_BASIC (see Appendix G, “VMX Capability Reporting Facility”).

The next 32 bits of the VMCS region are used for the **VMX-abort indicator**. The contents of these bits do not control processor operation in any way. A logical processor writes a non-zero value into these bits if a VMX abort occurs (see Section 23.7). Software may also write into this field.

The remainder of the VMCS region is used for **VMCS data** (those parts of the VMCS that control VMX non-root operation and the VMX transitions). The format of these data is implementation-specific. VMCS data are discussed in Section 20.3 through

[1] The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix G.1).

[2] Logical processors that use the same VMCS revision identifier use the same size for VMCS regions.

Section 20.9. To ensure proper behavior in VMX operation, software should maintain the VMCS region and related structures (enumerated in Section 20.10.3) in writeback cacheable memory. Future implementations may allow or require a different memory type[1]. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix G.1).

20.3 ORGANIZATION OF VMCS DATA

The VMCS data are organized into six logical groups:

- **Guest-state area.** Processor state is saved into the guest-state area on VM exits and loaded from there on VM entries.
- **Host-state area.** Processor state is loaded from the host-state area on VM exits.
- **VM-execution control fields.** These fields control processor behavior in VMX non-root operation. They determine in part the causes of VM exits.
- **VM-exit control fields.** These fields control VM exits.
- **VM-entry control fields.** These fields control VM entries.
- **VM-exit information fields.** These fields receive information on VM exits and describe the cause and the nature of VM exits. They are read-only.

The VM-execution control fields, the VM-exit control fields, and the VM-entry control fields are sometimes referred to collectively as VMX controls.

20.4 GUEST-STATE AREA

This section describes fields contained in the guest-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM entry (see Section 22.3.2) and stored into these fields on every VM exit (see Section 23.3).

20.4.1 Guest Register State

The following fields in the guest-state area correspond to processor registers:

- Control registers CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Debug register DR7 (64 bits; 32 bits on processors that do not support Intel 64 architecture).

[1] Alternatively, software may map any of these regions or structures with the UC memory type. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix G.1.

- RSP, RIP, and RFLAGS (64 bits each; 32 bits on processors that do not support Intel 64 architecture).[1]
- The following fields for each of the registers CS, SS, DS, ES, FS, GS, LDTR, and TR:
  - Selector (16 bits).
  - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture). The base-address fields for CS, SS, DS, and ES have only 32 architecturally-defined bits; nevertheless, the corresponding VMCS fields have 64 bits on processors that support Intel 64 architecture.
  - Segment limit (32 bits). The limit field is always a measure in bytes.
  - Access rights (32 bits). The format of this field is given in Table 20-2 and detailed as follows:
    - The low 16 bits correspond to bits 23:8 of the upper 32 bits of a 64-bit segment descriptor. While bits 19:16 of code-segment and data-segment descriptors correspond to the upper 4 bits of the segment limit, the corresponding bits (bits 11:8) are reserved in this VMCS field.
    - Bit 16 indicates an **unusable segment**. Attempts to use such a segment fault except in 64-bit mode. In general, a segment register is unusable if it has been loaded with a null selector.[2]
    - Bits 31:17 are reserved.

Table 20-2. Format of Access Rights

Bit Position(s)	Field
3:0	Segment type
4	S — Descriptor type (0 = system; 1 = code or data)
6:5	DPL — Descriptor privilege level

[1] This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

[2] There are a few exceptions to this statement. For example, a segment with a non-null selector may be unusable following a task switch that fails after its commit point; see “Interrupt 10—Invalid TSS Exception (#TS)” in Section 5.14, “Exception and Interrupt Handling in 64-bit Mode,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*. In contrast, the TR register is usable after processor reset despite having a null selector; see Table 9-1 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

Table 20-2. Format of Access Rights (Contd.)

Bit Position(s)	Field
7	P — Segment present
11:8	Reserved
12	AVL — Available for use by system software
13	Reserved (except for CS) L — 64-bit mode active (for CS only)
14	D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
15	G — Granularity
16	Segment unusable (0 = usable; 1 = unusable)
31:17	Reserved

The base address, segment limit, and access rights compose the “hidden” part (or “descriptor cache”) of each segment register. These data are included in the VMCS because it is possible for a segment register’s descriptor cache to be inconsistent with the segment descriptor in memory (in the GDT or the LDT) referenced by the segment register’s selector.

Note that the value of the DPL field for SS is always equal to the logical processor’s current privilege level (CPL).[1]

- The following fields for each of the registers GDTR and IDTR:
  - — Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture).
  - — Limit (32 bits). The limit fields contain 32 bits even though these fields are specified as only 16 bits in the architecture.
- The following MSRs:
  - — IA32_DEBUGCTL (64 bits)
  - — IA32_SYSENTER_CS (32 bits)
  - — IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture)
  - — IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on logical processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-entry control.

[1] In protected mode, CPL is also associated with the RPL field in the CS selector. However, the RPL fields are not meaningful in real-address mode or in virtual-8086 mode.

  - — IA32_PAT (64 bits). This field is supported only on logical processors that support either the 1-setting of the "load IA32_PAT" VM-entry control or that of the "save IA32_PAT" VM-exit control.
  - — IA32_EFER (64 bits). This field is supported only on logical processors that support either the 1-setting of the "load IA32_EFER" VM-entry control or that of the "save IA32_EFER" VM-exit control.

- The register SMBASE (32 bits). This register contains the base address of the logical processor's SMRAM image.

20.4.2 Guest Non-Register State

In addition to the register state described in Section 20.4.1, the guest-state area includes the following fields that characterize guest state but which do not correspond to processor registers:

- **Activity state** (32 bits). This field identifies the logical processor's activity state. When a logical processor is executing instructions normally, it is in the **active state**. Execution of certain instructions and the occurrence of certain events may cause a logical processor to transition to an **inactive state** in which it ceases to execute instructions.

  The following activity states are defined:[1]

  - — 0: **Active**. The logical processor is executing instructions normally.
  - — 1: **HLT**. The logical processor is inactive because it executed the HLT instruction.
  - — 2: **Shutdown**. The logical processor is inactive because it incurred a **triple fault**[2] or some other serious error.
  - — 3: **Wait-for-SIPI**. The logical processor is inactive because it is waiting for a startup-IPI (SIPI).

  Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix G.6) to determine what activity states are supported.

- **Interruptibility state** (32 bits). The IA-32 architecture includes features that permit certain events to be blocked for a period of time. This field contains information about such blocking. Details and the format of this field are given in Table 20-3.

- **Pending debug exceptions** (64 bits; 32 bits on processors that do not support Intel 64 architecture). IA-32 processors may recognize one or more debug

[1] Execution of the MWAIT instruction may put a logical processor into an inactive state. However, this VMCS field never reflects this state. See Section 23.1.

[2] A triple fault occurs when a logical processor encounters an exception while attempting to deliver a double fault.

Table 20-3. Format of Interruptibility State

Bit Position(s)	Bit Name	Notes
0	Blocking by STI	See the "STI—Set Interrupt Flag" section in Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*. Execution of STI with RFLAGS.IF = 0 blocks interrupts (and, optionally, other events) for one instruction after its execution. Setting this bit indicates that this blocking is in effect.
1	Blocking by MOV SS	See the "MOV—Move a Value from the Stack" and "POP—Pop a Value from the Stack" sections in Chapter 3 and Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A & 2B*, and Section 5.8.3 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. Execution of a MOV to SS or a POP to SS blocks interrupts for one instruction after its execution. In addition, certain debug exceptions are inhibited between a MOV to SS or a POP to SS and a subsequent instruction. Setting this bit indicates that the blocking of all these events is in effect. This document uses the term "blocking by MOV SS," but it applies equally to POP SS.
2	Blocking by SMI	See Section 26.2. System-management interrupts (SMIs) are disabled while the processor is in system-management mode (SMM). Setting this bit indicates that blocking of SMIs is in effect.
3	Blocking by NMI	See Section 5.7.1 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* and Section 26.8. Delivery of a non-maskable interrupt (NMI) or a system-management interrupt (SMI) blocks subsequent NMIs until the next execution of IRET. See Section 21.4 for how this behavior of IRET may change in VMX non-root operation. Setting this bit indicates that blocking of NMIs is in effect. Clearing this bit does not imply that NMIs are not (temporarily) blocked for other reasons. If the "virtual NMIs" VM-execution control (see Section 20.6.1) is 1, this bit does not control the blocking of NMIs. Instead, it refers to "virtual-NMI blocking" (the fact that guest software is not ready for an NMI).
31:4	Reserved	VM entry will fail if these bits are not 0. See Section 22.3.1.5.

exceptions without immediately delivering them.[1] This field contains information about such exceptions. This field is described in Table 20-4.

Table 20-4. Format of Pending-Debug-Exceptions

Bit Position(s)	Bit Name	Notes
3:0	B3 – B0	When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if the corresponding enabling bit in DR7 is not set.
11:4	Reserved	VM entry fails if these bits are not 0. See Section 22.3.1.5.
12	Enabled breakpoint	When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7.
13	Reserved	VM entry fails if this bit is not 0. See Section 22.3.1.5.
14	BS	When set, this bit indicates that a debug exception would have been triggered by single-step execution mode.
63:15	Reserved	VM entry fails if these bits are not 0. See Section 22.3.1.5. Bits 63:32 exist only on processors that support Intel 64 architecture.

- **VMCS link pointer** (64 bits). This field is included for future expansion. Software should set this field to FFFFFFFF_FFFFFFFFH to avoid VM-entry failures (see Section 22.3.1.5).
- **VMX-preemption timer value** (32 bits). This field is supported only on logical processors that support the 1-setting of the “activate VMX-preemption timer” VM-execution control. This field contains the value that the VMX-preemption timer will use following the next VM entry with that setting. See Section 21.7.1 and Section 22.6.4.
- **Page-directory-pointer-table entries** (PDPTEs; 64 bits each). These four (4) fields (PDPTE0, PDPTE1, PDPTE2, and PDPTE3) are supported only on logical processors that support the 1-setting of the “enable EPT” VM-execution control. They correspond to the PDPTEs referenced by CR3 when PAE paging is in use (see Section 3.8 in the *Intel® 64 and IA-32 Architectures Software Developer’s*

[1] For example, execution of a MOV to SS or a POP to SS may inhibit some debug exceptions for one instruction. See Section 5.8.3 of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*. In addition, certain events incident to an instruction (for example, an INIT signal) may take priority over debug traps generated by that instruction. See Table 5-2 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

Manual, Volume 3A). They are used only if the “enable EPT” VM-execution control is 1.

20.5 HOST-STATE AREA

This section describes fields contained in the host-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM exit (see Section 23.5).

All fields in the host-state area correspond to processor registers:

- CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- RSP and RIP (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Selector fields (16 bits each) for the segment registers CS, SS, DS, ES, FS, GS, and TR. There is no field in the host-state area for the LDTR selector.
- Base-address fields for FS, GS, TR, GDTR, and IDTR (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- The following MSRs:
  - IA32_SYSENTER_CS (32 bits)
  - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture).
  - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on logical processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-exit control.
  - IA32_PAT (64 bits). This field is supported only on logical processors that support either the 1-setting of the “load IA32_PAT” VM-exit control.
  - IA32_EFER (64 bits). This field is supported only on logical processors that support either the 1-setting of the “load IA32_EFER” VM-exit control.

In addition to the state identified here, some processor state components are loaded with fixed values on every VM exit; there are no fields corresponding to these components in the host-state area. See Section 23.5 for details of how state is loaded on VM exits.

20.6 VM-EXECUTION CONTROL FIELDS

The VM-execution control fields govern VMX non-root operation. These are described in Section 20.6.1 through Section 20.6.8.

20.6.1 Pin-Based VM-Execution Controls

The pin-based VM-execution controls constitute a 32-bit vector that governs the handling of asynchronous events (for example: interrupts).[1] Table 20-5 lists the controls supported. See Chapter 21 for how these controls affect processor behavior in VMX non-root operation.

Table 20-5. Definitions of Pin-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	External-interrupt exiting	If this control is 1, external interrupts cause VM exits. Otherwise, they are delivered normally through the guest interrupt-descriptor table (IDT). If this control is 1, the value of RFLAGS.IF does not affect interrupt blocking.
3	NMI exiting	If this control is 1, non-maskable interrupts (NMIs) cause VM exits. Otherwise, they are delivered normally using descriptor 2 of the IDT. This control also determines interactions between IRET and blocking by NMI (see Section 21.4).
5	Virtual NMIs	If this control is 1, NMIs are never blocked and the “blocking by NMI” bit (bit 3) in the interruptibility-state field indicates “virtual-NMI blocking” (see Table 20-3). This control also interacts with the “NMI-window exiting” VM-execution control (see Section 20.6.2). This control can be set only if the “NMI exiting” VM-execution control (above) is 1.
6	Activate VMX-preemption timer	If this control is 1, the VMX-preemption timer counts down in VMX non-root operation; see Section 21.7.1. A VM exit occurs when the timer counts down to zero; see Section 21.3.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_PINBASED_CTLS and IA32_VMX_TRUE_PINBASED_CTLS (see Appendix G.3.1) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 22.2).

Note that the first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 2, and 4. The VMX capability MSR IA32_VMX_PINBASED_CTLS will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PINBASED_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

[1] Some asynchronous events cause VM exits regardless of the settings of the pin-based VM-execution controls (see Section 21.3).

20.6.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls constitute two 32-bit vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions.[1] These are the **primary processor-based VM-execution controls** and the **secondary processor-based VM-execution controls**.

Table 20-6 lists the primary processor-based VM-execution controls. See Chapter 21 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 20-6. Definitions of Primary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
2	Interrupt-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if RFLAGS.IF = 1 and there are no other blocking of interrupts (see Section 20.4.2).
3	Use TSC offsetting	This control determines whether executions of RDTSC and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC offset field (see Section 20.6.5 and Section 21.4).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPMC exiting	This control determines whether executions of RDPMC cause VM exits.
12	RDTSC exiting	This control determines whether executions of RDTSC cause VM exits.
15	CR3-load exiting	In conjunction with the CR3-target controls (see Section 20.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 21.1.3. Note that the first processors to support the virtual-machine extensions supported only the 1-setting of this control.
16	CR3-store exiting	This control determines whether executions of MOV from CR3 cause VM exits. Note that the first processors to support the virtual-machine extensions supported only the 1-setting of this control.

[1] Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 21.1.2), as do task switches (see Section 21.3).

Table 20-6. Definitions of Primary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
19	CR8-load exiting	This control determines whether executions of MOV to CR8 cause VM exits. This control must be 0 on processors that do not support Intel 64 architecture.
20	CR8-store exiting	This control determines whether executions of MOV from CR8 cause VM exits. This control must be 0 on processors that do not support Intel 64 architecture.
21	Use TPR shadow	Setting this control to 1 activates the TPR shadow, which is maintained in a page of memory addressed by the virtual-APIC address. See Section 21.4. This control must be 0 on processors that do not support Intel 64 architecture.
22	NMI-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if there is no virtual-NMI blocking (see Section 20.4.2). This control can be set only if the “virtual NMIs” VM-execution control (see Section 20.6.1) is 1.
23	MOV-DR exiting	This control determines whether executions of MOV DR cause VM exits.
24	Unconditional I/O exiting	This control determines whether executions of I/O instructions (IN, INS/INSB/INSW/INSD, OUT, and OUTS/OUTSB/OUTSW/OUTSD) cause VM exits. This control is ignored if the “use I/O bitmaps” control is 1.
25	Use I/O bitmaps	This control determines whether I/O bitmaps are used to restrict executions of I/O instructions (see Section 20.6.4 and Section 21.1.3). For this control, “0” means “do not use I/O bitmaps” and “1” means “use I/O bitmaps.” If the I/O bitmaps are used, the setting of the “unconditional I/O exiting” control is ignored.
27	Monitor trap flag	If this control is 1, the monitor trap flag debugging feature is enabled. See Section 21.7.2.

Table 20-6. Definitions of Primary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
28	Use MSR bitmaps	This control determines whether MSR bitmaps are used to control execution of the RDMSR and WRMSR instructions (see Section 20.6.4 and Section 21.1.3). For this control, “0” means “do not use MSR bitmaps” and “1” means “use MSR bitmaps.” If the MSR bitmaps are not used, all executions of the RDMSR and WRMSR instructions cause VM exits. Not all processors support the 1-setting of this control. Software may consult the VMX capability MSR IA32_VMX_PROCBASED_CTLS (see Appendix G.3) to determine whether that setting is supported.
29	MONITOR exiting	This control determines whether executions of MONITOR cause VM exits.
30	PAUSE exiting	This control determines whether executions of PAUSE cause VM exits.
31	Activate secondary controls	This control determines whether the secondary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the secondary processor-based VM-execution controls were also 0.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_PROCBASED_CTLS and IA32_VMX_TRUE_PROCBASED_CTLS (see Appendix G.3.2) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 22.2).

Note that the first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 4–6, 8, 13–16, and 26. The VMX capability MSR IA32_VMX_PROCBASED_CTLS will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PROCBASED_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-execution controls determines whether the secondary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the secondary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 31 of the primary processor-based VM-execution controls do not support the secondary processor-based VM-execution controls.

Table 20-7 lists the secondary processor-based VM-execution controls. See Chapter 21 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 20-7. Definitions of Secondary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	Virtualize APIC accesses	If this control is 1, a VM exit occurs on any attempt to access data on the page with the APIC-access address. See Section 21.2.
1	Enable EPT	If this control is 1, extended page tables (EPT) are enabled. See Chapter 25.
2	Descriptor-table exiting	This control determines whether executions of LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, and STR cause VM exits.
4	Virtualize x2APIC mode	Setting this control to 1 causes RDMSR and WRMSR to MSR 808H to use the TPR shadow, which is maintained on the virtual-APIC page. See Section 21.4.
5	Enable VPID	If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Chapter 24.
6	WBINVD exiting	This control determines whether executions of WBINVD cause VM exits.

All other bits in these fields are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTLS2 (see Appendix G.3.3) to determine how to set reserved bits. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 22.2).

If a logical processor supports the 1-setting of bit 31 of the primary processor-based VM-execution controls but software has set that bit is 0, VM entry and VMX non-root operation function as if all the secondary processor-based VM-execution controls were 0. However, the logical processor will maintain the secondary processor-based VM-execution controls as written by VMWRITE.

20.6.3 Exception Bitmap

The **exception bitmap** is a 32-bit field that contains one bit for each exception. When an exception occurs, its vector is used to select a bit in this field. If the bit is 1, the exception causes a VM exit. If the bit is 0, the exception is delivered normally through the IDT, using the descriptor corresponding to the exception’s vector.

Whether a page fault (exception with vector 14) causes a VM exit is determined by bit 14 in the exception bitmap as well as the error code produced by the page fault and two 32-bit fields in the VMCS (the **page-fault error-code mask** and **page-fault error-code match)**. See Section 21.3 for details.

20.6.4 I/O-Bitmap Addresses

The VM-execution control fields include the 64-bit physical addresses of **I/O bitmaps** A and B (each of which are 4 KBytes in size). I/O bitmap A contains one bit for each I/O port in the range 0000H through 7FFFH; I/O bitmap B contains bits for ports in the range 8000H through FFFFH.

A logical processor uses these bitmaps if and only if the “use I/O bitmaps” control is 1. If the bitmaps are used, execution of an I/O instruction causes a VM exit if any bit in the I/O bitmaps corresponding to a port it accesses is 1. See Section 21.1.3 for details. If the bitmaps are used, their addresses must be 4-KByte aligned.

20.6.5 Time-Stamp Counter Offset

VM-execution control fields include a 64-bit **TSC-offset** field. If the “RDTSC exiting” control is 0 and the “use TSC offsetting” control is 1, this field controls executions of the RDTSC instruction and executions of the RDMSR instruction that read from the IA32_TIME_STAMP_COUNTER MSR. The signed value of the TSC offset is combined with the contents of the time-stamp counter (using signed addition) and the sum is reported to guest software in EDX:EAX. See Chapter 21 for a detailed treatment of the behavior of RDTSC and RDMSR in VMX non-root operation.

20.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4

VM-execution control fields include **guest/host masks** and **read shadows** for the CR0 and CR4 registers. These fields control executions of instructions that access those registers (including CLTS, LMSW, MOV CR, and SMSW). They are 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

In general, bits set to 1 in a guest/host mask correspond to bits “owned” by the host:

- Guest attempts to set them (using CLTS, LMSW, or MOV to CR) to values differing from the corresponding bits in the corresponding read shadow cause VM exits.
- Guest reads (using MOV from CR or SMSW) return values for these bits from the corresponding read shadow.

Bits cleared to 0 correspond to bits “owned” by the guest; guest attempts to modify them succeed and guest reads return values for these bits from the control register itself.

See Chapter 21 for details regarding how these fields affect VMX non-root operation.

20.6.7 CR3-Target Controls

The VM-execution control fields include a set of 4 **CR3-target values** and a **CR3-target count**. The CR3-target values each have 64 bits on processors that support

Intel 64 architecture and 32 bits on processors that do not. The CR3-target count has 32 bits on all processors.

An execution of MOV to CR3 in VMX non-root operation does not cause a VM exit if its source operand matches one of these values. If the CR3-target count is *n*, only the first *n* CR3-target values are considered; if the CR3-target count is 0, MOV to CR3 always causes a VM exit

There are no limitations on the values that can be written for the CR3-target values. VM entry fails (see Section 22.2) if the CR3-target count is greater than 4.

Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix G.6) to determine the number of values supported.

20.6.8 Controls for APIC Accesses

There are three mechanisms by which software accesses registers of the logical processor's local APIC:

- If the local APIC is in xAPIC mode, it can perform memory-mapped accesses to addresses in the 4-KByte page referenced by the physical address in the IA32_APIC_BASE MSR (see Section 9.4.4, "Local APIC Status and Location" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* and *Intel® 64 Architecture x2APIC Specification*).[1]
- If the local APIC is in x2APIC mode, it can accesses the local APIC's registers using the RDMSR and WRMSR instructions (see *Intel® 64 Architecture x2APIC Specification*).
- In 64-bit mode, it can access the local APIC's task-priority register (TPR) using the MOV CR8 instruction.

There are three processor-based VM-execution controls (see Section 20.6.2) that control such accesses. There are "use TPR shadow", "virtualize APIC accesses", and "virtualize x2APIC mode". These controls interact with the following fields:

- **APIC-access address** (64 bits). This field is the physical address of the 4-KByte **APIC-access page**. If the "virtualize APIC accesses" VM-execution control is 1, operations that access this page may cause VM exits. See Section 21.2 and Section 21.5.

  The APIC-access address exists only on processors that support the 1-setting of the "virtualize APIC accesses" VM-execution control.
- **Virtual-APIC address** (64 bits). This field is the physical address of the 4-KByte **virtual-APIC page**.

  If the "use TPR shadow" VM-execution control is 1, the virtual-APIC address must be 4-KByte aligned. The virtual-APIC page is accessed by the following operations if the "use TPR shadow" VM-execution control is 1:

[1] Note that, if the local APIC does not support x2APIC mode, it is always in xAPIC mode.

- — The MOV CR8 instructions (see Section 21.1.3 and Section 21.4).
- — Accesses to byte 80H on the APIC-access page if, in addition, the “virtualize APIC accesses” VM-execution control is 1 (see Section 21.5.3).
- — The RDMSR and WRMSR instructions if, in addition, the value of ECX is 808H (indicating the TPR MSR) and the “virtualize x2APIC mode” VM-execution control is 1 (see Section 21.4).

The virtual-APIC address exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.

- **TPR threshold** (32 bits). Bits 3:0 of this field determine the threshold below which the TPR shadow (bits 7:4 of byte 80H of the virtual-APIC page) cannot fall. A VM exit occurs after an operation (e.g., an execution of MOV to CR8) that reduces the TPR shadow below this value. See Section 21.4 and Section 21.5.3.

  The TPR threshold exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.

20.6.9 MSR-Bitmap Address

On processors that support the 1-setting of the “use MSR bitmaps” VM-execution control, the VM-execution control fields include the 64-bit physical address of four contiguous **MSR bitmaps**, which are each 1-KByte in size. This field does not exist on processors that do not support the 1-setting of that control. The four bitmaps are:

- **Read bitmap for low MSRs** (located at the MSR-bitmap address)**.** This contains one bit for each MSR address in the range 00000000H – 00001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Read bitmap for high MSRs** (located at the MSR-bitmap address plus 1024)**.** This contains one bit for each MSR address in the range C0000000H – C0001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Write bitmap for low MSRs** (located at the MSR-bitmap address plus 2048)**.** This contains one bit for each MSR address in the range 00000000H – 00001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.
- **Write bitmap for high MSRs** (located at the MSR-bitmap address plus 3072)**.** This contains one bit for each MSR address in the range C0000000H – C0001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.

A logical processor uses these bitmaps if and only if the “use MSR bitmaps” control is 1. If the bitmaps are used, an execution of RDMSR or WRMSR causes a VM exit if the value of RCX is in neither of the ranges covered by the bitmaps or if the appropriate bit in the MSR bitmaps (corresponding to the instruction and the RCX value) is 1. See Section 21.1.3 for details. If the bitmaps are used, their address must be 4-KByte aligned.

20.6.10 Executive-VMCS Pointer

The executive-VMCS pointer is a 64-bit field used in the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). SMM VM exits save this field as described in Section 26.15.2. VM entries that return from SMM use this field as described in Section 26.15.4.

20.6.11 Extended-Page-Table Pointer (EPTP)

The **extended-page-table pointer** (EPTP) contains the address of the base of EPML4 table (see Chapter 25), as well as other EPT configuration information. The format of this field is shown in Table 20-8.

Table 20-8. Format of Extended-Page-Table Pointer

Bit Position(s)	Field
2:0	EPT paging-structure memory type (see Section 25.5): 0 = Uncacheable (UC) 6 = Write-back (WB) Other values are reserved.[1]
5:3	EPT page-walk length (see Section 25.2)
11:6	Reserved
N–1:12	Bits N–1:12 of the physical address of the 4-KByte aligned EPT PML4 table[2]
51:N	Reserved
63:52	Reserved

NOTES:

1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix G.10) to determine what EPT paging-structure memory types are supported.
2. N is the physical-address width supported by the logical processor. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

The EPTP exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

20.6.12 Virtual-Processor Identifier (VPID)

The **virtual-processor identifier** (VPID) is a 16-bit field. It exists only on processors that support the 1-setting of the “enable VPID” VM-execution control. See Chapter 24 for details regarding the use of this field.

20.7 VM-EXIT CONTROL FIELDS

The VM-exit control fields govern the behavior of VM exits. They are discussed in Section 20.7.1 and Section 20.7.2.

20.7.1 VM-Exit Controls

The VM-exit controls constitute a 32-bit vector that governs the basic operation of VM exits. Table 20-9 lists the controls supported. See Chapter 23 for complete details of how these controls affect VM exits.

Table 20-9. Definitions of VM-Exit Controls

Bit Position(s)	Name	Description
2	Save debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are saved on VM exit. Note that the first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	Host address-space size	On processors that support Intel 64 architecture, this control determines whether a logical processor is in 64-bit mode after the next VM exit. Its value is loaded into CS.L, IA32_EFER.LME, and IA32_EFER.LMA on every VM exit.[1] This control must be 0 on processors that do not support Intel 64 architecture.
12	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM exit.
15	Acknowledge interrupt on exit	This control affects VM exits due to external interrupts: ▪ If such a VM exit occurs and this control is 1, the logical processor acknowledges the interrupt controller, acquiring the interrupt’s vector. The vector is stored in the VM-exit interruption-information field, which is marked valid. ▪ If such a VM exit occurs and this control is 0, the interrupt is not acknowledged and the VM-exit interruption-information field is marked invalid.

Table 20-9. Definitions of VM-Exit Controls (Contd.)

Bit Position(s)	Name	Description
18	Save IA32_PAT	This control determines whether the IA32_PAT MSR is saved on VM exit.
19	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM exit.
20	Save IA32_EFER	This control determines whether the IA32_EFER MSR is saved on VM exit.
21	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM exit.
22	Save VMX-preemption timer value	This control determines whether the value of the VMX-preemption timer is saved on VM exit.

NOTES:

1. Since Intel 64 architecture specifies that IA32_EFER.LMA is always set to the logical-AND of CR0.PG and IA32_EFER.LME, and since CR0.PG is always 1 in VMX operation, IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX operation.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_EXIT_CTLS and IA32_VMX_TRUE_EXIT_CTLS (see Appendix G.4) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 22.2).

Note that the first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8, 10, 11, 13, 14, 16, and 17. The VMX capability MSR IA32_VMX_EXIT_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_EXIT_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

20.7.2 VM-Exit Controls for MSRs

A VMM may specify lists of MSRs to be stored and loaded on VM exits. The following VM-exit control fields determine how MSRs are stored on VM exits:

- **VM-exit MSR-store count** (32 bits). This field specifies the number of MSRs to be stored on VM exit. It is recommended that this count not exceed 512 bytes.[1]

[1] Future implementations may allow more MSRs to be stored reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix G.6).

Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.

- **VM-exit MSR-store address** (64 bits). This field contains the physical address of the VM-exit MSR-store area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-store count. The format of each entry is given in Table 20-10. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 20-10. Format of an MSR Entry

Bit Position(s)	Contents
31:0	MSR index
63:32	Reserved
127:64	MSR data

See Section 23.4 for how this area is used on VM exits.

The following VM-exit control fields determine how MSRs are loaded on VM exits:

- **VM-exit MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM exit. It is recommended that this count not exceed 512 bytes. Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.[1]
- **VM-exit MSR-load address** (64 bits). This field contains the physical address of the VM-exit MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-load count (see Table 20-10). If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

See Section 23.6 for how this area is used on VM exits.

20.8 VM-ENTRY CONTROL FIELDS

The VM-entry control fields govern the behavior of VM entries. They are discussed in Sections 20.8.1 through 20.8.3.

20.8.1 VM-Entry Controls

The VM-entry controls constitute a 32-bit vector that governs the basic operation of VM entries. Table 20-11 lists the controls supported. See Chapter 22 for how these

[1] Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix G.6).

controls affect VM entries.

Table 20-11. Definitions of VM-Entry Controls

Bit Position(s)	Name	Description
2	Load debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are loaded on VM exit. Note that the first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	IA-32e mode guest	On processors that support Intel 64 architecture, this control determines whether the logical processor is in IA-32e mode after VM entry. Its value is loaded into IA32_EFER.LMA and IA32_EFER.LME as part of VM entry.[1] This control must be 0 on processors that do not support Intel 64 architecture.
10	Entry to SMM	This control determines whether the logical processor is in system-management mode (SMM) after VM entry. This control must be 0 for any VM entry from outside SMM.
11	Deactivate dual-monitor treatment	If set to 1, the default treatment of SMIs and SMM is in effect after the VM entry (see Section 26.15.7). This control must be 0 for any VM entry from outside SMM.
13	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM entry.
14	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM entry.
15	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM entry.

NOTES:

[1] Since Intel 64 architecture specifies that IA32_EFER.LMA is always set to the logical-AND of CR0.PG and IA32_EFER.LME, and since CR0.PG is always 1 in VMX operation; IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX operation.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_ENTRY_CTLS and IA32_VMX_TRUE_ENTRY_CTLS (see Appendix G.5) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 22.2).

Note that the first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8 and 12. The VMX capability MSR IA32_VMX_ENTRY_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_ENTRY_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

20.8.2 VM-Entry Controls for MSRs

A VMM may specify a list of MSRs to be loaded on VM entries. The following VM-entry control fields manage this functionality:

- **VM-entry MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM entry. It is recommended that this count not exceed 512 bytes. Otherwise, unpredictable processor behavior (including a machine check) may result during VM entry.[1]
- **VM-entry MSR-load address** (64 bits). This field contains the physical address of the VM-entry MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-entry MSR-load count. The format of entries is described in Table 20-10. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

See Section 22.4 for details of how this area is used on VM entries.

20.8.3 VM-Entry Controls for Event Injection

VM entry can be configured to conclude by delivering an event through the IDT (after all guest state and MSRs have been loaded). This process is called **event injection** and is controlled by the following three VM-entry control fields:

- **VM-entry interruption-information field** (32 bits). This field provides details about the event to be injected. Table 20-12 describes the field.

Table 20-12. Format of the VM-Entry Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Software interrupt 5: Privileged software exception 6: Software exception 7: Other event

[1] Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix G.6).

Table 20-12. Format of the VM-Entry Interruption-Information Field (Contd.)

Bit Position(s)	Content
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

- — The **vector** (bits 7:0) determines which entry in the IDT is used or which other event is injected.
- — The **interruption type** (bits 10:8) determines details of how the injection is performed. In general, a VMM should use the type **hardware exception** for all exceptions other than breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); it should use the type **software exception** for #BP and #OF. The type **other event** is used for injection of events that are not delivered through the IDT.
- — For exceptions, the **deliver-error-code bit** (bit 11) determines whether delivery pushes an error code on the guest stack.
- — VM entry injects an event if and only if the **valid bit** (bit 31) is 1. Note that the valid bit in this field is cleared on every VM exit.

- **VM-entry exception error code** (32 bits). This field is used if and only if the valid bit (bit 31) and the deliver-error-code bit (bit 11) are both set in the VM-entry interruption-information field.
- **VM-entry instruction length** (32 bits). For injection of events whose type is software interrupt, software exception, or privileged software exception, this field is used to determine the value of RIP that is pushed on the stack.

See Section 22.5 for details regarding the mechanics of event injection, including the use of the interruption type and the VM-entry instruction length.

VM exits clear the valid bit (bit 31) in the VM-entry interruption-information field.

20.9 VM-EXIT INFORMATION FIELDS

The VMCS contains a section of read-only fields that contain information about the most recent VM exit. Attempts to write to these fields with VMWRITE fail (see “VMWRITE—Write Field to Virtual-Machine Control Structure” in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*).

20.9.1 Basic VM-Exit Information

The following VM-exit information fields provide basic information about a VM exit:

- **Exit reason** (32 bits). This field encodes the reason for the VM exit and has the structure given in Table 20-13.

Table 20-13. Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason
27:16	Reserved (cleared to 0)
28	Pending MTF VM exit
29	VM exit from VMX root operation
30	Reserved (cleared to 0)
31	VM-entry failure (0 = true VM exit; 1 = VM-entry failure)

  — Bits 15:0 provide basic information about the cause of the VM exit (if bit 31 is clear) or of the VM-entry failure (if bit 31 is set). Appendix I enumerates the basic exit reasons.

  — Bit 28 is set only by an SMM VM exit (see Section 26.15.2) that took priority over an MTF VM exit (see Section 21.7.2) that would have occurred had the SMM VM exit not occurred. See Section 26.15.2.3.

  — Bit 29 is set if and only if the processor was in VMX root operation at the time the VM exit occurred. This can happen only for SMM VM exits. See Section 26.15.2.

  — Because some VM-entry failures load processor state from the host-state area (see Section 22.7), software must be able to distinguish such cases from true VM exits. Bit 31 is used for that purpose.

- **Exit qualification** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field contains additional information about the cause of VM exits due to the following: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); task switches; INVLPG; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; control-register accesses; MOV DR; I/O instructions; and MWAIT. The format of the field depends on the cause of the VM exit. See Section 23.2.1 for details.

- **Guest-linear address** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is used in the following cases:

  — VM exits due to attempts to execute LMSW with a memory operand.

  — VM exits due to attempts to execute INS or OUTS.

  — VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions.

— Certain VM exits due to EPT violations

See Section 23.2.1 and Section 26.15.2.3 for details of when and how this field is used.

- **Guest-physical address** (64 bits). This field is used VM exits due to EPT violations and EPT misconfigurations. See Section 23.2.1 for details of when and how this field is used.

20.9.2 Information for VM Exits Due to Vectored Events

Event-specific information is provided for VM exits due to the following vectored events: exceptions (including those generated by the instructions INT3, INTO, BOUND, and UD2); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs). This information is provided in the following fields:

- **VM-exit interruption information** (32 bits). This field receives basic information associated with the event causing the VM exit. Table 20-14 describes this field.

Table 20-14. Format of the VM-Exit Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4 – 5: Not used 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	NMI unblocking due to IRET
30:13	Reserved (cleared to 0)
31	Valid

- **VM-exit interruption error code** (32 bits). For VM exits caused by hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

Section 23.2.2 provides details of how these fields are saved on VM exits.

20.9.3 Information for VM Exits That Occur During Event Delivery

Additional information is provided for VM exits that occur during event delivery in VMX non-root operation.[1] This information is provided in the following fields:

- **IDT-vectoring information** (32 bits). This field receives basic information associated with the event that was being delivered when the VM exit occurred. Table 20-15 describes this field.

Table 20-15. Format of the IDT-Vectoring Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Software interrupt 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	Undefined
30:13	Reserved (cleared to 0)
31	Valid

- **IDT-vectoring error code** (32 bits). For VM exits the occur during delivery of hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

See Section 23.2.3 provides details of how these fields are saved on VM exits.

20.9.4 Information for VM Exits Due to Instruction Execution

The following fields are used for VM exits caused by attempts to execute certain instructions in VMX non-root operation:

- **VM-exit instruction length** (32 bits). For VM exits resulting from instruction execution, this field receives the length in bytes of the instruction whose execution led to the VM exit.[2] See Section 23.2.4 for details of when and how this field is used.

[1] This includes cases in which the event delivery was caused by event injection as part of VM entry; see Section 22.5.1.2.

- **VM-exit instruction information** (32 bits). This field is used in the following cases:
  - VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, or VMXON.
  - On some processors, VM exits due to attempts to execute INS or OUTS.[1]

  The format of the field depends on the cause of the VM exit. See Section 23.2.4 for details.

The following fields (64 bits each; 32 bits on processors that do not support Intel 64 architecture) are used only for VM exits due to SMIs that arrive immediately after retirement of I/O instructions. They provide information about that I/O instruction:

- **I/O RCX**. The value of RCX before the I/O instruction started.
- **I/O RSI**. The value of RSI before the I/O instruction started.
- **I/O RDI**. The value of RDI before the I/O instruction started.
- **I/O RIP**. The value of RIP before the I/O instruction started (the RIP that addressed the I/O instruction).

20.9.5 VM-Instruction Error Field

The 32-bit **VM-instruction error field** does not provide information about the most recent VM exit. In fact, it is not modified on VM exits. Instead, it provides information about errors encountered by a non-faulting execution of one of the VMX instructions.

20.10 SOFTWARE ACCESS TO THE VMCS AND RELATED STRUCTURES

This section details guidelines that software should observe when accessing a VMCS and related structures. It also provides descriptions of consequences for failing to follow guidelines.

20.10.1 Software Access to the Virtual-Machine Control Structure

To ensure proper processor behavior, software should observe certain guidelines when accessing an active VMCS.

[2] This field is also used for VM exits that occur during the delivery of a software interrupt or software exception.

[1] Whether the processor provides this information on these VM exits can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix G.1).

No VMCS should ever be active on more than one logical processor. If a VMCS is to be "migrated" from one logical processor to another, the first logical processor should execute VMCLEAR for the VMCS (to make it inactive on that logical processor and to ensure that all VMCS data are in memory) before the other logical processor executes VMPTRLD for the VMCS (to make it active on the second logical processor).

Software should never access or modify the VMCS data of an active VMCS using ordinary memory operations, in part because the format used to store the VMCS data is implementation-specific and not architecturally defined, and also because a logical processor may maintain some VMCS data of an active VMCS on the processor and not in the VMCS region. The following items detail some of the hazards of performing such accesses:

- Any data read from a VMCS with an ordinary memory read does not reliably reflect the state of the VMCS. Results may vary from time to time or from logical processor to logical processor.
- Writing to a VMCS with an ordinary memory write is not guaranteed to have a deterministic effect on the VMCS. Doing so may lead to unpredictable behavior. Any or all of the following may occur: (1) VM entries may fail for unexplained reasons or may load undesired processor state; (2) the processor may not correctly support VMX non-root operation as documented in Chapter 21 and may generate unexpected VM exits; and (3) VM exits may load undesired processor state, save incorrect state into the VMCS, or cause the logical processor to transition to a shutdown state.

Software can avoid such problems by removing any linear-address mappings to a VMCS region before executing a VMPTRLD for that region and by not remapping it until after executing VMCLEAR for that region.

Software should use the VMREAD and VMWRITE instructions to access the different fields in the current VMCS (see Section 20.10.2).

Software should initialize all fields in a VMCS (using VMWRITE) before using the VMCS for VM entry. Failure to do so may result in unpredictable behavior; for example, a VM entry may fail for unexplained reasons, or a successful transition (VM entry or VM exit) may load processor state with unexpected values.

20.10.2 VMREAD, VMWRITE, and Encodings of VMCS Fields

Every field of the VMCS is associated with a 32-bit value that is its **encoding**. The encoding is provided in an operand to VMREAD and VMWRITE when software wishes to read or write that field. These instructions fail if given, in 64-bit mode, an operand that sets an encoding bit beyond bit 32. See Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*, for a description of these instructions.

The structure of the 32-bit encodings of the VMCS components is determined principally by the width of the fields and their function in the VMCS. See Table 20-16.

Table 20-16. Structure of VMCS Component Encoding

Bit Position(s)	Contents
31:15	Reserved (must be 0)
14:13	Width: 0: 16-bit 1: 64-bit 2: 32-bit 3: natural-width
12	Reserved (must be 0)
11:10	Type: 0: control 1: read-only data 2: guest state 3: host state
9:1	Index
0	Access type (0 = full; 1 = high); must be full for 16-bit, 32-bit, and natural-width fields

The following items detail the meaning of the bits in each encoding:

- **Field width.** Bits 14:13 encode the width of the field.
  - A value of 0 indicates a 16-bit field.
  - A value of 1 indicates a 64-bit field.
  - A value of 2 indicates a 32-bit field.
  - A value of 3 indicates a **natural-width** field. Such fields have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

  Fields whose encodings use value 1 are specially treated to allow 32-bit software access to all 64 bits of the field. Such access is allowed by defining, for each such field, an encoding that allows direct access to the high 32 bits of the field. See below.
- **Field type.** Bits 11:10 encode the type of VMCS field: control, guest-state, host-state, or read-only data. The last category includes the VM-exit information fields and the VM-instruction error field.
- **Index.** Bits 9:1 distinguish components with the same field width and type.

- **Access type.** Bit 0 must be 0 for all fields except for 64-bit fields (those with field-width 1; see above). A VMREAD or VMWRITE using an encoding with this bit cleared to 0 accesses the entire field. For a 64-bit field with field-width 1, a VMREAD or VMWRITE using an encoding with this bit set to 1 accesses only the high 32 bits of the field.

Appendix H gives the encodings of all fields in the VMCS.

The following describes the operation of VMREAD and VMWRITE based on processor mode, VMCS-field width, and access type:

- 16-bit fields:
  - A VMREAD returns the value of the field in bits 15:0 of the destination operand; other bits of the destination operand are cleared to 0.
  - A VMWRITE writes the value of bits 15:0 of the source operand into the VMCS field; other bits of the source operand are not used.
- 32-bit fields:
  - A VMREAD returns the value of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
  - A VMWRITE writes the value of bits 31:0 of the source operand into the VMCS field; in 64-bit mode, bits 63:32 of the source operand are not used.
- 64-bit fields and natural-width fields using the full access type outside IA-32e mode.
  - A VMREAD returns the value of bits 31:0 of the field in its destination operand; bits 63:32 of the field are ignored.
  - A VMWRITE writes the value of its source operand to bits 31:0 of the field and clears bits 63:32 of the field.
- 64-bit fields and natural-width fields using the full access type in 64-bit mode (only on processors that support Intel 64 architecture).
  - A VMREAD returns the value of the field in bits 63:0 of the destination operand
  - A VMWRITE writes the value of bits 63:0 of the source operand into the VMCS field.
- 64-bit fields using the high access type.
  - A VMREAD returns the value of bits 63:32 of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
  - A VMWRITE writes the value of bits 31:0 of the source operand to bits 63:32 of the field; in 64-bit mode, bits 63:32 of the source operand are not used.

Software seeking to read a 64-bit field outside IA-32e mode can use VMREAD with the full access type (reading bits 31:0 of the field) and VMREAD with the high access type (reading bits 63:32 of the field); the order of the two VMREAD executions is not important. Software seeking to modify a 64-bit field outside IA-32e mode should first use VMWRITE with the full access type (establishing bits 31:0 of the field while clearing bits 63:32) and then use VMWRITE with the high access type (establishing bits 63:32 of the field).

20.10.3 Software Access to Related Structures

In addition to data in the VMCS region itself, VMX non-root operation can be controlled by data structures that are referenced by pointers in a VMCS (for example, the I/O bitmaps). Note that, while the pointers to these data structures are parts of the VMCS, the data structures themselves are not. They are not accessible using VMREAD and VMWRITE but by ordinary memory writes.

Software should ensure that each such data structure is modified only when no logical processor with a current VMCS that references it is in VMX non-root operation. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 20.10.1).

20.10.4 VMXON Region

Before executing VMXON, software allocates a region of memory (called the VMXON region)[1] that the logical processor uses to support VMX operation. The physical address of this region (the VMXON pointer) is provided in an operand to VMXON. The VMXON pointer is subject to the limitations that apply to VMCS pointers:

- The VMXON pointer must be 4-KByte aligned (bits 11:0 must be zero).
- On processors that support Intel 64 architecture, the VMXON pointer must not set any bits beyond the processor's physical-address width.[2] On processors that do not support Intel 64 architecture, the VMXON pointer must not set any bits in the range 63:32.

Before executing VMXON, software should write the VMCS revision identifier (see Section 20.2) to the VMXON region. It need not initialize the VMXON region in any other way. Software should use a separate region for each logical processor and should not access or modify the VMXON region of a logical processor between execution of VMXON and VMXOFF on that logical processor. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 20.10.1).

[1] The amount of memory required for the VMXON region is the same as that required for a VMCS region. This size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix G.1).

[2] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

20.11 USING VMCLEAR TO INITIALIZE A VMCS REGION

A processor may use the VMCS data portion of a VMCS region to maintain implementation-specific information about the VMCS. When software first allocates a region of memory for use as a VMCS region, the data in that region may be interpreted in an implementation-specific manner. In addition to its other functions, the VMCLEAR instruction initializes any implementation-specific information in the VMCS region referenced by its operand. To avoid the uncertainties of implementation-specific behavior, software should execute VMCLEAR on a VMCS region before making the corresponding VMCS active with VMPTRLD.

A logical processor uses the VMCS region to maintain the **launch state** of the corresponding VMCS. The launch state may be **clear** or **launched**. The VMCLEAR instruction puts the VMCS referenced by its operand into the clear state. The VMLAUNCH instruction requires a VMCS whose launch state is clear and changes its launch state to launched. The VMRESUME instruction requires a VMCS whose launch state is launched. There are no other ways to modify the launch state of a VMCS (it cannot be modified using VMWRITE) and there is no direct way to read it (it cannot be read using VMREAD). Improper software usage (for example, software writing to the VMCS data of an active VMCS) may leave the launch state undefined.

The following software usage is consistent with these limitations:

- VMCLEAR should be executed for a VMCS before it is used for VM entry.
- VMLAUNCH should be used for the first VM entry using a VMCS after VMCLEAR has been executed for that VMCS.
- VMRESUME should be used for any subsequent VM entry using a VMCS (until the next execution of VMCLEAR for the VMCS).

It is expected that, in general, VMRESUME will have lower latency than VMLAUNCH. Since “migrating” a VMCS from one logical processor to another requires use of VMCLEAR (see Section 20.10.1), which sets the launch state of the VMCS to “clear,” such migration requires the next VM entry to be performed using VMLAUNCH. Software developers can avoid the performance cost of increased VM-entry latency by avoiding unnecessary migration of a VMCS from one logical processor to another.

CHAPTER 21
VMX NON-ROOT OPERATION

In a virtualized environment using VMX, the guest software stack typically runs on a logical processor in VMX non-root operation. This mode of operation is similar to that of ordinary processor operation outside of the virtualized environment. This chapter describes the differences between VMX non-root operation and ordinary processor operation with special attention to causes of VM exits (which bring a logical processor from VMX non-root operation to root operation). The differences between VMX non-root operation and ordinary processor operation are described in the following sections:

- Section 21.1, “Instructions That Cause VM Exits”
- Section 21.2, “APIC-Access VM Exits”
- Section 21.3, “Other Causes of VM Exits”
- Section 21.4, “Changes to Instruction Behavior in VMX Non-Root Operation”
- Section 21.5, “APIC Accesses That Do Not Cause VM Exits”
- Section 21.6, “Other Changes in VMX Non-Root Operation”
- Section 21.7, “Features Specific to VMX Non-Root Operation”

Chapter 20, “Virtual-Machine Control Structures,” describes the data control structure that governs VMX operation (root and non-root). Chapter 22, “VM Entries,” describes the operation of VM entries which allow the processor to transition from VMX root operation to non-root operation.

21.1 INSTRUCTIONS THAT CAUSE VM EXITS

Certain instructions may cause VM exits if executed in VMX non-root operation. Unless otherwise specified, such VM exits are “fault-like,” meaning that the instruction causing the VM exit does not execute and no processor state is updated by the instruction. Section 23.1 details architectural state in the context of a VM exit.

Section 21.1.1 defines the prioritization between faults and VM exits for instructions subject to both. Section 21.1.2 identifies instructions that cause VM exits whenever they are executed in VMX non-root operation (and thus can never be executed in VMX non-root operation). Section 21.1.3 identifies instructions that cause VM exits depending on the settings of certain VM-execution control fields (see Section 20.6).

21.1.1 Relative Priority of Faults and VM Exits

The following principles describe the ordering between existing faults and VM exits:

- Certain exceptions have priority over VM exits. These include invalid-opcode exceptions, faults based on privilege level, and general-protection exceptions that are based on checking I/O permission bits in the task-state segment (TSS). For example, execution of RDMSR with CPL = 3 generates a general-protection exception and not a VM exit.[1]
- Faults incurred while fetching instruction operands have priority over VM exits that are conditioned based on the contents of those operands (see LMSW in Section 21.1.3).
- VM exits caused by execution of the INS and OUTS instructions (resulting either because the “unconditional I/O exiting” VM-execution control is 1 or because the “use I/O bitmaps control is 1) have priority over the following faults:
  - A general-protection fault due to the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) being unusable
  - A general-protection fault due to an offset beyond the limit of the relevant segment
  - An alignment-check exception
- Fault-like VM exits have priority over exceptions other than those mentioned above. For example, RDMSR of a non-existent MSR with CPL = 0 generates a VM exit and not a general-protection exception.

When Section 21.1.2 or Section 21.1.3 (below) identify an instruction execution that may lead to a VM exit, it is assumed that the instruction does not incur a fault that takes priority over a VM exit.

21.1.2 Instructions That Cause VM Exits Unconditionally

The following instructions cause VM exits when they are executed in VMX non-root operation: CPUID, GETSEC,[2] INVD, and XSETBV.[3] This is also true of instructions introduced with VMX, which include: VMCALL,[4] VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.

[1] MOV DR is an exception to this rule; see Section 21.1.3.

[2] An execution of GETSEC in VMX non-root operation causes a VM exit if CR4.SMXE[Bit 14] = 1 regardless of the value of CPL or RAX. An execution of GETSEC causes an invalid-opcode exception (#UD) if CR4.SMXE[Bit 14] = 0.

[3] An execution of XSETBV in VMX non-root operation causes a VM exit if CR4.OSXSAVE[Bit 18] = 1 regardless of the value of CPL, RAX, RCX, or RDX. An execution of XSETBV causes an invalid-opcode exception (#UD) if CR4.OSXSAVE[Bit 18] = 0.

[4] Under the dual-monitor treatment of SMIs and SMM, executions of VMCALL cause SMM VM exits in VMX root operation outside SMM. See Section 26.15.2.

21.1.3 Instructions That Cause VM Exits Conditionally

Certain instructions cause VM exits in VMX non-root operation depending on the setting of the VM-execution controls. The following instructions can cause "fault-like" VM exits based on the conditions described:

- **CLTS**. The CLTS instruction causes a VM exit if the bits in position 3 (corresponding to CR0.TS) are set in both the CR0 guest/host mask and the CR0 read shadow.
- **HLT.** The HLT instruction causes a VM exit if the "HLT exiting" VM-execution control is 1.
- **IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD.** The behavior of each of these instructions is determined by the settings of the "unconditional I/O exiting" and "use I/O bitmaps" VM-execution controls:
  - — If both controls are 0, the instruction executes normally.
  - — If the "unconditional I/O exiting" VM-execution control is 1 and the "use I/O bitmaps" VM-execution control is 0, the instruction causes a VM exit.
  - — If the "use I/O bitmaps" VM-execution control is 1, the instruction causes a VM exit if it attempts to access an I/O port corresponding to a bit set to 1 in the appropriate I/O bitmap (see Section 20.6.4). If an I/O operation "wraps around" the 16-bit I/O-port space (accesses ports FFFFH and 0000H), the I/O instruction causes a VM exit (the "unconditional I/O exiting" VM-execution control is ignored if the "use I/O bitmaps" VM-execution control is 1).

  See Section 21.1.1 for information regarding the priority of VM exits relative to faults that may be caused by the INS and OUTS instructions.
- **INLVPG.** The INLVPG instruction causes a VM exit if the "INLVPG exiting" VM-execution control is 1.
- **LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR.** These instructions cause VM exits if the "descriptor-table exiting" VM-execution control is 1.[1]
- **LMSW.** In general, the LMSW instruction causes a VM exit if it would write, for any bit set in the low 4 bits of the CR0 guest/host mask, a value different than the corresponding bit in the CR0 read shadow. Note that LMSW never clears bit 0 of CR0 (CR0.PE). Thus, LMSW causes a VM exit if either of the following are true:
  - — The bits in position 0 (corresponding to CR0.PE) are set in both the CR0 guest/mask and the source operand, and the bit in position 0 is clear in the CR0 read shadow.
  - — For any bit position in the range 3:1, the bit in that position is set in the CR0 guest/mask and the values of the corresponding bits in the source operand and the CR0 read shadow differ.

[1] Note that "descriptor-table exiting" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "descriptor-table exiting" VM-execution control were 0. See Section 20.6.2.

- **MONITOR.** The MONITOR instruction causes a VM exit if the "MONITOR exiting" VM-execution control is 1.
- **MOV from CR3.** The MOV from CR3 instruction causes a VM exit if the "CR3-store exiting" VM-execution control is 1. Note that the first processors to support the virtual-machine extensions supported only the 1-setting of this control.
- **MOV from CR8.** The MOV from CR8 instruction (which can be executed only in 64-bit mode) causes a VM exit if the "CR8-store exiting" VM-execution control is 1. Note that, if this control is 0, the behavior of the MOV from CR8 instruction is modified if the "use TPR shadow" VM-execution control is 1 (see Section 21.4).
- **MOV to CR0.** The MOV to CR0 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR0 guest/host mask, the corresponding bit in the CR0 read shadow. (If every bit is clear in the CR0 guest/host mask, MOV to CR0 cannot cause a VM exit.)
- **MOV to CR3.** The MOV to CR3 instruction causes a VM exit unless the "CR3-load exiting" VM-execution control is 0 or the value of its source operand is equal to one of the CR3-target values specified in the VMCS. Note that, if the CR3-target count in *n*, only the first *n* CR3-target values are considered; if the CR3-target count is 0, MOV to CR3 always causes a VM exit.

  Note that the first processors to support the virtual-machine extensions supported only the 1-setting of the "CR3-load exiting" VM-execution control. These processors always consult the CR3-target controls to determine whether an execution of MOV to CR3 causes a VM exit.
- **MOV to CR4.** The MOV to CR4 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR4 guest/host mask, the corresponding bit in the CR4 read shadow.
- **MOV to CR8.** The MOV to CR8 instruction (which can be executed only in 64-bit mode) causes a VM exit if the "CR8-load exiting" VM-execution control is 1. Note that, if this control is 0, the behavior of the MOV to CR8 instruction is modified if the "use TPR shadow" VM-execution control is 1 (see Section 21.4) and it may cause a trap-like VM exit (see below).
- **MOV DR.** The MOV DR instruction causes a VM exit if the "MOV-DR exiting" VM-execution control is 1. Such VM exits represent an exception to the principles identified in Section 21.1.1; they take priority over all faults that may occur in the execution of MOV DR.
- **MWAIT.** The MWAIT instruction causes a VM exit if the "MWAIT exiting" VM-execution control is 1.
- **PAUSE.** The PAUSE instruction causes a VM exit if the "PAUSE exiting" VM-execution control is 1.
- **RDMSR.** The RDMSR instruction causes a VM exit if any of the following are true:
  - The "use MSR bitmaps" VM-execution control is 0.
  - The value of ECX is not in the range 00000000H – 00001FFFH or C0000000H – C0001FFFH.

- — The value of ECX is in the range 00000000H – 00001FFFH and bit *n* in read bitmap for low MSRs is 1, where *n* is the value of ECX.
- — The value of ECX is in the range C0000000H – C0001FFFH and bit *n* in read bitmap for high MSRs is 1, where *n* is the value of ECX & 00001FFFH.

See Section 20.6.9 for details regarding how these bitmaps are identified.

- **RDPMC.** The RDPMC instruction causes a VM exit if the “RDPMC exiting” VM-execution control is 1.
- **RDTSC.** The RDTSC instruction causes a VM exit if the “RDTSC exiting” VM-execution control is 1.
- **RSM.** The RSM instruction causes a VM exit if executed in system-management mode (SMM).[1]
- **WBINVD.** The WBINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.[2]
- **WRMSR.** The WRMSR instruction causes a VM exit if any of the following are true:
  - — The “use MSR bitmaps” VM-execution control is 0.
  - — The value of ECX is not in the range 00000000H – 00001FFFH or C0000000H – C0001FFFH.
  - — The value of ECX is in the range 00000000H – 00001FFFH and bit *n* in write bitmap for low MSRs is 1, where *n* is the value of ECX.
  - — The value of ECX is in the range C0000000H – C0001FFFH and bit *n* in write bitmap for high MSRs is 1, where *n* is the value of ECX & 00001FFFH.

  See Section 20.6.9 for details regarding how these bitmaps are identified.

  If an execution of WRMSR does not cause a VM exit as specified above and ECX = 808H (indicating the TPR MSR), instruction behavior is modified if the “virtualize x2APIC mode” VM-execution control is 1 (see Section 21.4) and it may cause a trap-like VM exit (see below).[3]

The MOV to CR8 and WRMSR instructions may cause “trap-like” VM exits. In such a case, the instruction completes before the VM exit occurs and that processor state is updated by the instruction (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

[1] Execution of the RSM instruction outside SMM causes an invalid-opcode exception regardless of whether the processor is in VMX operation. It also does so in VMX root operation in SMM; see Section 26.15.3.

[2] Note that “WBINVD exiting” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “WBINVD exiting” VM-execution control were 0. See Section 20.6.2.

[3] Note that “virtualize x2APIC mode” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “virtualize x2APIC mode” VM-execution control were 0. See Section 20.6.2.

Specifically, a trap-like VM exit occurs following either instruction if the execution reduces the value of the TPR shadow below that of the TPR threshold VM-execution control field (see Section 20.6.8 and Section 21.4) and the following hold:

- For MOV to CR8:
  - — The “CR8-load exiting” VM-execution control is 0.
  - — The “use TPR shadow” VM-execution control is 1.
- For WRMSR:
  - — The “use MSR bitmaps” VM-execution control is 1, the value of ECX is 808H, and bit 808H in write bitmap for low MSRs is 0 (see above).
  - — The “virtualize x2APIC mode” VM-execution control is 1.

21.2 APIC-ACCESS VM EXITS

If the “virtualize APIC accesses” VM-execution control is 1, an attempt to access memory using a physical address on the APIC-access page (see Section 20.6.8) causes a VM exit.[1,2] Such a VM exit is called an **APIC-access VM exit**.

Whether an operation that attempts to access memory with a physical address on the APIC-access page causes an APIC-access VM exit may be qualified based on the type of access. Section 21.2.1 describes the treatment of linear accesses, while Section 21.2.3 describes that of physical accesses. Section 21.2.4 discusses accesses to the TPR field on the APIC-access page (called VTPR accesses), which do not, if the “use TPR shadow” VM-execution control is 1, cause APIC-access VM exits.

21.2.1 Linear Accesses to the APIC-Access Page

An access to the APIC-access page is called a **linear access** if (1) it results from a memory access using a linear address; and (2) the access’s physical address is the translation of that linear address. Section 21.2.1.1 specifies which linear accesses to the APIC-access page cause APIC-access VM exits.

In general, the treatment of APIC-access VM exits caused by linear accesses is similar to that of page faults and EPT violations. Based upon this treatment, Section 21.2.1.2 specifies the priority of such VM exits with respect to other events, while Section 21.2.1.3 discusses instructions that may cause page faults without accessing memory and the treatment when they access the APIC-access page.

[1] Note that “virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 20.6.2.

[2] Even when addresses are translated using EPT (see Chapter 25), the determination of whether an APIC-access VM exit occurs depends on an access’s physical address, not its guest-physical address.

21.2.1.1 Linear Accesses That Cause APIC-Access VM Exits

Whether a linear access to the APIC-access page causes an APIC-access VM exit depends in part of the nature of the translation used by the linear address:

- If the linear access uses a translation with a 4-KByte page, it causes an APIC-access VM exit.
- If the linear access uses a translation with a large page (2-MByte or 4-MByte), the access may or may not cause an APIC-access VM exit. Section 21.5.1 describes the treatment of such accesses that do not cause an APIC-access VM exits.

  If EPT is in use, a linear access uses a translation with a large page only if a large page is specified by both the guest paging structures and the EPT paging structures.

It is recommended that software configure the paging structures so that any translation to the APIC-access page uses a 4-KByte page.

A linear access to the APIC-access page might not cause an APIC-access VM exit if the “enable EPT” VM-execution control is 1 and software has not properly invalidate information cached from the EPT paging structures:

- At time t_1, EPT was in use, the EPTP value was X, and some guest-physical address Y translated to an address that was not on the APIC-access page at that time. (This might be because the “virtualize APIC accesses” VM-execution control was 0.)
- At later time t_2, EPT is in use, the EPTP value is X, and a memory access uses a linear address that translates to Y, which now translates to an address on the APIC-access page. (This implies that the “virtualize APIC accesses” VM-execution control is 1 at this time.)
- Software did not execute the INVEPT instruction between times t_1 and t_2, either with the all-context INVEPT type or with the single-context INVEPT type and X as the INVEPT descriptor.

In this case, the linear access at time t_2 might or might not cause an APIC-access VM exit. If it does not, the access operates on memory on the APIC-access page.

Software can avoid this situation through appropriate use of the INVEPT instruction; see Section 25.6.3.

A linear access to the APIC-access page might not cause an APIC-access VM exit if the “enable VPID” VM-execution control is 1 and software has not properly invalidated the TLBs and paging-structure caches:

- At time t_1, the processor was in VMX non-root operation with non-zero VPID X, and some linear address Y translated to an address that was not on the APIC-access page at that time. (This might be because the “virtualize APIC accesses” VM-execution control was 0.)
- At later time t_2, the processor was again in VMX non-root operation with VPID X, and a memory access uses linear address, which now translates to an address on

the APIC-access page. (This implies that the "virtualize APIC accesses" VM-execution control is 1 at this time.)

- Software did not execute the INVVPID instruction in any of the following ways between times t_1 and t_2:
  - — With the individual-address INVVPID type and an INVVPID descriptor specifying VPID X and linear address Y.
  - — With the single-context INVVPID type and an INVVPID descriptor specifying VPID X.
  - — With the all-context INVEPT type.
  - — With the single-context-retaining-globals INVVPID type and an INVVPID descriptor specifying VPID X (assuming that, at time t1, the translation for Y was global; see Section 3.12, "Translation Lookaside Buffers (TLBs)" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* for details regarding global translations).

In this case, the linear access at time t_2 might or might not cause an APIC-access VM exit. If it does not, the access operates on memory on the APIC-access page.

Software can avoid this situation through appropriate use of the INVVPID instruction; see Section 24.4.

21.2.1.2 Priority of APIC-Access VM Exits Caused by Linear Accesses

The following items specify the priority relative to other events of APIC-access VM exits caused by linear accesses.

- The priority of an APIC-access VM exit on a linear access to memory is below that of any page fault or EPT violation that that access may incur. That is, a linear access does not cause an APIC-access VM exit if it would cause a page fault or an EPT violation.
- A linear access does not cause an APIC-access VM exit until after the accessed bits are set in the paging structures.
- A linear write access will not cause an APIC-access VM exit until after the dirty bit is set in the appropriate paging structure.
- With respect to all other events, any APIC-access VM exit due to a linear access has the same priority as any page fault or EPT violation that the linear access could cause. (This item applies to other events that the linear access may generate as well as events that may be generated by other accesses by the same instruction or operation.)

These principles imply among other things, that an APIC-access VM exit may occur during the execution of a repeated string instruction (including INS and OUTS). Suppose, for example, that the first *n* iterations (*n* may be 0) of such an instruction do not access the APIC-access page and that the next iteration does access that page. As a result, the first *n* iterations may complete and be followed by an APIC-access VM exit. The instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers reflect the completion of *n* iterations.

21.2.1.3 Instructions That May Cause Page Faults or EPT Violations Without Accessing Memory

APIC-access VM exits may occur as a result of executing an instruction that can cause a page fault or an EPT violation even if that instruction would not access the APIC-access page. The following are some examples:

- The CLFLUSH instruction is considered to read from the linear address in its source operand. If that address translates to one on the APIC-access page, the instruction causes an APIC-access VM exit.
- The ENTER instruction causes a page fault if the byte referenced by the final value of the stack pointer is not writable (even though ENTER does not write to that byte if its size operand is non-zero). If that byte is writable but is on the APIC-access page, ENTER causes an APIC-access VM exit.[1]
- An execution of the MASKMOVQ or MASKMOVDQU instructions with a zero mask may or may not cause a page fault or an EPT violation if the destination page is unwritable (the behavior is implementation-specific). An execution with a zero mask causes an APIC-access VM exit only on processors for which it could cause a page fault or an EPT violation.
- The MONITOR instruction is considered to read from the effective address in RAX. If the linear address corresponding to that address translates to one on the APIC-access page, the instruction causes an APIC-access VM exit.[2]
- An execution of the PREFETCH instruction that would result in an access to the APIC-access page does not cause an APIC-access VM exit.

21.2.2 Guest-Physical Accesses to the APIC-Access Page

An access to the APIC-access page is called a **guest-physical access** if (1) guest-physical addresses are being translated using EPT (see Chapter 25); (2) the access's physical address is the result of an EPT translation; and (3) either (a) the access was not generated by a linear address; or (b) the access's guest-physical address is not

[1] The ENTER instruction may also cause page faults due to the memory accesses that it actually does perform. With regard to APIC-access VM exits, these are treated just as accesses by any other instruction.

[2] This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

the translation of the access’s linear address. Guest-physical accesses include the following when guest-physical addresses are being translated using EPT:

- Reads from the guest paging structures when translating a linear address (such an access uses a guest-physical address that is not the translation of that linear address).
- Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging.[1]
- Updates to the accessed and dirty bits in the guest paging structures when using a linear address (such an access uses a guest-physical address that is not the translation of that linear address).

Section 21.2.2.1 specifies when guest-physical accesses to the APIC-access page might not cause APIC-access VM exits. In general, the treatment of APIC-access VM exits caused by guest-physical accesses is similar to that of EPT violations. Based upon this treatment, Section 21.2.2.2 specifies the priority of such VM exits with respect to other events.

21.2.2.1 Guest-Physical Accesses That Might Not Cause APIC-Access VM Exits

Whether a guest-physical access to the APIC-access page causes an APIC-access VM exit depends on the nature of the EPT translation used by the guest-physical address and on how software is managing information cached from the EPT paging structures. The following items detail cases in which a guest-physical access to the APIC-access page might not an APIC-access VM exit:

- If the access uses a guest-physical address whose translation to the APIC-access page uses an EPT PDE that maps a 2-MByte page (because bit 7 of the EPT PDE is 1).
- Software has not properly invalidated information cached from the EPT paging structures:
  - At time t_1, EPT was in use, the EPTP value was X, and some guest-physical address Y translated to an address that was not on the APIC-access page at that time. (This might be because the “virtualize APIC accesses” VM-execution control was 0.)
  - At later time t_2, the EPTP value is X and a memory access uses guest-physical address Y, which now translates to an address on the APIC-access page. (This implies that the “virtualize APIC accesses” VM-execution control is 1 at this time.)
  - Software did not execute the INVEPT instruction, either with the all-context INVEPT type or with the single-context INVEPT type and X as the INVEPT descriptor, between times t_1 and t_2.

[1] A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 3.8 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

In any of the above cases, the guest-physical access at time t_2 might or might not an APIC-access VM exit. If it does not, the access operates on memory on the APIC-access page.

Software can avoid this situation through appropriate use of the INVEPT instruction; see Section 25.6.3.

21.2.2.2 Priority of APIC-Access VM Exits Caused by Guest-Physical Accesses

The following items specify the priority relative to other events of APIC-access VM exits caused by guest-physical accesses.

- The priority of an APIC-access VM exit caused by a guest-physical access to memory is below that of any EPT violation that that access may incur. That is, a guest-physical access does not cause an APIC-access VM exit if it would cause an EPT violation.
- With respect to all other events, any APIC-access VM exit caused by a guest-physical access has the same priority as any EPT violation that the guest-physical access could cause.

21.2.3 Physical Accesses to the APIC-Access Page

An access to the APIC-access page is called a **physical access** if (1) either (a) the "enable EPT" VM-execution control is 0;[1] or (b) the access's physical address is not the result of a translation through the EPT paging structures; and (2) either (a) the access is not generated by a linear address; or (b) the access's physical address is not the translation of its linear address.

Physical accesses include the following:

- If the "enable EPT" VM-execution control is 0:
  - Reads from the paging structures when translating a linear address.
  - Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging.[2]
  - Updates to the accessed and dirty bits in the paging structures.
- If the "enable EPT" VM-execution control is 1, accesses to the EPT paging structures.

[1] Note that "enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "enable EPT" VM-execution control were 0. See Section 20.6.2.

[2] A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 3.8 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

- Any of the following accesses made by the processor to support VMX non-root operation:
  - Accesses to the VMCS region.
  - Accesses to data structures referenced (directly or indirectly) by physical addresses in VM-execution control fields in the VMCS. These include the I/O bitmaps, the MSR bitmaps, and the virtual-APIC page.
- Accesses that effect transitions into and out of SMM.[1] These include the following:
  - Accesses to SMRAM during SMI delivery and during execution of RSM.
  - Accesses during SMM VM exits (including accesses to MSEG) and during VM entries that return from SMM.

A physical access to the APIC-access page may or may not cause an APIC-access VM exit. (A physical write to the APIC-access page may write to memory as specified in Section 21.5.2 before causing the APIC-access VM exit.) The priority of an APIC-access VM exit caused by physical access is not defined relative to other events that the access may cause. Section 21.5.2 describes the treatment of physical accesses to the APIC-access page that do not cause APIC-access VM exits.

It is recommended that software not set the APIC-access address to any of those used by physical memory accesses (identified above). For example, it should not set the APIC-access address to the physical address of any of the active paging structures if the “enable EPT” VM-execution control is 0.

21.2.4 VTPR Accesses

A memory access is a **VTPR access** if all of the following hold: (1) the “use TPR shadow” VM-execution control is 1; (2) the access is not for an instruction fetch; (3) the access is at most 32 bits in width; and (4) the access is to offset 80H on the APIC-access page.

A memory access is not a VTPR access (even if it accesses only bytes in the range 80H–83H on the APIC-access page) if any of the following hold: (1) the “use TPR shadow” VM-execution control is 0; (2) the access is for an instruction fetch; (3) the access is more than 32 bits in width; or (4) the access is to some offset is on the APIC-access page other than 80H. For example, a 16-bit access to offset 81H on the APIC-access page is **not** a VTPR access, even if the “use TPR shadow” VM-execution control is 1.

In general, VTPR accesses do not cause APIC-access VM exits. Instead, they are treated as described in Section 21.5.3. Physical VTPR accesses (see Section 21.2.3) may or may not cause APIC-access VM exits; see Section 21.5.2.

[1] Technically, these accesses do not occur in VMX non-root operation. They are included here for clarity.

21.3 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 20.6.3). If an exception occurs, its vector (in the range 0–31) is used to select a bit in the exception bitmap. If the bit is 1, a VM exit occurs; if the bit is 0, the exception is delivered normally through the guest IDT. This use of the exception bitmap applies also to exceptions generated by the instructions INT3, INTO, BOUND, and UD2.

  Page faults (exceptions with vector 14) are specially treated. When a page fault occurs, a logical processor consults (1) bit 14 of the exception bitmap; (2) the error code produced with the page fault [PFEC]; (3) the page-fault error-code mask field [PFEC_MASK]; and (4) the page-fault error-code match field [PFEC_MATCH]. It checks if PFEC & PFEC_MASK = PFEC_MATCH. If there is equality, the specification of bit 14 in the exception bitmap is followed (for example, a VM exit occurs if that bit is set). If there is inequality, the meaning of that bit is reversed (for example, a VM exit occurs if that bit is clear).

  Thus, if software desires VM exits on all page faults, it can set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 00000000H. If software desires VM exits on no page faults, it can set bit 14 in the exception bitmap to 1, the page-fault error-code mask field to 00000000H, and the page-fault error-code match field to FFFFFFFFH.
- **Triple fault.** A VM exit occurs if the logical processor encounters an exception while attempting to call the double-fault handler and that exception itself does not cause a VM exit due to the exception bitmap. This applies to the case in which the double-fault exception was generated within VMX non-root operation, the case in which the double-fault exception was generated during event injection by VM entry, and to the case in which VM entry is injecting a double-fault exception.
- **External interrupts.** An external interrupt causes a VM exit if the “external-interrupt exiting” VM-execution control is 1. Otherwise, the interrupt is delivered normally through the IDT. (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The interrupt is not delivered through the IDT and no VM exit occurs.)
- **Non-maskable interrupts (NMIs).** An NMI causes a VM exit if the “NMI exiting” VM-execution control is 1. Otherwise, it is delivered using descriptor 2 of the IDT. (If a logical processor is in the wait-for-SIPI state, NMIs are blocked. The NMI is not delivered through the IDT and no VM exit occurs.)
- **INIT signals.** INIT signals cause VM exits. A logical processor performs none of the operations normally associated with these events. Such exits do not modify register state or clear pending events as they would outside of VMX operation. (If a logical processor is in the wait-for-SIPI state, INIT signals are blocked. They do not cause VM exits in this case.)

- **Start-up IPIs (SIPIs). SIPIs cause VM exits.** If a logical processor is not in the wait-for-SIPI activity state when a SIPI arrives, no VM exit occurs and the SIPI is discarded. VM exits due to SIPIs do not perform any of the normal operations associated with those events: they do not modify register state as they would outside of VMX operation. (If a logical processor is not in the wait-for-SIPI state, SIPIs are blocked. They do not cause VM exits in this case.)
- **Task switches.** Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. See Section 21.6.2.
- **System-management interrupts (SMIs).** If the logical processor is using the dual-monitor treatment of SMIs and system-management mode (SMM), SMIs cause SMM VM exits. See Section 26.15.2.[1]
- **VMX-preemption timer.** A VM exit occurs when the timer counts down to zero. See Section 21.7.1 for details of operation of the VMX-preemption timer. As noted in that section, the timer does not cause VM exits if the logical processor is outside the C-states C0, C1, and C2.

  Debug-trap exceptions and higher priority events take priority over VM exits caused by the VMX-preemption timer. VM exits caused by the VMX-preemption timer take priority over VM exits caused by the “NMI-window exiting” VM-execution control and lower priority events.

  These VM exits wake a logical processor from the same inactive states as would an external interrupt. Specifically, they wake a logical processor from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.

In addition, there are controls that cause VM exits based on the readiness of guest software to receive interrupts:

- If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if RFLAGS.IF = 1 and there is no blocking of events by STI or by MOV SS (see Table 20-3). Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 22.6.5).

  Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.

  These VM exits wake a logical processor from the same inactive states as would an external interrupt. Specifically, they wake a logical processor from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.
- If the “NMI-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if there is no virtual-NMI blocking and there is no

[1] Under the dual-monitor treatment of SMIs and SMM, SMIs also cause SMM VM exits if they occur in VMX root operation outside SMM. If the processor is using the default treatment of SMIs and SMM, SMIs are delivered as described in Section 26.14.1.

blocking of events by MOV SS (see Table 20-3). (A logical processor may also prevent such a VM exit if there is blocking of events by STI.) Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 22.6.6).

VM exits caused by the VMX-preemption timer and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an NMI. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

21.4 CHANGES TO INSTRUCTION BEHAVIOR IN VMX NON-ROOT OPERATION

The behavior of some instructions is changed in VMX non-root operation. Some of these changes are determined by the settings of certain VM-execution control fields. The following items detail such changes:

- **CLTS.** Behavior of the CLTS instruction is determined by the bits in position 3 (corresponding to CR0.TS) in the CR0 guest/host mask and the CR0 read shadow:
  - — If bit 3 in the CR0 guest/host mask is 0, CLTS clears CR0.TS normally (the value of bit 3 in the CR0 read shadow is irrelevant in this case), unless CR0.TS is fixed to 1 in VMX operation (see Section 19.8), in which case CLTS causes a general-protection exception.
  - — If bit 3 in the CR0 guest/host mask is 1 and bit 3 in the CR0 read shadow is 0, CLTS completes but does not change the contents of CR0.TS.
  - — If the bits in position 3 in the CR0 guest/host mask and the CR0 read shadow are both 1, CLTS causes a VM exit (see Section 21.1.3).
- **IRET.** Behavior of IRET with regard to NMI blocking (see Table 20-3) is determined by the settings of the “NMI exiting” and “virtual NMIs” VM-execution controls:
  - — If the “NMI exiting” VM-execution control is 0, IRET operates normally and unblocks NMIs.
  - — If the “NMI exiting” VM-execution control is 1, IRET does not affect blocking of NMIs.
  - — If the “virtual NMIs” VM-execution control is 1, the logical processor tracks virtual-NMI blocking. In this case, IRET removes any virtual-NMI blocking.

  If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0. (See Section 22.2.1.1.)

- **LMSW.** An execution of LMSW that does not cause a VM exit (see Section 21.1.3) leaves unmodified any bit in CR0 corresponding to a bit set in the CR0 guest/host mask. It causes a general-protection exception if it attempts to set any bit to a value not supported in VMX operation (see Section 19.8)
- **MOV from CR0.** The behavior of MOV from CR0 is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, MOV from CR0 reads normally from CR0; if every bit is set in the CR0 guest/host mask, MOV from CR0 returns the value of the CR0 read shadow.

  Note that, depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.
- **MOV from CR3.** If the “enable EPT” VM-execution control is 1 and an execution of MOV from CR3 does not cause a VM exit (see Section 21.1.3), the value loaded from CR3 is a guest-physical address; see Section 25.1.
- **MOV from CR4.** The behavior of MOV from CR4 is determined by the CR4 guest/host mask and the CR4 read shadow. For each position corresponding to a bit clear in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR4. For each position corresponding to a bit set in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR4 read shadow. Thus, if every bit is cleared in the CR4 guest/host mask, MOV from CR4 reads normally from CR4; if every bit is set in the CR4 guest/host mask, MOV from CR4 returns the value of the CR4 read shadow.

  Note that, depending on the contents of the CR4 guest/host mask and the CR4 read shadow, bits may be set in the destination that would never be set when reading directly from CR4.
- **MOV from CR8.** Behavior of the MOV from CR8 instruction (which can be executed only in 64-bit mode) is determined by the settings of the “CR8-store exiting” and “use TPR shadow” VM-execution controls:
  - If both controls are 0, MOV from CR8 operates normally.
  - If the “CR8-store exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution control is 1, MOV from CR8 reads from the TPR shadow. Specifically, it loads bits 3:0 of its destination operand with the value of bits 7:4 of byte 80H of the virtual-APIC page (see Section 20.6.8). Bits 63:4 of the destination operand are cleared.
  - If the “CR8-store exiting” VM-execution control is 1, MOV from CR8 causes a VM exit (see Section 21.1.3); the “use TPR shadow” VM-execution control is ignored in this case.

- **MOV to CR0.** An execution of MOV to CR0 that does not cause a VM exit (see Section 21.1.3) leaves unmodified any bit in CR0 corresponding to a bit set in the CR0 guest/host mask. It causes a general-protection exception if it attempts to set any bit to a value not supported in VMX operation (see Section 19.8).
- **MOV to CR3.** If the "enable EPT" VM-execution control is 1 and an execution of MOV to CR3 does not cause a VM exit (see Section 21.1.3), the value loaded into CR3 is treated as a guest-physical address; see Section 25.1.
  - If PAE paging is not being used, the instruction does not use the guest-physical address to access memory and it does not cause it to be translated through EPT.[1]
  - If PAE paging is being used, the instruction translates the guest-physical address through EPT and uses the result to load the four (4) page-directory-pointer-table entries (PDPTEs). The instruction does not use the guest-physical addresses the PDPTEs to access memory and it does not cause them to be translated through EPT.
- **MOV to CR4.** An execution of MOV to CR4 that does not cause a VM exit (see Section 21.1.3) leaves unmodified any bit in CR4 corresponding to a bit set in the CR4 guest/host mask. Such an execution causes a general-protection exception if it attempts to set any bit to a value not supported in VMX operation (see Section 19.8).
- **MOV to CR8.** Behavior of the MOV to CR8 instruction (which can be executed only in 64-bit mode) is determined by the settings of the "CR8-load exiting" and "use TPR shadow" VM-execution controls:
  - If both controls are 0, MOV to CR8 operates normally.
  - If the "CR8-load exiting" VM-execution control is 0 and the "use TPR shadow" VM-execution control is 1, MOV to CR8 writes to the TPR shadow. Specifically, it stores bits 3:0 of its source operand into bits 7:4 of byte 80H of the virtual-APIC page (see Section 20.6.8); bits 3:0 of that byte and bytes 129-131 of that page are cleared. Such a store may cause a VM exit to occur after it completes (see Section 21.1.3).
  - If the "CR8-load exiting" VM-execution control is 1, MOV to CR8 causes a VM exit (see Section 21.1.3); the "use TPR shadow" VM-execution control is ignored in this case.
- **RDMSR.** Section 21.1.3 identifies when executions of the RDMSR instruction cause VM exits. If such an execution causes neither a fault due to CPL > 0 nor a VM exit, the instruction's behavior may be modified for certain values of ECX:
  - If ECX contains 10H (indicating the IA32_TIME_STAMP_COUNTER MSR), the value returned by the instruction is determined by the setting of the "use TSC offsetting" VM-execution control as well as the TSC offset:

[1] A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 3.8 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

    - If the control is 0, the instruction operates normally, loading EAX:EDX with the value of the IA32_TIME_STAMP_COUNTER MSR.
    - If the control is 1, the instruction loads EAX:EDX with the sum (using signed addition) of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset (interpreted as a signed value).
  - If ECX contains 808H (indicating the TPR MSR), instruction behavior is determined by the setting of the “virtualize x2APIC mode” VM-execution control:[1]
    - If the control is 0, the instruction operates normally. If the local APIC is in x2APIC mode, EAX[7:0] is loaded with the value of the APIC’s task-priority register (EDX and EAX[31:8] are cleared to 0). If the local APIC is not in x2APIC mode, a general-protection fault occurs.
    - If the control is 1, the instruction loads EAX:EDX with the value of bytes 87H:80H of the virtual-APIC page. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not x2APIC mode).
- **RDTSC.** Behavior of the RDTSC instruction is determined by the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls as well as the TSC offset:
  - If both controls are 0, RDTSC operates normally.
  - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, RDTSC loads EAX:EDX with the sum (using signed addition) of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset (interpreted as a signed value).
  - If the “RDTSC exiting” VM-execution control is 1, RDTSC causes a VM exit (see Section 21.1.3).
- **SMSW.** The behavior of SMSW is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, MOV from CR0 reads normally from CR0; if every bit is set in the CR0 guest/host mask, MOV from CR0 returns the value of the CR0 read shadow.

  Note the following: (1) for any memory destination or for a 16-bit register destination, only the low 16 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:16 of a register destination are left unchanged); (2) for a 32-bit register destination, only the low 32 bits of the CR0 guest/host mask and the CR0

[1] Note that “virtualize x2APIC mode” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “virtualize x2APIC mode” VM-execution control were 0. See Section 20.6.2.

read shadow are used (bits 63:32 of the destination are cleared); and (3) depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.

- **WRMSR.** Section 21.1.3 identifies when executions of the WRMSR instruction cause VM exits. If such an execution neither a fault due to CPL > 0 nor a VM exit, the instruction's behavior may be modified for certain values of ECX:
  - — If ECX contains 79H (indicating IA32_BIOS_UPDT_TRIG MSR), no microcode update is loaded, and control passes to the next instruction. This implies that microcode updates cannot be loaded in VMX non-root operation.
  - — If ECX contains 808H (indicating the TPR MSR) and either EDX or EAX[31:8] is non-zero, a general-protection fault occurs (this is true even if the logical processor is not in VMX non-root operation). Otherwise, instruction behavior is determined by the setting of the "virtualize x2APIC mode" VM-execution control and the value of the TPR-threshold VM-execution control field:
    - If the control is 0, the instruction operates normally. If the local APIC is in x2APIC mode, the value of EAX[7:0] is written to the APIC's task-priority register. If the local APIC is not in x2APIC mode, a general-protection fault occurs.
    - If the control is 1, the instruction stores the value of EAX:EDX to bytes 87H:80H of the virtual-APIC page. This store occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not x2APIC mode). The store may cause a VM exit to occur after the instruction completes (see Section 21.1.3).

21.5 APIC ACCESSES THAT DO NOT CAUSE VM EXITS

As noted in Section 21.2, if the "virtualize APIC accesses" VM-execution control is 1, most memory accesses to the APIC-access page (see Section 20.6.8) cause APIC-access VM exits.[1] Section 21.2 identifies potential exceptions. These are covered in Section 21.5.1 through Section 21.5.3.

In some cases, an attempt to access memory on the APIC-access page is converted to an access to the virtual-APIC page (see Section 20.6.8). In these cases, the access uses the memory type reported in bit 53:50 of the IA32_VMX_BASIC MSR (see Appendix G.1).

[1] Note that "virtualize APIC accesses" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "virtualize APIC accesses" VM-execution control were 0. See Section 20.6.2.

21.5.1 Linear Accesses to the APIC-Access Page Using Large-Page Translations

As noted in Section 21.2.1, a linear access to the APIC-access page using translation with a large page (2-MByte or 4-MByte) may or may not cause an APIC-access VM exit. If it does not and the access is not a VTPR access (see Section 21.2.4), the access operates on memory on the APIC-access page. Section 21.5.3 describes the treatment if there is no APIC-access VM exit and the access is a VTPR access.

21.5.2 Physical Accesses to the APIC-Access Page

A physical access to the APIC-access page may or may not cause an APIC-access VM exit. If it does not and the access is not a VTPR access (see Section 21.2.4), the access operates on memory on the APIC-access page (this may happen if the access causes an APIC-access VM exit). Section 21.5.3 describes the treatment if there is no APIC-access VM exit and the access is a VTPR access.

21.5.3 VTPR Accesses

As noted in Section 21.2.4, a memory access is a VTPR access if all of the following hold: (1) the “use TPR shadow” VM-execution control is 1; (2) the access is not for an instruction fetch; (3) the access is at most 32 bits in width; and (4) the access is to offset 80H on the APIC-access page.

The treatment of VTPR accesses depends on the nature of the access:

- A linear VTPR access using a translation with a 4-KByte page does not cause an APIC-access VM exit. Instead, it is converted so that, instead of accessing offset 80H on the APIC-access page, it accesses offset 80H on the virtual-APIC page. Further details are provided in Section 21.5.3.1 to Section 21.5.3.3.
- A linear VTPR access using a translation with a large page (2-MByte or 4-MByte) may be treated in either of two ways:
  - It may operate on memory on the APIC-access page. The details in Section 21.5.3.1 to Section 21.5.3.3 do not apply.
  - It may be converted so that, instead of accessing offset 80H on the APIC-access page, it accesses offset 80H on the virtual-APIC page. Further details are provided in Section 21.5.3.1 to Section 21.5.3.3.
- A physical VTPR access may be treated in one of three ways:
  - It may cause an APIC-access VM exit. The details in Section 21.5.3.1 to Section 21.5.3.3 do not apply.
  - It may operate on memory on the APIC-access page (and possibly then cause an APIC-access VM exit). The details in Section 21.5.3.1 to Section 21.5.3.3 do not apply.

  - — It may be converted so that, instead of accessing offset 80H on the APIC-access page, it accesses offset 80H on the virtual-APIC page. Further details are provided in Section 21.5.3.1 to Section 21.5.3.3.

Linear VTPR accesses never cause APIC-access VM exits (recall that an access is a VTPR access only if the “use TPR shadow” VM-execution control is 1).

21.5.3.1 Treatment of Individual VTPR Accesses

The following items detail the treatment of VTPR accesses:

- VTPR read accesses. Such an access completes normally (reading data from the field at offset 80H on the virtual-APIC page).

  The following items detail certain instructions that are considered to perform read accesses and how they behavior when accessing the VTPR:

  - — A VTPR access using the CLFLUSH instruction flushes data for offset 80H on the virtual-APIC page.
  - — A VTPR access using the LMSW instruction may cause a VM exit due to the CR0 guest/host mask and the CR0 read shadow.
  - — A VTPR access using the MONITOR instruction causes the logical processor to monitor offset 80H on the virtual-APIC page.
  - — A VTPR access using the PREFETCH instruction may prefetch data; if so, it is from offset 80H on the virtual-APIC page.

- VTPR write accesses. Such an access completes normally (writing data to the field at offset 80H on the virtual-APIC page) and causes a TPR-shadow update (see Section 21.5.3.3).

  The following items detail certain instructions that are considered to perform write accesses and how they behavior when accessing the VTPR:

  - — The ENTER instruction is considered to write to VTPR if the byte referenced by the final value of the stack pointer is at offset 80H on the APIC-access page (even though ENTER does not write to that byte if its size operand is non-zero). The instruction is followed by a TPR-shadow update.
  - — A VTPR access using the SMSW instruction stores data determined by the current CR0 contents, the CR0 guest/host mask, and the CR0 read shadow. The instruction is followed by a TPR-shadow update.

21.5.3.2 Operations with Multiple Accesses

Some operations may access multiple addresses. These operations include the execution of some instructions and the delivery of events through the IDT (including those injected with VM entry). In some cases, the Intel® 64 architecture specifies the ordering of these memory accesses. The following items describe the treatment of VTPR accesses that are part of such multi-access operations:

- Read-modify-write instructions may first perform a VTPR read access and then a VTPR write access. Both accesses complete normally (as described in Section 21.5.3.1). The instruction is followed by a TPR-shadow update (see Section 21.5.3.3).
- Some operations may perform a VTPR write access and subsequently cause a fault. This situation is treated as follows:
  - If the fault leads to a VM exit, no TPR-shadow update occurs.
  - If the fault does not lead to a VM exit, a TPR-shadow update occurs after fault delivery completes and before execution of the fault handler.
- If an operation includes a VTPR access and an access to some other field on the APIC-access page, the latter access causes an APIC-access VM exit as described in Section 21.2.

  If the operation performs a VTPR write access before the APIC-access VM exit, there is no TPR-shadow update.
- Suppose that the first iteration of a repeated string instruction (including OUTS) that accesses the APIC-access page performs a VTPR read access and that the next iteration would read from the APIC-access page using an offset other than 80H. The following items describe the behavior of the logical processor:
  - The iteration that performs the VTPR read access completes successfully, reading data from offset 80H on the virtual-APIC page.
  - The iteration that would read from the other offset causes an APIC-access VM exit. The instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers are such that iteration would be repeated if the instruction were restarted.
- Suppose that the first iteration of a repeated string instruction (including INS) that accesses the APIC-access page performs a VTPR write access and that the next iteration would write to the APIC-access page using an offset other than 80H. The following items describe the behavior of the logical processor:
  - The iteration that performs the VTPR write access writes data to offset 80H on the virtual-APIC page. The write is followed by a TPR-shadow update, which may cause a VM exit (see Section 21.5.3.3).
  - If the TPR-shadow update does cause a VM exit, the instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers are such that the next iteration would be performed if the instruction were restarted.
  - If the TPR-shadow update does not cause a VM exit, the iteration that would write to the other offset causes an APIC-access VM exit. The instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers are such that that iteration would be repeated if the instruction were restarted.
- Suppose that the last iteration of a repeated string instruction (including INS) performs a VTPR write access. The iteration writes data to offset 80H on the

virtual-APIC page. The write is followed by a TPR-shadow update, which may cause a VM exit (see Section 21.5.3.3). If it does, the instruction pointer saved in the VMCS references the instruction after the string instruction and the values of the general-purpose registers reflect completion of the string instruction.

21.5.3.3 TPR-Shadow Updates

If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1, a logical processor performs certain actions after any operation (or iteration of a repeated string instruction) with a VTPR write access. These actions are called a **TPR-shadow update**. (As noted in Section 21.5.3.2, a TPR-shadow update does not occur following an access that causes a VM exit.)

A TPR-shadow update includes the following actions:

1. Bits 31:8 at offset 80H on the virtual-APIC page are cleared.
2. If the value of bits 3:0 of the TPR threshold VM-execution control field is greater than the value of bits 7:4 at offset 80H on the virtual-APIC page, a VM exit will occur.

TPR-shadow updates take priority over system-management interrupts (SMIs), INIT signals, and lower priority events. A TPR-shadow update thus has priority over any debug exceptions that may have been triggered by the operation causing the TPR-shadow update. TPR-shadow updates (and any VM exits they cause) are not blocked if RFLAGS.IF = 0 or by the MOV SS, POP SS, or STI instructions.

21.6 OTHER CHANGES IN VMX NON-ROOT OPERATION

Treatments of event blocking and of task switches differ in VMX non-root operation as described in the following sections.

21.6.1 Event Blocking

Event blocking is modified in VMX non-root operation as follows:

- If the “external-interrupt exiting” VM-execution control is 1, RFLAGS.IF does not control the blocking of external interrupts. In this case, an external interrupt that is not blocked for other reasons causes a VM exit (even if RFLAGS.IF = 0).
- If the “external-interrupt exiting” VM-execution control is 1, external interrupts may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).
- If the “NMI exiting” VM-execution control is 1, non-maskable interrupts (NMIs) may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).

21.6.2 Treatment of Task Switches

Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. However, the following checks are performed (in the order indicated), possibly resulting in a fault, before there is any possibility of a VM exit due to task switch:

1. If a task gate is being used, appropriate checks are made on its P bit and on the proper values of the relevant privilege fields. The following cases detail the privilege checks performed:
    a. If CALL, INT *n*, or JMP accesses a task gate in IA-32e mode, a general-protection exception occurs.
    b. If CALL, INT *n*, INT3, INTO, or JMP accesses a task gate outside IA-32e mode, privilege-levels checks are performed on the task gate but, if they pass, privilege levels are not checked on the referenced task-state segment (TSS) descriptor.
    c. If CALL or JMP accesses a TSS descriptor directly in IA-32e mode, a general-protection exception occurs.
    d. If CALL or JMP accesses a TSS descriptor directly outside IA-32e mode, privilege levels are checked on the TSS descriptor.
    e. If a non-maskable interrupt (NMI), an exception, or an external interrupt accesses a task gate in the IDT in IA-32e mode, a general-protection exception occurs.
    f. If a non-maskable interrupt (NMI), an exception other than breakpoint exceptions (#BP) and overflow exceptions (#OF), or an external interrupt accesses a task gate in the IDT outside IA-32e mode, no privilege checks are performed.
    g. If IRET is executed with RFLAGS.NT = 1 in IA-32e mode, a general-protection exception occurs.
    h. If IRET is executed with RFLAGS.NT = 1 outside IA-32e mode, a TSS descriptor is accessed directly and no privilege checks are made.
2. Checks are made on the new TSS selector (for example, that is within GDT limits).
3. The new TSS descriptor is read. (A page fault results if a relevant GDT page is not present).
4. The TSS descriptor is checked for proper values of type (depends on type of task switch), P bit, S bit, and limit.

Only if checks 1–4 all pass (do not generate faults) might a VM exit occur. However, the ordering between a VM exit due to a task switch and a page fault resulting from accessing the old TSS or the new TSS is implementation-specific. Some logical processors may generate a page fault (instead of a VM exit due to a task switch) if accessing either TSS would cause a page fault. Other logical processors may generate a VM exit due to a task switch even if accessing either TSS would cause a page fault.

If an attempt at a task switch through a task gate in the IDT causes an exception (before generating a VM exit due to the task switch) and that exception causes a VM exit, information about the event whose delivery that accessed the task gate is recorded in the IDT-vectoring information fields and information about the exception that caused the VM exit is recorded in the VM-exit interruption-information fields. See Section 23.2. The fact that a task gate was being accessed is not recorded in the VMCS.

If an attempt at a task switch through a task gate in the IDT causes VM exit due to the task switch, information about the event whose delivery accessed the task gate is recorded in the IDT-vectoring fields of the VMCS. Since the cause of such a VM exit is a task switch and not an interruption, the valid bit for the VM-exit interruption information field is 0. See Section 23.2.

21.7 FEATURES SPECIFIC TO VMX NON-ROOT OPERATION

Some VM-execution controls cause VM exits using features that are specific to VMX non-root operation. These are the VMX-preemption timer (Section 21.7.1) and the monitor trap flag (Section 21.7.2).

21.7.1 VMX-Preemption Timer

If the last VM entry was performed with the 1-setting of “activate VMX-preemption timer” VM-execution control, the **VMX-preemption timer** counts down (from the value loaded by VM entry; see Section 22.6.4) in VMX non-root operation. When the timer counts down to zero, it stops counting down and a VM exit occurs (see Section 21.3).

The VMX-preemption timer counts down at rate proportional to that of the timestamp counter (TSC). Specifically, the timer counts down by 1 every time bit X in the TSC changes due to a TSC increment. The value of X is in the range 0–31 and can be determined by consulting the VMX capability MSR IA32_VMX_MISC (see Appendix G.6).

The VMX-preemption timer operates only in the C-states C0, C1, and C2. If the timer counts down to zero in C1 or C2, the logical processor transitions to the C0 C-state and causes a VM exit. Outside the C-states C0, C1, and C2, the timer is not decremented and does not cause VM exits.

Treatment of the timer in the case of system management interrupts (SMIs) and system-management mode (SMM) depends on whether the treatment of SMIs and SMM:

- If the default treatment of SMIs and SMM (see Section 26.14) is active, the VMX-preemption timer counts across an SMI to VMX non-root operation, subsequent execution in SMM, and the return from SMM via the RSM instruction. However, the timer can cause a VM exit only from VMX non-root operation. If the timer expires during SMI, in SMM, or during RSM, a timer-induced VM exit occurs immediately after RSM with its normal priority unless it is blocked based on activity state (Section 21.3).

- If the dual-monitor treatment of SMIs and SMM (see Section 26.15) is active, transitions into and out of SMM are VM exits and VM entries, respectively. The treatment of the VMX-preemption timer by those transitions is mostly the same as for ordinary VM exits and VM entries; Section 26.15.2 and Section 26.15.4 detail some differences.

21.7.2 Monitor Trap Flag

The **monitor trap flag** is a debugging feature that causes VM exits to occur on certain instruction boundaries in VMX non-root operation. Such VM exits are called **MTF VM exits**. An MTF VM exit may occur on an instruction boundary in VMX non-root operation as follows:

- If the “monitor trap flag” VM-execution control is 1 and VM entry is injecting a vectored event (see Section 22.5.1), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry.
- If VM entry is injecting a pending MTF VM exit (see Section 22.5.2), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0.
- If the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and a pending event (e.g., debug exception or interrupt) is delivered before an instruction can execute, an MTF VM exit is pending on the instruction boundary following delivery of the event (or any nested exception).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is a REP-prefixed string instruction:
  - — If the first iteration of the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).
  - — If the first iteration of the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary after that iteration.
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is **not** a REP-prefixed string instruction:
  - — If the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).[1]

— If the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary following execution of that instruction. If the instruction is INT3 or INTO, this boundary follows delivery of any software exception. If the instruction is INT *n*, this boundary follows delivery of a software interrupt. If the instruction is HLT, the MTF VM exit will be from the HLT activity state.

No MTF VM exit occurs if another VM exit occurs before reaching the instruction boundary on which an MTF VM exit would be pending (e.g., due to an exception or triple fault).

An MTF VM exit occurs on the instruction boundary on which it is pending unless a higher priority event takes precedence or the MTF VM exit is blocked due to the activity state:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over MTF VM exits. MTF VM exits take priority over debug-trap exceptions and lower priority events.
- No MTF VM exit occurs if the processor is in either the shutdown activity state or wait-for-SIPI activity state. If a non-maskable interrupt subsequently takes the logical processor out of the shutdown activity state without causing a VM exit, an MTF VM exit is pending after delivery of that interrupt.

21.7.3 Translation of Guest-Physical Addresses Using EPT

The extended page-table mechanism (EPT) is a feature that can be used to support the virtualization of physical memory. When EPT is in use, certain physical addresses are treated as guest-physical addresses and are not used to access memory directly. Instead, guest-physical addresses are translated by traversing a set of EPT paging structures to produce physical addresses that are used to access memory.

Details of the EPT are given in Chapter 25.

[1] This item includes the cases of an invalid opcode exception—#UD— generated by the UD2 instruction and a BOUND-range exceeded exception—#BR—generated by the BOUND instruction.

CHAPTER 22
VM ENTRIES

Software can enter VMX non-root operation using either of the VM-entry instructions VMLAUNCH and VMRESUME. VMLAUNCH can be used only with a VMCS whose launch state is clear and VMRESUME can be used only with a VMCS whose the launch state is launched. VMLAUNCH should be used for the first VM entry after VMCLEAR; VMRESUME should be used for subsequent VM entries with the same VMCS.

Each VM entry performs the following steps in the order indicated:

1. Basic checks are performed to ensure that VM entry can commence (Section 22.1).
2. The control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation and that the VMCS is correctly configured to support the next VM exit (Section 22.2).
3. The following may be performed in parallel or in any order (Section 22.3):
   - The guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures.
   - Processor state is loaded from the guest-state area and based on controls in the VMCS.
   - Address-range monitoring is cleared.
4. MSRs are loaded from the VM-entry MSR-load area (Section 22.4).
5. If VMLAUNCH is being executed, the launch state of the VMCS is set to “launched.”
6. An event may be injected in the guest context (Section 22.5).

Steps 1–4 above perform checks that may cause VM entry to fail. Such failures occur in one of the following three ways:

- Some of the checks in Section 22.1 may generate ordinary faults (for example, an invalid-opcode exception). Such faults are delivered normally.
- Some of the checks in Section 22.1 and all the checks in Section 22.2 cause control to pass to the instruction following the VM-entry instruction. The failure is indicated by setting RFLAGS.ZF[1] (if there is a current VMCS) or RFLAGS.CF (if there is no current VMCS). If there is a current VMCS, an error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 5

[1] This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B* for the error numbers.

- The checks in Section 22.3 and Section 22.4 cause processor state to be loaded from the host-state area of the VMCS (as would be done on a VM exit). Information about the failure is stored in the VM-exit information fields. See Section 22.7 for details.

EFLAGS.TF = 1 causes a VM-entry instruction to generate a single-step debug exception only if failure of one of the checks in Section 22.1 and Section 22.2 causes control to pass to the following instruction. A VM-entry does not generate a single-step debug exception in any of the following cases: (1) the instruction generates a fault; (2) failure of one of the checks in Section 22.3 or in loading MSRs causes processor state to be loaded from the host-state area of the VMCS; or (3) the instruction passes all checks in Section 22.1, Section 22.2, and Section 22.3 and there is no failure in loading MSRs.

Section 26.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, code running in SMM returns using VM entries instead of the RSM instruction. A VM entry **returns from SMM** if it is executed in SMM and the “entry to SMM” VM-entry control is 0. VM entries that return from SMM differ from ordinary VM entries in ways that are detailed in Section 26.15.4.

22.1 BASIC VM-ENTRY CHECKS

Before a VM entry commences, the current state of the logical processor is checked in the following order:

1. If the logical processor is in virtual-8086 mode or compatibility mode, an invalid-opcode exception is generated.
2. If the current privilege level (CPL) is not zero, a general-protection exception is generated.
3. If there is no current VMCS, RFLAGS.CF is set to 1 and control passes to the next instruction.
4. If there is a current VMCS, the following conditions are evaluated in order; any of these cause VM entry to fail:

   a. if there is MOV-SS blocking (see Table 20-3)

   b. if the VM entry is invoked by VMLAUNCH and the VMCS launch state is not clear

   c. if the VM entry is invoked by VMRESUME and the VMCS launch state is not launched

   If any of these checks fail, RFLAGS.ZF is set to 1 and control passes to the next instruction. An error number indicating the cause of the failure is stored in the

VM-instruction error field. See Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B* for the error numbers.

22.2 CHECKS ON VMX CONTROLS AND HOST-STATE AREA

If the checks in Section 22.1 do not cause VM entry to fail, the control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation, that the VMCS is correctly configured to support the next VM exit, and that, after the next VM exit, the processor's state is consistent with the Intel 64 and IA-32 architectures.

VM entry fails if any of these checks fail. When such failures occur, control is passed to the next instruction, RFLAGS.ZF is set to 1 to indicate the failure, and the VM-instruction error field is loaded with an error number that indicates whether the failure was due to the controls or the host-state area (see Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*).

These checks may be performed in any order. Thus, an indication by error number of one cause (for example, host state) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same VMCS.

The checks on the controls and the host-state area are presented in Section 22.2.1 through Section 22.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

22.2.1 Checks on VMX Controls

This section identifies VM-entry checks on the VMX control fields.

22.2.1.1 VM-Execution Control Fields

VM entries perform the following checks on the VM-execution control fields:[1]

- Reserved bits in the pin-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix G.3.1).
- Reserved bits in the primary processor-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix G.3.2).
- If the “activate secondary controls” primary processor-based VM-execution control is 1, reserved bits in the secondary processor-based VM-execution

[1] If the “activate secondary controls” primary processor-based VM-execution control is 0, VM entry operates as if each secondary processor-based VM-execution control were 0.

controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix G.3.3).

If the "activate secondary controls" primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary processor-based VM-execution controls. The logical processor operates as if all the secondary processor-based VM-execution controls were 0.

- The CR3-target count must not be greater than 4. Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC to determine the number of values supported (see Appendix G.6).
- If the "use I/O bitmaps" VM-execution control is 1, bits 11:0 of each I/O-bitmap address must be 0. On processors that support Intel 64 architecture, neither address should set any bits beyond the processor's physical-address width.[1] On processors that do not support Intel 64 architecture, neither address should set any bits in the range 63:32.
- If the "use TPR shadow" VM-execution control is 1, the virtual-APIC address must satisfy the following checks:
  - — Bits 11:0 of the address must be 0.
  - — On processors that support Intel 64 architecture, the address should not set any bits beyond the processor's physical-address width.
  - — On processors that support the IA-32 architecture, the address should not set any bits in the range 63:32.

  The following items describe the treatment of bytes 81H-83H on the virtual-APIC page (see Section 20.6.8) if all of the above checks are satisfied and the "use TPR shadow" VM-execution control is 1, treatment depends upon the setting of the "virtualize APIC accesses" VM-execution control:[2]

  - — If the "virtualize APIC accesses" VM-execution control is 0, the bytes may be cleared. (If the bytes are not cleared, they are left unmodified.)
  - — If the "virtualize APIC accesses" VM-execution control is 1, the bytes are cleared.
  - — Any clearing of the bytes occurs even if the VM entry subsequently fails.
- If the "use TPR shadow" VM-execution control is 1, bits 31:4 of the TPR threshold VM-execution control field must be 0.

[1] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

[2] Note that "virtualize APIC accesses" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "virtualize APIC accesses" VM-execution control were 0. See Section 20.6.2.

- The following check is performed if the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” VM-execution control is 0: the value of bits 3:0 of the TPR threshold VM-execution control field should not be greater than the value of bits 7:4 in byte 80H on the virtual-APIC page (see Section 20.6.8).
- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” VM-execution control must be 0.
- If the “virtual NMIs” VM-execution control is 0, the “NMI-window exiting” VM-execution control must be 0.
- If the “virtualize APIC-accesses” VM-execution control is 1, the APIC-access address must satisfy the following checks:
  - Bits 11:0 of the address must be 0.
  - On processors that support Intel 64 architecture, the address should not set any bits beyond the processor's physical-address width.
  - On processors that support the IA-32 architecture, the address should not set any bits in the range 63:32.
- If the “virtualize x2APIC mode” VM-execution control is 1, the “use TPR shadow” VM-execution control must be 1 and the “virtualize APIC accesses” VM-execution control must be 0.[1]
- If the “enable VPID” VM-execution control is 1, the value of the VPID VM-execution control field must not be 0000H.
- If the “enable EPT” VM-execution control is 1, the EPTP VM-execution control field (see Table 20-8 in Section 20.6.11) must satisfy the following checks:[2]
  - The EPT memory type (bits 2:0) must be a value supported by the logical processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix G.10).
  - The EPT page-walk length (bits 5:3; see Section 25.2) must be 3.
  - Reserved bits 11:6 and 63:N (where N is the processor’s physical-address width) must all be 0.

22.2.1.2 VM-Exit Control Fields

VM entries perform the following checks on the VM-exit control fields.

[1] Note that “virtualize APIC accesses” and “virtualize x2APIC mode” are both secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if both theses controls were 0. See Section 20.6.2.

[2] Note that “enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “enable EPT” VM-execution control were 0. See Section 20.6.2.

- Reserved bits in the VM-exit controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix G.4).
- If "activate VMX-preemption timer" VM-execution control is 0, the "save VMX-preemption timer value" VM-exit control must also be 0.
- The following checks are performed for the VM-exit MSR-store address if the VM-exit MSR-store count field is non-zero:
  - The lower 4 bits of the VM-exit MSR-store address must be 0. On processors that support Intel 64 architecture, the address should not set any bits beyond the processor's physical-address width.[1] On processors that do not support Intel 64 architecture, the address should not set any bits in the range 63:32.
  - On processors that support Intel 64 architecture, the address of the last byte in the VM-exit MSR-store area should not set any bits beyond the processor's physical-address width. On processors that do not support Intel 64 architecture, the address of the last byte in the VM-exit MSR-store area should not set any bits in the range 63:32. The address of this last byte is VM-exit MSR-store address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)
- The following checks are performed for the VM-exit MSR-load address if the VM-exit MSR-load count field is non-zero:
  - The lower 4 bits of the VM-exit MSR-load address must be 0. On processors that support Intel 64 architecture, the address should not set any bits beyond the processor's physical-address width. On processors that do not support Intel 64 architecture, the address should not set any bits in the range 63:32.
  - On processors that support Intel 64 architecture, the address of the last byte in the VM-exit MSR-load area should not set any bits beyond the processor's physical-address width. On processors that do not support Intel 64 architecture, the address of the last byte in the VM-exit MSR-load area should not set any bits in the range 63:32. The address of this last byte is VM-exit MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)

22.2.1.3 VM-Entry Control Fields

VM entries perform the following checks on the VM-entry control fields.

- Reserved bits in the VM-entry controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix G.5).
- Fields relevant to VM-entry event injection must be set properly. These fields are the VM-entry interruption-information field (see Table 20-12 in Section 20.8.3), the VM-entry exception error code, and the VM-entry instruction length. If the

[1] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

valid bit (bit 31) in the VM-entry interruption-information field is 1, the following must hold:

- — The field's interruption type (bits 10:8) is not set to a reserved value. Value 1 is reserved on all logical processors; value 7 (other event) is reserved on logical processors that do not support the 1-setting of the "monitor trap flag" VM-execution control.
- — The field's vector (bits 7:0) is consistent with the interruption type:
    - If the interruption type is non-maskable interrupt (NMI), the vector is 2.
    - If the interruption type is hardware exception, the vector is at most 31.
    - If the interruption type is other event, the vector is 0 (pending MTF VM exit).
- — The field's deliver-error-code bit (bit 11) is 1 if and only if the interruption type is hardware exception and the vector indicates an exception that would normally deliver an error code (8 = #DF; 10 = TS; 11 = #NP; 12 = #SS; 13 = #GP; 14 = #PF; or 17 = #AC).
- — Reserved bits in the field (30:12) are 0.
- — If the deliver-error-code bit (bit 11) is 1, bits 31:15 of the VM-entry exception error-code field are 0.
- — If the interruption type is software interrupt, software exception, or privileged software exception, the VM-entry instruction-length field is in the range 1–15.

- The following checks are performed for the VM-entry MSR-load address if the VM-entry MSR-load count field is non-zero:
    - — The lower 4 bits of the VM-entry MSR-load address must be 0. On processors that support Intel 64 architecture, the address should not set any bits beyond the processor's physical-address width.[1] On processors that do not support Intel 64 architecture, the address should not set any bits in the range 63:32.
    - — On processors that support Intel 64 architecture, the address of the last byte in the VM-entry MSR-load area should not set any bits beyond the processor's physical-address width. On processors that do not support Intel 64 architecture, the address of the last byte in the VM-entry MSR-load area should not set any bits in the range 63:32. The address of this last byte is VM-entry MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)
- If the processor is not in SMM, the "entry to SMM" and "deactivate dual-monitor treatment" VM-entry controls must be 0.
- The "entry to SMM" and "deactivate dual-monitor treatment" VM-entry controls cannot both be 1.

[1] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

22.2.2 Checks on Host Control Registers and MSRs

The following checks are performed on fields in the host-state area that correspond to control registers and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 19.8).[1]
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 19.8).
- On processors that support Intel 64 architecture, the CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width must be 0.[2]
- On processors that support Intel 64 architecture, the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
- If the "load IA32_PERF_GLOBAL_CTRL" VM-exit control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 18-14).
- If the "load IA32_PAT" VM-exit control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the "load IA32_EFER" VM-exit control is 1, bits reserved in the IA32_EFER MSR must be 0 in the field for that register. In addition, the values of the LMA and LME bits in the field must each be that of the "host address-space size" VM-exit control.

22.2.3 Checks on Host Segment and Descriptor-Table Registers

The following checks are performed on fields in the host-state area that correspond to segment and descriptor-table registers:

- In the selector field for each of CS, SS, DS, ES, FS, GS and TR, the RPL (bits 1:0) and the TI flag (bit 2) must be 0.
- The selector fields for CS and TR cannot be 0000H.
- The selector field for SS cannot be 0000H if the "host address-space size" VM-exit control is 0.
- On processors that support Intel 64 architecture, the base-address fields for FS, GS, GDTR, IDTR, and TR must contain canonical addresses.

[1] The bits corresponding to NW (bit 29) and CD (bit 30) are never checked because the values of these bits are not changed by VM exit; see Section 23.5.1.

[2] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

22.2.4 Checks Related to Address-Space Size

On processors that support Intel 64 architecture, the following checks related to address-space size are performed on VMX controls and fields in the host-state area:

- If the logical processor is outside IA-32e mode (if IA32_EFER.LMA = 0) at the time of VM entry, the following must hold:
  - — The “IA-32e mode guest” VM-entry control is 0.
  - — The “host address-space size” VM-exit control is 0.
- If the logical processor is in IA-32e mode (if IA32_EFER.LMA = 1) at the time of VM entry, the “host address-space size” VM-exit control must be 1.
- If the “host address-space size” VM-exit control is 0, the following must hold:
  - — The “IA-32e mode guest” VM-entry control is 0.
  - — Bits 63:32 in the RIP field is 0.
- If the “host address-space size” VM-exit control is 1, the following must hold:
  - — Bit 5 of the CR4 field (corresponding to CR4.PAE) is 1.
  - — The RIP field contains a canonical address.

On processors that do not support Intel 64 architecture, checks are performed to ensure that the “IA-32e mode guest” VM-entry control and the “host address-space size” VM-exit control are both 0.

22.3 CHECKING AND LOADING GUEST STATE

If all checks on the VMX controls and the host-state area pass (see Section 22.2), the following operations take place concurrently: (1) the guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures; (2) processor state is loaded from the guest-state area or as specified by the VM-entry control fields; and (3) address-range monitoring is cleared.

Because the checking and the loading occur concurrently, a failure may be discovered only after some state has been loaded. For this reason, the logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 22.7.

22.3.1 Checks on the Guest State Area

This section describes checks performed on fields in the guest-state area. These checks may be performed in any order. The following subsections reference fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

22.3.1.1 Checks on Guest Control Registers, Debug Registers, and MSRs

The following checks are performed on fields in the guest-state area corresponding to control registers, debug registers, and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 19.8).[1]
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 19.8).
- If the “load debug controls” VM-entry control is 1, bits reserved in the IA32_DEBUGCTL MSR must be 0 in the field for that register. Note that the first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally.
- The following checks are performed on processors that support Intel 64 architecture:
  - If the “IA-32e mode guest” VM-entry control is 1, bit 5 in the CR4 field (corresponding to CR4.PAE) must be 1.
  - The CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor’s physical-address width are 0.[2]
  - If the “load debug controls” VM-entry control is 1, bits 63:32 in the DR7 field must be 0. Note that the first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally (if they supported Intel 64 architecture).
  - The IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 18-14).
- If the “load IA32_PAT” VM-entry control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the “load IA32_EFER” VM-entry control is 1, bits reserved in the IA32_EFER MSR must be 0 in the field for that register. In addition, the values of the LMA and LME bits in the field must each be that of the “IA-32e mode guest” VM-exit control.

[1] The bits corresponding to NW (bit 29) and CD (bit 30) are never checked because the values of these bits are not changed by VM entry; see Section 22.3.2.1.

[2] Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

22.3.1.2 Checks on Guest Segment Registers

This section specifies the checks on the fields for CS, SS, DS, ES, FS, GS, TR, and LDTR. The following terms are used in defining these checks:

- The guest will be **virtual-8086** if the VM flag (bit 17) is 1 in the RFLAGS field in the guest-state area.
- The guest will be **IA-32e mode** if the "IA-32e mode guest" VM-entry control is 1. (This is possible only on processors that support Intel 64 architecture.)
- Any one of these registers is said to be **usable** if the unusable bit (bit 16) is 0 in the access-rights field for that register.

The following are the checks on these fields:

- Selector fields.
  - — TR. The TI flag (bit 2) must be 0.
  - — LDTR. If LDTR is usable, the TI flag (bit 2) must be 0.
  - — SS. If the guest will not be virtual-8086, the RPL (bits 1:0) must equal the RPL of the selector field for CS.
- Base-address fields.
  - — CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the address must be the selector field shifted left 4 bits (multiplied by 16).
  - — The following checks are performed on processors that support Intel 64 architecture:
    - TR, FS, GS. The address must be canonical.
    - LDTR. If LDTR is usable, the address must be canonical.
    - CS. Bits 63:32 of the address must be zero.
    - SS, DS, ES. If the register is usable, bits 63:32 of the address must be zero.
- Limit fields for CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the field must be 0000FFFFH.
- Access-rights fields.
  - — CS, SS, DS, ES, FS, GS.
    - If the guest will be virtual-8086, the field must be 000000F3H. Note that this implies the following:
      - — Bits 3:0 (Type) must be 3, indicating an expand-up read/write accessed data segment.
      - — Bit 4 (S) must be 1.
      - — Bits 6:5 (DPL) must be 3.
      - — Bit 7 (P) must be 1.

  - — Bits 11:8 (reserved), bit 12 (software available), bit 13 (reserved/L), bit 14 (D/B), bit 15 (G), bit 16 (unusable), and bits 31:17 (reserved) must all be 0.

- If the guest will not be virtual-8086, the different sub-fields are considered separately:
  - — Bits 3:0 (Type).
    - CS. Bit 0 of the Type must be 1 (accessed) and bit 3 of the Type must be 1 (code segment).
    - SS. If SS is usable, the Type must be 3 or 7 (read/write, accessed data segment).
    - DS, ES, FS, GS. The following checks apply if the register is usable:
      - — Bit 0 of the Type must be 1 (accessed).
      - — If bit 3 of the Type is 1 (code segment), then bit 1 of the Type must be 1 (readable).
  - — Bit 4 (S). If the register is CS or if the register is usable, S must be 1.
  - — Bits 6:5 (DPL).
    - CS.
      - — If the Type is in the range 8–11 (non-conforming code segment), the DPL must equal the RPL (bits 1:0) from the selector field.
      - — If the Type is in the range 12–15 (conforming code segment), the DPL cannot be greater than the RPL from the selector field.
    - SS. The DPL must equal the RPL from the selector field
    - DS, ES, FS, GS. If the register is usable and the register's Type is in the range 0 – 11 (data segment or non-conforming code segment), then the DPL cannot be less than the RPL from the selector field.
  - — Bit 7 (P). If the register is CS or if the register is usable, P must be 1.
  - — Bits 11:8 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
  - — Bit 14 (D/B). For CS, D/B must be 0 if the guest will be IA-32e mode and the L bit (bit 13) in the access-rights field is 1.
  - — Bit 15 (G). The following checks apply if the register is CS or if the register is usable:
    - If any bit in the limit field in the range 11:0 is 0, G must be 0.

        - If any bit in the limit field in the range 31:20 is 1, G must be 1.
    - Bits 31:17 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
- TR. The different sub-fields are considered separately:
    - Bits 3:0 (Type).
        - If the guest will not be IA-32e mode, the Type must be 3 (16-bit busy TSS) or 11 (32-bit busy TSS).
        - If the guest will be IA-32e mode, the Type must be 11 (64-bit busy TSS).
    - Bit 4 (S). S must be 0.
    - Bit 7 (P). P must be 1.
    - Bits 11:8 (reserved). These bits must all be 0.
    - Bit 15 (G).
        - If any bit in the limit field in the range 11:0 is 0, G must be 0.
        - If any bit in the limit field in the range 31:20 is 1, G must be 1.
    - Bit 16 (Unusable). The unusable bit must be 0.
    - Bits 31:17 (reserved). These bits must all be 0.
- LDTR. The following checks on the different sub-fields apply only if LDTR is usable:
    - Bits 3:0 (Type). The Type must be 2 (LDT).
    - Bit 4 (S). S must be 0.
    - Bit 7 (P). P must be 1.
    - Bits 11:8 (reserved). These bits must all be 0.
    - Bit 15 (G).
        - If any bit in the limit field in the range 11:0 is 0, G must be 0.
        - If any bit in the limit field in the range 31:20 is 1, G must be 1.
    - Bits 31:17 (reserved). These bits must all be 0.

22.3.1.3 Checks on Guest Descriptor-Table Registers

The following checks are performed on the fields for GDTR and IDTR:

- On processors that support Intel 64 architecture, the base-address fields must contain canonical addresses.
- Bits 31:16 of each limit field must be 0.

22.3.1.4 Checks on Guest RIP and RFLAGS

The following checks are performed on fields in the guest-state area corresponding to RIP and RFLAGS:

- RIP. The following checks are performed on processors that support Intel 64 architecture:
  - Bits 63:32 must be 0 if the “IA-32e mode guest” VM-entry control is 0 or if the L bit (bit 13) in the access-rights field for CS is 0.
  - If the processor supports N < 64 linear-address bits, bits 63:N must be identical if the “IA-32e mode guest” VM-entry control is 1 and the L bit in the access-rights field for CS is 1.[1] (No check applies if the processor supports 64 linear-address bits.)
- RFLAGS.
  - Reserved bits 63:22 (bits 31:22 on processors that do not support Intel 64 architecture), bit 15, bit 5 and bit 3 must be 0 in the field, and reserved bit 1 must be 1.
  - On processors that support Intel 64 architecture, the VM flag (bit 17) must be 0 if the “IA-32e mode guest” VM-entry control is 1.
  - The IF flag (RFLAGS[bit 9]) must be 1 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) is external interrupt.

22.3.1.5 Checks on Guest Non-Register State

The following checks are performed on fields in the guest-state area corresponding to non-register state:

- Activity state.
  - The activity-state field must contain a value in the range 0 – 3, indicating an activity state supported by the implementation (see Section 20.4.2). Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix G.6) to determine what activity states are supported.
  - The activity-state field must not indicate the HLT state if the DPL (bits 6:5) in the access-rights field for SS is not 0.[2]
  - The activity-state field must indicate the active state if the interruptibility-state field indicates blocking by either MOV-SS or by STI (if either bit 0 or bit 1 in that field is 1).

[1] Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

[2] As noted in Section 20.4.1, SS.DPL corresponds to the logical processor’s current privilege level (CPL).

  - If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the interruption to be delivered (as defined by interruption type and vector) must not be one that would normally be blocked while a logical processor is in the activity state corresponding to the contents of the activity-state field. The following items enumerate the interruptions (as specified in the VM-entry interruption-information field) whose injection is allowed for the different activity states:
    - Active. Any interruption is allowed.
    - HLT. The only events allowed are the following:
      - Those with interruption type external interrupt or non-maskable interrupt (NMI).
      - Those with interruption type hardware exception and vector 1 (debug exception) or vector 18 (machine-check exception).
      - Those with interruption type other event and vector 0 (pending MTF VM exit).

      See Table 20-12 in Section 20.8.3 for details regarding the format of the VM-entry interruption-information field.
    - Shutdown. Only NMIs and machine-check exceptions are allowed.
    - Wait-for-SIPI. No interruptions are allowed.
  - The activity-state field must not indicate the wait-for-SIPI state if the “entry to SMM” VM-entry control is 1.
- Interruptibility state.
  - The reserved bits (bits 31:4) must be 0.
  - The field cannot indicate blocking by both STI and MOV SS (bits 0 and 1 cannot both be 1).
  - Bit 0 (blocking by STI) must be 0 if the IF flag (bit 9) is 0 in the RFLAGS field.
  - Bit 0 (blocking by STI) and bit 1 (blocking by MOV-SS) must both be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 0, indicating external interrupt.
  - Bit 1 (blocking by MOV-SS) must be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 2, indicating non-maskable interrupt (NMI).
  - Bit 2 (blocking by SMI) must be 0 if the processor is not in SMM.
  - Bit 2 (blocking by SMI) must be 1 if the “entry to SMM” VM-entry control is 1.
  - A processor may require bit 0 (blocking by STI) to be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 2, indicating NMI. Other processors may not make this requirement.

- Bit 3 (blocking by NMI) must be 0 if the “virtual NMIs” VM-execution control is 1, the valid bit (bit 31) in the VM-entry interruption-information field is 1, and the interruption type (bits 10:8) in that field has value 2 (indicating NMI).

NOTE

If the “virtual NMIs” VM-execution control is 0, there is no requirement that bit 3 be 0 if the valid bit in the VM-entry interruption-information field is 1 and the interruption type in that field has value 2.

- Pending debug exceptions.
  - Bits 11:4, bit 13, and bits 63:15 (bits 31:15 on processors that do not support Intel 64 architecture) must be 0.
  - The following checks are performed if any of the following holds: (1) the interruptibility-state field indicates blocking by STI (bit 0 in that field is 1); (2) the interruptibility-state field indicates blocking by MOV SS (bit 1 in that field is 1); or (3) the activity-state field indicates HLT:
    - Bit 14 (BS) must be 1 if the TF flag (bit 8) in the RFLAGS field is 1 and the BTF flag (bit 1) in the IA32_DEBUGCTL field is 0.
    - Bit 14 (BS) must be 0 if the TF flag (bit 8) in the RFLAGS field is 0 or the BTF flag (bit 1) in the IA32_DEBUGCTL field is 1.
- VMCS link pointer. The following checks apply if the field contains a value other than FFFFFFFF_FFFFFFFFH:
  - Bits 11:0 must be 0.
  - On processors that support Intel 64 architecture, bits beyond the processor’s physical-address width must be 0.[1] On processors that do not support Intel 64 architecture, bits in the range 63:32 must be 0.
  - The 32 bits located in memory referenced by the value of the field (as a physical address) must contain the processor’s VMCS revision identifier (see Section 20.2).
  - If the processor is not in SMM or the “entry to SMM” VM-entry control is 1, the field must not contain the current VMCS pointer.
  - If the processor is in SMM and the “entry to SMM” VM-entry control is 0, the field must not contain the VMXON pointer.

[1] Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

22.3.1.6 Checks on Guest Page-Directory-Pointer-Table Entries

If CR0.PG =1 and CR4.PAE = 1, the logical processor uses the **physical-address extension** (PAE). If IA32_EFER.LMA = 0, the logical processor also uses **PAE paging** (see Section 3.8 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).[1] When PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTEs.

A VM entry is to a guest that uses PAE paging if (1) bit 5 (corresponding to CR4.PAE) is set in the CR4 field in the guest-state area; and (2) the "IA-32e mode guest" VM-entry control is 0. Such a VM entry checks the validity of the PDPTEs:

- If the "enable EPT" VM-execution control is 0, VM entry checks the validity of the PDPTEs referenced by the CR3 field in the guest-state area if either (1) PAE paging was not in use before the VM entry; or (2) the value of CR3 is changing as a result of the VM entry. VM entry may check their validity even if neither (1) nor (2) hold.[2]
- If the "enable EPT" VM-execution control is 1, VM entry checks the validity of the PDPTE fields in the guest-state area (see Section 20.4.2).

A VM entry to a guest that does not use PAE paging does not check the validity of any PDPTEs.

A VM entry that checks the validity of the PDPTEs uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use.[3] If MOV to CR3 would cause a general-protection exception due to the PDPTEs that would be loaded (e.g., because a reserved bit is set), the VM entry fails.

22.3.2 Loading Guest State

Processor state is updated on VM entries in the following ways:

- Some state is loaded from the guest-state area.
- Some state is determined by VM-entry controls.
- The page-directory pointers are loaded based on the values of certain control registers.

[1] On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine the number physical-address bits supported by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

[2] Note that "enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "enable EPT" VM-execution control were 0. See Section 20.6.2.

[3] This implies that (1) bits 11:9 in each PDPTE are ignored; and (2) if bit 0 (present) is clear in one of the PDPTEs, bits 63:1 of that PDPTE are ignored.

This loading may be performed in any order and in parallel with the checking of VMCS contents (see Section 22.3.1).

The loading of guest state is detailed in Section 22.3.2.1 to Section 22.3.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

In addition to the state loading described in this section, VM entries may load MSRs from the VM-entry MSR-load area (see Section 22.4). This loading occurs only after the state loading described in this section and the checking of VMCS contents described in Section 22.3.1.

22.3.2.1 Loading Guest Control Registers, Debug Registers, and MSRs

The following items describe how guest control registers, debug registers, and MSRs are loaded on VM entry:

- CR0 is loaded from the CR0 field with the exception of the following bits, which are never modified on VM entry: ET (bit 4); reserved bits 15:6, 17, and 28:19; NW (bit 29) and CD (bit 30).[1] The values of these bits in the CR0 field are ignored.
- CR3 and CR4 are loaded from the CR3 field and the CR4 field, respectively.
- If the “load debug controls” VM-execution control is 1, DR7 is loaded from the DR7 field with the exception that bit 12 and bits 15:14 are always 0 and bit 10 is always 1. The values of these bits in the DR7 field are ignored.

  Note that the first processors to support the virtual-machine extensions supported only the 1-setting of the “load debug controls” VM-execution control and thus always loaded DR7 from the DR7 field.
- The following describes how some MSRs are loaded using fields in the guest-state area:
  — If the “load debug controls” VM-execution control is 1, the IA32_DEBUGCTL MSR is loaded from the IA32_DEBUGCTL field. Note that the first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always loaded the IA32_DEBUGCTL MSR from the IA32_DEBUGCTL field.

  — The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since this field has only 32 bits, bits 63:32 of the MSR are cleared to 0.

  — The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.

  — The following are performed on processors that support Intel 64 architecture:

[1] Bits 15:6, bit 17, and bit 28:19 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. Bits 15:6, bit 17, and bit 28:19 of CR0 are always 0 and CR0.ET is always 1.

    - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 22.3.2.2).
    - If the "load IA32_EFER" VM-entry control is 0, the LMA and LME bits in the IA32_EFER MSR are each loaded with the setting of the "IA-32e mode guest" VM-entry control. See below for the case in which the "load IA32_EFER" VM-entry control is 1
  - If the "load IA32_PERF_GLOBAL_CTRL" VM-entry control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field.
  - If the "load IA32_PAT" VM-entry control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field.
  - If the "load IA32_EFER" VM-entry control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field.

  With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-entry MSR-load area. See Section 22.4.
- The SMBASE register is unmodified by all VM entries except those that return from SMM.

If the "enable VPID" VM-execution control is 0 and any of CR3[63:5] (CR3[31:5] on processors that do not support Intel 64 architecture), CR4.PAE, CR4.PSE, or IA32_EFER.LMA is changing, the TLBs and paging-structure caches are updated so that, after VM entry, the logical processor will not use any translations that were cached before the transition. This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if CR4.PAE was 1 before and after the transition).[1] If the "enable VPID" VM-execution control is 1, the logical processor may invalidate entries in the TLBs and paging-structure caches, but it is not required to do so.

22.3.2.2 Loading Guest Segment Registers and Descriptor-Table Registers

For each of CS, SS, DS, ES, FS, GS, TR, and LDTR, fields are loaded from the guest-state area as follows:

- The unusable bit is loaded from the access-rights field. This bit can never be set for TR (see Section 22.3.1.2). If it is set for one of the other registers, the following apply:
  - For each of CS, SS, DS, ES, FS, and GS, uses of the segment cause faults (general-protection exception or stack-fault exception) outside 64-bit mode, just as they would had the segment been loaded using a null selector. This bit does not cause accesses to fault in 64-bit mode.

[1] This paragraph applies only to translations of linear addresses. It does not apply to translations of guest-physical addresses using EPT. See Section 25.6 for details regarding the caching of information relevant to the translation of guest-physical addresses.

  - If this bit is set for LDTR, uses of LDTR cause general-protection exceptions in all modes, just as they would had LDTR been loaded using a null selector.

  If this bit is clear for any of CS, SS, DS, ES, FS, GS, TR, and LDTR, a null selector value does not cause a fault (general-protection exception or stack-fault exception).

- TR. The selector, base, limit, and access-rights fields are loaded.
- CS.
  - The following fields are always loaded: selector, base address, limit, and (from the access-rights field) the L, D, and G bits.
  - For the other fields, the unusable bit of the access-rights field is consulted:
    - If the unusable bit is 0, all of the access-rights fields are loaded.
    - If the unusable bit is 1, the remainder of CS access rights are undefined after VM entry.
- SS, DS, ES, FS, and GS, and LDTR.
  - The selector fields are loaded.
  - For the other fields, the unusable bit of the corresponding access-rights field is consulted:
    - If the unusable bit is 0, the base-address, limit, and access-rights fields are loaded.
    - If the unusable bit is 1, the base address, the segment limit, and the remainder of the access rights are undefined after VM entry. The only exceptions are the following:
      - SS.DPL: always loaded from the SS access-rights field. This will be the current privilege level (CPL) after the VM entry completes.
      - The base addresses for FS and GS: always loaded. Note that, on processors that support Intel 64 architecture, the values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
      - The base address for LDTR on processors that support Intel 64 architecture: set to an undefined but canonical value.
      - Bits 63:32 of the base addresses for SS, DS, and ES on processors that support Intel 64 architecture: cleared to 0.

GDTR and IDTR are loaded using the base and limit fields.

22.3.2.3 Loading Guest RIP, RSP, and RFLAGS

RSP, RIP, and RFLAGS are loaded from the RSP field, the RIP field, and the RFLAGS field, respectively. The following items regard the upper 32 bits of these fields on VM entries that are not to 64-bit mode:

- Bits 63:32 of RSP are undefined outside 64-bit mode. Thus, a logical processor may ignore the contents of bits 63:32 of the RSP field on VM entries that are not to 64-bit mode.
- As noted in Section 22.3.1.4, bits 63:32 of the RIP and RFLAGS fields must be 0 on VM entries that are not to 64-bit mode.

22.3.2.4 Loading Page-Directory-Pointer-Table Entries

As noted in Section 22.3.1.6, the logical processor uses PAE paging if bit 5 in CR4 (CR4.PAE) is 1 and IA32_EFER.LMA is 0. A VM entry to a guest that uses PAE paging loads the PDPTEs into internal, non-architectural registers based on the setting of the "enable EPT" VM-execution control:

- If the control is 0, the PDPTEs are loaded from the page-directory-pointer table referenced by the physical address in the value of CR3 being loaded by the VM entry (see Section 22.3.2.1). The values loaded are treated as physical addresses in VMX non-root operation.
- If the control is 1, the PDPTEs are loaded from corresponding fields in the guest-state area (see Section 20.4.2). The values loaded are treated as guest-physical addresses in VMX non-root operation.

22.3.3 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 7.11.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. VM entries clear any address-range monitoring that may be in effect.

22.4 LOADING MSRS

VM entries may load MSRs from the VM-entry MSR-load area (see Section 20.8.2). Specifically each entry in that area (up to the number specified in the VM-entry MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101 (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM entry did not commence in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)

- The value of bits 31:0 indicates an MSR that cannot be loaded on VM entries for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Appendix B.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.[1]

The VM entry fails if processing fails for any entry. The logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 22.7.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM entry, the logical processor will not use any translations that were cached before the transition.

22.5 EVENT INJECTION

If the valid bit in the VM-entry interruption-information field (see Section 20.8.3) is 1, VM entry causes an event to be delivered (or made pending) after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established.

- If the interruption type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception), the event is delivered as described in Section 22.5.1.
- If the interruption type in the field is 7 (other event) and the vector field is 0, an MTF VM exit is pending after VM entry. See Section 22.5.2.

22.5.1 Vectored-Event Injection

VM entry delivers an injected vectored event within the guest context established by VM entry. This means that delivery occurs after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established.[2] The event is delivered using the vector in that field to select a descriptor in the IDT. Since event injection occurs after loading IDTR from the guest-state area, this is the guest IDT.

[1] Note the following about processors that support Intel 64 architecture. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CR0.PG is always 1 in VMX operation, the IA32_EFER MSR should not be included in the VM-entry MSR-load area for the purpose of modifying the LME bit.

Section 22.5.1.1 provides details of vectored-event injection. In general, the event is delivered exactly as if it had been generated normally.

If event delivery encounters a nested exception (for example, a general-protection exception because the vector indicates a descriptor beyond the IDT limit), the exception bitmap is consulted using the vector of that exception. If the bit is 0, the exception is delivered through the IDT. If the bit is 1, a VM exit occurs. Section 22.5.1.2 details cases in which event injection causes a VM exit.

22.5.1.1 Details of Vectored-Event Injection

The event-injection process is controlled by the contents of the VM-entry interruption information field (format given in Table 20-12), the VM-entry exception error-code field, and the VM-entry instruction-length field. The following items provide details of the process:

- The value pushed on the stack for RFLAGS is generally that which was loaded from the guest-state area. The value pushed for the RF flag is not modified based on the type of event being delivered. However, the pushed value of RFLAGS may be modified if a software interrupt is being injected into a guest that will be in virtual-8086 mode (see below). After RFLAGS is pushed on the stack, the value in the RFLAGS register is modified as is done normally when delivering an event through the IDT.
- The instruction pointer that is pushed on the stack depends on the type of event and whether nested exceptions occur during its delivery. The term **current guest RIP** refers to the value to be loaded from the guest-state area. The value pushed is determined as follows:[1]
  - — If VM entry successfully injects (with no nested exception) an event with interruption type external interrupt, NMI, or hardware exception, the current guest RIP is pushed on the stack.
  - — If VM entry successfully injects (with no nested exception) an event with interruption type software interrupt, privileged software exception, or software exception, the current guest RIP is incremented by the VM-entry instruction length before being pushed on the stack.
  - — If VM entry encounters an exception while injecting an event and that exception does not cause a VM exit, the current guest RIP is pushed on the stack regardless of event type or VM-entry instruction length. If the encountered exception does cause a VM exit that saves RIP, the saved RIP is current guest RIP.

[2] This does not imply that injection of an exception or interrupt will cause a VM exit due to the settings of VM-execution control fields (such as the exception bitmap) that would cause a VM exit if the event had occurred in VMX non-root operation. In contrast, a nested exception encountered during event delivery may cause a VM exit; see Section 22.5.1.1.

[1] While these items refer to RIP, the width of the value pushed (16 bits, 32 bits, or 64 bits) is determined normally.

- If the deliver-error-code bit (bit 11) is set in the VM-entry interruption-information field, the contents of the VM-entry exception error-code field is pushed on the stack as an error code would be pushed during delivery of an exception.
- DR6, DR7, and the IA32_DEBUGCTL MSR are not modified by event injection, even if the event has vector 1 (normal deliveries of debug exceptions, which have vector 1, do update these registers).
- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode (RFLAGS.VM = 1), no general-protection exception can occur due to RFLAGS.IOPL < 3. A VM monitor should check RFLAGS.IOPL before injecting such an event and, if desired, inject a general-protection exception instead of a software interrupt.
- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode with virtual-8086 mode extensions (RFLAGS.VM = CR4.VME = 1), event delivery is subject to VME-based interrupt redirection based on the software interrupt redirection bitmap in the task-state segment (TSS) as follows:
  - — If bit *n* in the bitmap is clear (where *n* is the number of the software interrupt), the interrupt is directed to an 8086 program interrupt handler: the processor uses a 16-bit interrupt-vector table (IVT) located at linear address zero. If the value of RFLAGS.IOPL is less than 3, the following modifications are made to the value of RFLAGS that is pushed on the stack: IOPL is set to 3, and IF is set to the value of VIF.
  - — If bit *n* in the bitmap is set (where *n* is the number of the software interrupt), the interrupt is directed to a protected-mode interrupt handler. (In other words, the injection is treated as described in the next item.) In this case, the software interrupt does not invoke such a handler if RFLAGS.IOPL < 3 (a general-protection exception occurs instead). However, as noted above, RFLAGS.IOPL cannot cause an injected software interrupt to cause such a exception. Thus, in this case, the injection invokes a protected-mode interrupt handler independent of the value of RFLAGS.IOPL.

  Injection of events of other types are not subject to this redirection.
- If VM entry is injecting a software interrupt (not redirected as described above) or software exception, privilege checking is performed on the IDT descriptor being accessed as would be the case for executions of INT *n*, INT3, or INTO (the descriptor's DPL cannot be less than CPL). There is no checking of RFLAGS.IOPL, even if the guest will be in virtual-8086 mode. Failure of this check may lead to a nested exception. Injection of an event with interruption type external interrupt, NMI, hardware exception, and privileged software exception, or with interruption type software interrupt and being redirected as described above, do not perform these checks.
- If VM entry is injecting a non-maskable interrupt (NMI) and the "virtual NMIs" VM-execution control is 1, virtual-NMI blocking is in effect after VM entry.

- The transition causes a last-branch record to be logged if the LBR bit is set in the IA32_DEBUGCTL MSR. This is true even for events such as debug exceptions, which normally clear the LBR bit before delivery.
- The last-exception record MSRs (LERs) may be updated based on the setting of the LBR bit in the IA32_DEBUGCTL MSR. Events such as debug exceptions, which normally clear the LBR bit before they are delivered, and therefore do not normally update the LERs, may do so as part of VM-entry event injection.
- If injection of an event encounters a nested exception that does not itself cause a VM exit, the value of the EXT bit (bit 0) in any error code pushed on the stack is determined as follows:
  - If event being injected has interruption type external interrupt, NMI, hardware exception, or privileged software exception and encounters a nested exception (but does not produce a double fault), the error code for the first such exception encountered sets the EXT bit.
  - If event being injected is a software interrupt or an software exception and encounters a nested exception (but does not produce a double fault), the error code for the first such exception encountered clears the EXT bit.
  - If event delivery encounters a nested exception and delivery of that exception encounters another exception (but does not produce a double fault), the error code for that exception sets the EXT bit. If a double fault is produced, the error code for the double fault is 0000H (the EXT bit is clear).

22.5.1.2 VM Exits During Event Injection

An event being injected never causes a VM exit directly regardless of the settings of the VM-execution controls. For example, setting the “NMI exiting” VM-execution control to 1 does not cause a VM exit due to injection of an NMI.

However, the event-delivery process may lead to a VM exit:

- If the vector in the VM-entry interruption-information field identifies a task gate in the IDT, the attempted task switch may cause a VM exit just as it would had the injected event occurred during normal execution in VMX non-root operation (see Section 21.6.2).
- If event delivery encounters a nested exception, a VM exit may occur depending on the contents of the exception bitmap (see Section 21.3).
- If event delivery generates a double-fault exception (due to a nested exception); the logical processor encounters another nested exception while attempting to call the double-fault handler; and that exception does not cause a VM exit due to the exception bitmap; then a VM exit occurs due to triple fault (see Section 21.3).
- If event delivery injects a double-fault exception and encounters a nested exception that does not cause a VM exit due to the exception bitmap, then a VM exit occurs due to triple fault (see Section 21.3).

- If the “virtualize APIC accesses” VM-execution control is 1 and event delivery generates an access to the APIC-access page, that access may cause an APIC-access VM exit (see Section 21.2) or, if the access is a VTPR access, be treated as specified in Section 21.5.3.[1]

If the event-delivery process does cause a VM exit, the processor state before the VM exit is determined just as it would be had the injected event occurred during normal execution in VMX non-root operation. If the injected event directly accesses a task gate that cause a VM exit or if the first nested exception encountered causes a VM exit, information about the injected event is saved in the IDT-vectoring information field (see Section 23.2.3).

22.5.2 Injection of Pending MTF VM Exits

If the interruption type in the VM-entry interruption-information field is 7 (other event) and the vector field is 0, VM entry causes an MTF VM exit to be pending on the instruction boundary following VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0. See Section 21.7.2 for the treatment of pending MTF VM exits.

22.6 SPECIAL FEATURES OF VM ENTRY

This section details a variety of features of VM entry. It uses the following terminology: a VM entry is **vectoring** if the valid bit (bit 31) of the VM-entry interruption information field is 1 and the interruption type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception).

22.6.1 Interruptibility State

The interruptibility-state field in the guest-state area (see Table 20-3) contains bits that control blocking by STI, blocking by MOV SS, and blocking by NMI. This field impacts event blocking after VM entry as follows:

- If the VM entry is vectoring, there is no blocking by STI or by MOV SS following the VM entry, regardless of the contents of the interruptibility-state field.
- If the VM entry is not vectoring, the following apply:
  - Events are blocked by STI if and only if bit 0 in the interruptibility-state field is 1. Such blocking is cleared after the guest executes one instruction or

[1] Note that “virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 20.6.2.

incurs an exception (including a debug exception made pending by VM entry; see Section 22.6.3).

- — Events are blocked by MOV SS if and only if bit 1 in the interruptibility-state field is 1. This may affect the treatment of pending debug exceptions; see Section 22.6.3. Such blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry).

- The blocking of non-maskable interrupts (NMIs) is determined as follows:
  - — If the “virtual NMIs” VM-execution control is 0, NMIs are blocked if bit 3 (blocking by NMI) in the interruptibility-state field is 1. If the “NMI exiting” VM-execution control is 0, such blocking remains in effect until IRET is executed (even if the instruction generates a fault). If the “NMI exiting” control is 1, such blocking remains in effect as long as the logical processor is in VMX non-root operation.
  - — The following items describe the use of bit 3 (blocking by NMI) in the interruptibility-state field if the “virtual NMIs” VM-execution control is 1:
    - The bit’s value does not affect the blocking of NMIs after VM entry. NMIs are not blocked in VMX non-root operation (except for ordinary blocking for other reasons, such as by the MOV SS instruction, the wait-for-SIPI state, etc.)
    - The bit’s value determines whether there is virtual-NMI blocking after VM entry. If the bit is 1, virtual-NMI blocking is in effect after VM entry. If the bit is 0, there is no virtual-NMI blocking after VM entry unless the VM entry is injecting an NMI (see Section 22.5.1.1).
- Blocking of system-management interrupts (SMIs) is determined as follows:
  - — If the VM entry was not executed in system-management mode (SMM), SMI blocking is unchanged by VM entry.
  - — If the VM entry was executed in SMM, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.

22.6.2 Activity State

The activity-state field in the guest-state area controls whether, after VM entry, the logical processor is active or in one of the inactive states identified in Section 20.4.2. The use of this field is determined as follows:

- If the VM entry is vectoring, the logical processor is in the active state after VM entry. While the consistency checks described in Section 22.3.1.5 on the activity-state field do apply in this case, the contents of the activity-state field do not determine the activity state after VM entry.
- If the VM entry is not vectoring, the logical processor ends VM entry in the activity state specified in the guest-state area. If VM entry ends with the logical processor in an inactive activity state, the VM entry generates any special bus

cycle that is normally generated when that activity state is entered from the active state. If VM entry would end with the logical processor in the shutdown state and the logical processor is in SMX operation,[1] an Intel® TXT shutdown condition occurs. The error code used is 0000H, indicating "legacy shutdown." See *Intel® Trusted Execution Technology Preliminary Architecture Specification*.

- Some activity states unconditionally block certain events. The following blocking is in effect after any VM entry that puts the processor in the indicated state:
  - The active state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the active state and in VMX non-root operation are discarded and do not cause VM exits.
  - The HLT state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the HLT state and in VMX non-root operation are discarded and do not cause VM exits.
  - The shutdown state blocks external interrupts and SIPIs. External interrupts that arrive while a logical processor is in the shutdown state and in VMX non-root operation do not cause VM exits even if the "external-interrupt exiting" VM-execution control is 1. SIPIs that arrive while a logical processor is in the shutdown state and in VMX non-root operation are discarded and do not cause VM exits.
  - The wait-for-SIPI state blocks external interrupts, non-maskable interrupts (NMIs), INIT signals, and system-management interrupts (SMIs). Such events do not cause VM exits if they arrive while a logical processor is in the wait-for-SIPI state and in VMX non-root operation do not cause VM exits regardless of the settings of the pin-based VM-execution controls.

22.6.3 Delivery of Pending Debug Exceptions after VM Entry

The pending debug exceptions field in the guest-state area indicates whether there are debug exceptions that have not yet been delivered (see Section 20.4.2). This section describes how these are treated on VM entry.

There are no pending debug exceptions after VM entry if any of the following are true:

- The VM entry is vectoring with one of the following interruption types: external interrupt, non-maskable interrupt (NMI), hardware exception, or privileged software exception.
- The interruptibility-state field does not indicate blocking by MOV SS and the VM entry is vectoring with either of the following interruption type: software interrupt or software exception.

[1] A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

- The VM entry is not vectoring and the activity-state field indicates either shutdown or wait-for-SIPI.

If none of the above hold, the pending debug exceptions field specifies the debug exceptions that are pending for the guest. There are **valid pending debug exceptions** if either the BS bit (bit 14) or the enable-breakpoint bit (bit 12) is 1. If there are valid pending debug exceptions, they are handled as follows:

- If the VM entry is not vectoring, the pending debug exceptions are treated as they would had they been encountered normally in guest execution:
  - If the logical processor is not blocking such exceptions (the interruptibility-state field indicates no blocking by MOV SS), a debug exception is delivered after VM entry (see below).
  - If the logical processor is blocking such exceptions (due to blocking by MOV SS), the pending debug exceptions are held pending or lost as would normally be the case.
- If the VM entry is vectoring (with interruption type software interrupt or software exception and with blocking by MOV SS), the following items apply:
  - For injection of a software interrupt or of a software exception with vector 3 (#BP) or vector 4 (#OF), the pending debug exceptions are treated as they would had they been encountered normally in guest execution if the corresponding instruction (INT3 or INTO) were executed after a MOV SS that encountered a debug trap.
  - For injection of a software exception with a vector other than 3 and 4, the pending debug exceptions may be lost or they may be delivered after injection (see below).

If there are no valid pending debug exceptions (as defined above), no pending debug exceptions are delivered after VM entry.

If a pending debug exception is delivered after VM entry, it has the priority of “traps on the previous instruction” (see Section 5.9 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*). Thus, INIT signals and system-management interrupts (SMIs) take priority of such an exception, as do VM exits induced by the TPR shadow (see Section 22.6.7) and pending MTF VM exits (see Section 22.6.8. The exception takes priority over any pending non-maskable interrupt (NMI) or external interrupt and also over VM exits due to the 1-settings of the “interrupt-window exiting” and “NMI-window exiting” VM-execution controls.

A pending debug exception delivered after VM entry causes a VM exit if the bit 1 (#DB) is 1 in the exception bitmap. If it does not cause a VM exit, it updates DR6 normally.

22.6.4 VMX-Preemption Timer

If the “activate VMX-preemption timer” VM-entry control is 1, VM entry starts the VMX-preemption timer with the unsigned value in the VMX-preemption timer-value field.

If the “activate VMX-preemption timer” 1 and the value in the VMX-preemption timer-value field is zero, a VM exit occurs before execution of any instruction following VM entry (if it is not blocked by activity state). The VM exit occurs with its normal priority after any event injection. For example, any pending debug exceptions established by VM entry (see Section 22.6.3) take priority over a timer-induced VM exit. (The timer-induced VM exit will occur after delivery of the debug exception, unless that exception or its delivery causes a different VM exit.)

See Section 21.7.1 for details of the operation of the VMX-preemption timer in VMX non-root operation, including the blocking and priority of the VM exits that it causes.

22.6.5 Interrupt-Window Exiting

The “interrupt-window exiting” VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 21.3 for details).

The following items detail the treatment of these VM exits:

- These VM exits follow event injection if such injection is specified for VM entry.
- Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.
- VM exits caused by this control wake the logical processor if it just entered the HLT state because of a VM entry (see Section 22.6.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

22.6.6 NMI-Window Exiting

The “NMI-window exiting” VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 21.3 for details).

The following items detail the treatment of these VM exits:

- These VM exits follow event injection if such injection is specified for VM entry.
- Debug-trap exceptions (see Section 22.6.3) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.
- VM exits caused by this control wake the logical processor if it just entered either the HLT state or the shutdown state because of a VM entry (see Section 22.6.2). They do not occur if the logical processor just entered the wait-for-SIPI state.

22.6.7 VM Exits Induced by the TPR Shadow

If the "use TPR shadow" and "virtualize APIC accesses" VM-execution controls are both 1, a VM exit occurs immediately after VM entry if the value of bits 3:0 of the TPR threshold VM-execution control field is greater than the value of bits 7:4 in byte 80H on the virtual-APIC page (see Section 20.6.8).[1]

The following items detail the treatment of these VM exits:

- The VM exits are not blocked if RFLAGS.IF = 0 or by the setting of bits in the interruptibility-state field in guest-state area.
- The VM exits follow event injection if such injection is specified for VM entry.
- VM exits caused by this control take priority over system-management interrupts (SMIs), INIT signals, and lower priority events. They thus have priority over the VM exits described in Section 22.6.5, Section 22.6.6, and Section 22.6.8, as well as any interrupts or debug exceptions that may be pending at the time of VM entry.
- These VM exits wake the logical processor if it just entered the HLT state as part of a VM entry (see Section 22.6.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

  If such a VM exit is suppressed because the processor just entered the shutdown state, it occurs after the delivery of any event that cause the logical processor to leave the shutdown state while remaining in VMX non-root operation (e.g., due to an NMI that occurs while the "NMI-exiting" VM-execution control is 0).
- The basic exit reason is "TPR below threshold."

22.6.8 Pending MTF VM Exits

As noted in Section 22.5.2, VM entry may cause an MTF VM exit to be pending immediately after VM entry. The following items detail the treatment of these VM exits:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these VM exits. These VM exits take priority over debug-trap exceptions and lower priority events.
- These VM exits wake the logical processor if it just entered the HLT state because of a VM entry (see Section 22.6.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

[1] Note that "virtualize APIC accesses" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "virtualize APIC accesses" VM-execution control were 0. See Section 20.6.2.

22.6.9 VM Entries and Advanced Debugging Features

VM entries are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

22.7 VM-ENTRY FAILURES DURING OR AFTER LOADING GUEST STATE

VM-entry failures due to the checks identified in Section 22.3.1 and failures during the MSR loading identified in Section 22.4 are treated differently from those that occur earlier in VM entry. In these cases, the following steps take place:

1. Information about the VM-entry failure is recorded in the VM-exit information fields:
   - Exit reason.
     - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM-entry failure. The following numbers are used:

       33.VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 22.3.1.

       34.VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs (see Section 22.4).

       41.VM-entry failure due to machine check. A machine check occurred during VM entry (see Section 22.8).
     - Bit 31 is set to 1 to indicate a VM-entry failure.
     - The remainder of the field (bits 30:16) is cleared.
   - Exit qualification. This field is set based on the exit reason.
     - VM-entry failure due to invalid guest state. In most cases, the exit qualification is cleared to 0. The following non-zero values are used in the cases indicated:

       1. Not used.

       2. Failure was due to a problem loading the PDPTEs (see Section 22.3.1.6).

       3. Failure was due to an attempt to inject a non-maskable interrupt (NMI) into a guest that is blocking events through the STI blocking bit in the interruptibility-state field. Such failures are implementation-specific (see Section 22.3.1.5).

       4.Failure was due to an invalid VMCS link pointer (see Section 22.3.1.5).

       VM-entry checks on guest-state fields may be performed in any order. Thus, an indication by exit qualification of one cause does not imply that there are not also other errors. Different processors may give different exit qualifications for the same VMCS.

  - VM-entry failure due to MSR loading. The exit qualification is loaded to indicate which entry in the VM-entry MSR-load area caused the problem (1 for the first entry, 2 for the second, etc.).

— All other VM-exit information fields are unmodified.

2. Processor state is loaded as would be done on a VM exit (see Section 23.5). If this results in [CR4.PAE & CR0.PG & ~IA32_EFER.LMA] = 1, page-directory-pointer-table entries (PDPTES) may be checked and loaded (see Section 23.5.4).
3. The state of blocking by NMI is what it was before VM entry.
4. MSRs are loaded as specified in the VM-exit MSR-load area (see Section 23.6).

Although this process resembles that of a VM exit, many steps taken during a VM exit do not occur for these VM-entry failures:

- Most VM-exit information fields are not updated (see step 1 above).
- The valid bit in the VM-entry interruption-information field is not cleared.
- The guest-state area is not modified.
- No MSRs are saved into the VM-exit MSR-store area.

22.8 MACHINE CHECKS DURING VM ENTRY

If a machine check occurs during a VM entry, one of the following occurs:

- The machine check is handled normally:

  — If CR4.MCE = 1, a machine-check exception (#MC) is delivered through the IDT.

  — If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:[1]

    - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine check condition.” See *Intel® Trusted Execution Technology Preliminary Architecture Specification*.
    - If the logical processor is outside SMX operation, it goes to the shutdown state.

[1] A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

- A VM-entry failure occurs as described in Section 22.7. The basic exit reason is 41, for “VM-entry failure due to machine check.”

The first option is not used if the machine check occurs after any guest state has been loaded.

CHAPTER 23
VM EXITS

VM exits occur in response to certain instructions and events in VMX non-root operation. Section 21.1 through Section 21.3 detail the causes of VM exits. VM exits perform the following operation:

1. Information about the cause of the VM exit is recorded in the VM-exit information fields and VM-entry control fields are modified as described in Section 23.2.
2. Processor state is saved in the guest-state area (Section 23.3).
3. MSRs may be saved in the VM-exit MSR-store area (Section 23.4).
4. The following may be performed in parallel and in any order (Section 23.5):
   - Processor state is loaded based in part on the host-state area and some VM-exit controls. This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM. See Section 26.15.6 for information on how processor state is loaded by such VM exits.
   - Address-range monitoring is cleared.
5. MSRs may be loaded from the VM-exit MSR-load area (Section 23.6). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.

VM exits are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

Section 23.1 clarifies the nature of the architectural state before a VM exit begins. The steps described above are detailed in Section 23.2 through Section 23.6.

Section 26.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, ordinary transitions to SMM are replaced by VM exits to a separate SMM monitor. Called **SMM VM exits**, these are caused by the arrival of an SMI or the execution of VMCALL in VMX root operation. SMM VM exits differ from other VM exits in ways that are detailed in Section 26.15.2.

23.1 ARCHITECTURAL STATE BEFORE A VM EXIT

This section describes the architectural state that exists before a VM exit, especially for VM exits caused by events that would normally be delivered through the IDT. Note the following:

- An exception causes a VM exit **directly** if the bit corresponding to that exception is set in the exception bitmap. A non-maskable interrupt (NMI) causes a VM exit directly if the “NMI exiting” VM-execution control is 1. An external interrupt causes a VM exit directly if the “external-interrupt exiting” VM-execution control is 1. A start-up IPI (SIPI) that arrives while a logical processor is in the wait-for-SIPI activity state causes a VM exit directly. INIT signals that arrive while the processor is not in the wait-for-SIPI activity state cause VM exits directly.

- An exception, NMI, external interrupt, or software interrupt causes a VM exit **indirectly** if it does not do so directly but delivery of the event causes a nested exception, double fault, task switch, APIC access (see Section 21.2), EPT violation, or EPT misconfiguration that causes a VM exit.
- An event **results** in a VM exit if it causes a VM exit (directly or indirectly).

The following bullets detail when architectural state is and is not updated in response to VM exits:

- If an event causes a VM exit directly, it does not update architectural state as it would have if it had not caused the VM exit:
  - — A debug exception does not update DR6, DR7.GD, or IA32_DEBUGCTL.LBR. (Information about the nature of the debug exception is saved in the exit qualification field.)
  - — A page fault does not update CR2. (The linear address causing the page fault is saved in the exit-qualification field.)
  - — An NMI causes subsequent NMIs to be blocked, but only after the VM exit completes.
  - — An external interrupt does not acknowledge the interrupt controller and the interrupt remains pending, unless the “acknowledge interrupt on exit” VM-exit control is 1. In such a case, the interrupt controller is acknowledged and the interrupt is no longer pending.
  - — The flags L0 – L3 in DR7 (bit 0, bit 2, bit 4, and bit 6) are not cleared when a task switch causes a VM exit.
  - — If a task switch causes a VM exit, none of the following are modified by the task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT[1]; or the TR register.
  - — No last-exception record is made if the event that would do so directly causes a VM exit.
  - — If a machine-check exception causes a VM exit directly, this does not prevent machine-check MSRs from being updated. These are updated by the machine check itself and not the resulting machine-check exception.

[1] This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

  - If the logical processor is in an inactive state (see Section 20.4.2) and not executing instructions, some events may be blocked but others may return the logical processor to the active state. Unblocked events may cause VM exits.[1] If an unblocked event causes a VM exit directly, a return to the active state occurs only after the VM exit completes.[2] The VM exit generates any special bus cycle that is normally generated when the active state is entered from that activity state.

    MTF VM exits (see Section 21.7.2 and Section 22.6.8) are not blocked in the HLT activity state. If an MTF VM exit occurs in the HLT activity state, the logical processor returns to the active state only after the VM exit completes. MTF VM exits are blocked the shutdown state and the wait-for-SIPI state.

- If an event causes a VM exit indirectly, the event does update architectural state:
  - A debug exception updates DR6, DR7, and the IA32_DEBUGCTL MSR. No debug exceptions are considered pending.
  - A page fault updates CR2.
  - An NMI causes subsequent NMIs to be blocked before the VM exit commences.
  - An external interrupt acknowledges the interrupt controller and the interrupt is no longer pending.
  - If the logical processor had been in an inactive state, it enters the active state and, before the VM exit commences, generates any special bus cycle that is normally generated when the active state is entered from that activity state.
  - There is no blocking by STI or by MOV SS when the VM exit commences.
  - Processor state that is normally updated as part of delivery through the IDT (CS, RIP, SS, RSP, RFLAGS) is not modified. However, the incomplete delivery of the event may write to the stack.
  - The treatment of last-exception records is implementation dependent:
    - Some processors make a last-exception record when beginning the delivery of an event through the IDT (before it can encounter a nested exception). Such processors perform this update even if the event encounters a nested exception that causes a VM exit (including the case where nested exceptions lead to a triple fault).
    - Other processors delay making a last-exception record until event delivery has reached some event handler successfully (perhaps after one or more nested exceptions). Such processors do not update the last-

[1] If a VM exit takes the processor from an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

[2] An exception is made if the logical processor had been inactive due to execution of MWAIT; in this case, it is considered to have become active before the VM exit.

exception record if a VM exit or triple fault occurs before an event handler is reached.

- If the “virtual NMIs” VM-execution control is 1, VM entry injects an NMI, and delivery of the NMI causes a nested exception, double fault, task switch, or APIC access that causes a VM exit, virtual-NMI blocking is in effect before the VM exit commences.
- If a VM exit results from a fault, EPT violation, or EPT misconfiguration encountered during execution of IRET and the “NMI exiting” VM-execution control is 0, any blocking by NMI is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the VM-exit interruption-information field; see Section 23.2.2.
- If a VM exit results from a fault, EPT violation, or EPT misconfiguration encountered during execution of IRET and the “virtual NMIs” VM-execution control is 1, virtual-NMI blocking is cleared before the VM exit commences. However, the previous state of virtual-NMI blocking may be recorded in the VM-exit interruption-information field; see Section 23.2.2.
- Suppose that a VM exit is caused directly by an x87 FPU Floating-Point Error (#MF) or by any of the following events if the event was unblocked due to (and given priority over) an x87 FPU Floating-Point Error: an INIT signal, an external interrupt, an NMI, an SMI; or a machine-check exception. In these cases, there is no blocking by STI or by MOV SS when the VM exit commences.
- Normally, a last-branch record may be made when an event is delivered through the IDT. However, if such an event results in a VM exit before delivery is complete, no last-branch record is made.
- If machine-check exception results in a VM exit, processor state is suspect and may result in suspect state being saved to the guest-state area. A VM monitor should consult the RIPV and EIPV bits in the IA32_MCG_STATUS MSR before resuming a guest that caused a VM exit resulting from a machine-check exception.
- If a VM exit results from a fault, EPT violation, or EPT misconfiguration encountered while executing an instruction, data breakpoints due to that instruction may have been recognized and information about them may be saved in the pending debug exceptions field (see Section 23.3.4).
- The following VM exits are considered to happen after an instruction is executed:
  - VM exits resulting from debug traps (single-step, I/O breakpoints, and data breakpoints).
  - VM exits resulting from debug exceptions whose recognition was delayed by blocking by MOV SS.
  - VM exits resulting from some machine-check exceptions.
  - Trap-like VM exits due to execution of MOV to CR8 when the “CR8-load exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution control is 1. (Such VM exits can occur only from 64-bit mode and thus only on processors that support Intel 64 architecture.)

- — Trap-like VM exits due to execution of WRMSR when the “use MSR bitmaps” VM-execution control is 1, the value of ECX is 808H, bit 808H in write bitmap for low MSRs is 0, and the “virtualize x2APIC mode” VM-execution control is 1. See Section 21.1.3.
- — VM exits caused by TPR-shadow updates (see Section 21.5.3.3) that result from APIC accesses as part of instruction execution.

For these VM exits, the instruction’s modifications to architectural state complete before the VM exit occurs. Such modifications include those to the logical processor’s interruptibility state (see Table 20-3). If there had been blocking by MOV SS, POP SS, or STI before the instruction executed, such blocking is no longer in effect.

23.2 RECORDING VM-EXIT INFORMATION AND UPDATING VM-ENTRY CONTROL FIELDS

VM exits begin by recording information about the nature of and reason for the VM exit in the VM-exit information fields. Section 23.2.1 to Section 23.2.4 detail the use of these fields.

In addition to updating the VM-exit information fields, the valid bit (bit 31) is cleared in the VM-entry interruption-information field.

23.2.1 Basic VM-Exit Information

Section 20.9.1 defines the basic VM-exit information fields. The following items detail their use.

- **Exit reason.**
  - — Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM exit. Appendix I lists the numbers used and their meaning.
  - — The remainder of the field (bits 31:16) is cleared on every VM exit.
- **Exit qualification.** This field is saved for VM exits due to the following causes: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); system-management interrupts (SMIs) that arrive immediately after the retirement of I/O instructions; task switches; INVLPG;LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 21.2); and EPT violations. For all other VM exits, this field is cleared. The following items provide details:

— For debug exceptions, the exit qualification contains information about the debug exception. The information has the format given in Table 23-1.

Table 23-1. Exit Qualification for Debug Exceptions

Bit Position(s)	Contents
3:0	B3 – B0. When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if its corresponding enabling bit in DR7 is not set.
12:4	Reserved (cleared to 0).
13	BD. When set, this bit indicates that the cause of the debug exception is “debug register access detected.”
14	BS. When set, this bit indicates that the cause of the debug exception is either the execution of a single instruction (if RFLAGS.TF = 1 and IA32_DEBUGCTL.BTF = 0) or a taken branch (if RFLAGS.TF = DEBUGCTL.BTF = 1).
63:15	Reserved (cleared to 0). Bits 63:32 exist only on processors that support Intel 64 architecture.

— For page-fault exceptions, the exit qualification contains the linear address that caused the page fault. On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

— Start-up IPI (SIPI). The SIPI vector information is stored in bits 7:0 of the exit qualification. Bits 63:8 are cleared to 0.

— Task switch. Details about the reason for the VM exit are encoded as shown in Table 23-2.

Table 23-2. Exit Qualification for Task Switch

Bit Position(s)	Contents
15:0	Selector of task-state segment (TSS) to which the guest attempted to switch
29:16	Reserved (cleared to 0)
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT

Table 23-2. Exit Qualification for Task Switch (Contd.)

Bit Position(s)	Contents
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

— For INVLPG, the exit qualification contains the linear-address operand of the instruction.

- On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
- If the INVLPG source operand specifies an unusable segment, the linear address specified in the exit qualification will match the linear address that the INVLPG would have used if no VM exit occurred. Note that this address is not architecturally defined and may be implementation-specific.

— For LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, and VMXON, the exit qualification receives the value of the instruction's displacement field, which is sign-extended to 64 bits if necessary (32 bits on processors that do not support Intel 64 architecture). If the instruction has no displacement (for example, has a register operand), zero is stored into the exit qualification.

On processors that support Intel 64 architecture, an exception is made for RIP-relative addressing (used only in 64-bit mode). Such addressing causes an instruction to use an address that is the sum of the displacement field and the value of RIP that references the following instruction. In this case, the exit qualification is loaded with the sum of the displacement field and the appropriate RIP value.

In all cases, bits of this field beyond the instruction's address size are undefined. For example, suppose that the address-size field in the VM-exit instruction-information field (see Section 20.9.4 and Section 23.2.4) reports an *n*-bit address size. Then bits 63:*n* (bits 31:*n* on processors that do not support Intel 64 architecture) of the instruction displacement are undefined.

— For control-register accesses, the exit qualification contains information about the access and has the format given in Table 23-3.

Table 23-3. Exit Qualification for Control-Register Accesses

Bit Positions	Contents
3:0	Number of control register (0 for CLTS and LMSW). Bit 3 is always 0 on processors that do not support Intel 64 architecture as they do not support CR8.

Table 23-3. Exit Qualification for Control-Register Accesses (Contd.)

Bit Positions	Contents
5:4	Access type: 0 = MOV to CR 1 = MOV from CR 2 = CLTS 3 = LMSW
6	LMSW operand type: 0 = register 1 = memory For CLTS and MOV CR, cleared to 0
7	Reserved (cleared to 0)
11:8	For MOV CR, the general-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) For CLTS and LMSW, cleared to 0
15:12	Reserved (cleared to 0)
31:16	For LMSW, the LMSW source data For CLTS and MOV CR, cleared to 0
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

— For MOV DR, the exit qualification contains information about the instruction and has the format given in Table 23-4.

Table 23-4. Exit Qualification for MOV DR

Bit Position(s)	Contents
2:0	Number of debug register
3	Reserved (cleared to 0)
4	Direction of access (0 = MOV to DR; 1 = MOV from DR)
7:5	Reserved (cleared to 0)
11:8	General-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8 –15 = R8 – R15, respectively
63:12	Reserved (cleared to 0)

— For I/O instructions, the exit qualification contains information about the instruction and has the format given in Table 23-5.

Table 23-5. Exit Qualification for I/O Instructions

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)

Table 23-5. Exit Qualification for I/O Instructions (Contd.)

Bit Position(s)	Contents
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in the I/O instruction)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

— MWAIT. A value that indicates whether address-range monitoring hardware was armed. The exit qualification is set to either 0 (if address-range monitoring hardware is not armed) or 1 (if address-range monitoring hardware is armed).

— For APIC-access VM exits resulting from linear accesses and guest-physical accesses to the APIC-access page (see Section 21.2.1 and Section 21.2.1), the exit qualification contains information about the instruction and has the format given in Table 23-6.[1]

Table 23-6. Exit Qualification for APIC-Access VM Exits from Linear Accesses and Guest-Physical Accesses

Bit Position(s)	Contents
11:0	▪ If the APIC-access VM exit is due to a linear access, the offset of access within the APIC page. ▪ Undefined if the APIC-access VM exit is due a guest-physical access
15:12	Access type: 0 = linear access for a data read during instruction execution 1 = linear access for a data write during instruction execution 2 = linear access for an instruction fetch 3 = linear access (read or write) during event delivery 10 = guest-physical access during event delivery 15 = guest-physical access for an instruction fetch or during instruction execution Other values not used
63:16	Reserved (cleared to 0). Bits 63:32 exist only on processors that support Intel 64 architecture.

Such a VM exit that set bits 15:12 of the exit qualification to 0000b (data read during instruction execution) or 0001b (data write during instruction execution) set bit 12—which distinguishes data read from data write—to that which would have been stored in bit 1—W/R—of the page-fault error code had the access caused a page fault instead of an APIC-access VM exit. This implies the following:

- For APIC-access VM exits caused by the CLFLUSH instruction, the access type is “data read during instruction execution.”
- For APIC-access VM exits caused by the ENTER instruction, the access type is “data write during instruction execution.”
- For APIC-access VM exits caused by the MASKMOVQ and MASKMOVDQU instructions, the access type is “data write during instruction execution.”
- For APIC-access VM exits caused by the MONITOR instruction, the access type is “data read during instruction execution.”

Such a VM exit stores 1 for bit 31 for IDT-vectoring information field (see Section 23.2.3) if and only if it sets bits 15:12 of the exit qualification to 0011b (linear access during event delivery) or 1010b (guest-physical access during event delivery).

See Section 21.2.1.3 for further discussion of these instructions and APIC-access VM exits.

For APIC-access VM exits resulting from physical accesses, the APIC-access page (see Section 21.2.3), the exit qualification is undefined.

— For EPT violations, the exit qualification contains information about the instruction and has the format given in Table 23-5.

Table 23-7. Exit Qualification for EPT Violations

Bit Position(s)	Contents
0	Set if the access causing the EPT violation was a data read.
1	Set if the access causing the EPT violation was a data write.
2	Set if the access causing the EPT violation was an instruction fetch.
3	The logical-AND of bit 0 in the EPT paging-structures entries used to translate the guest-physical address of the access causing the EPT violation (indicates that the guest-physical address was readable).[1]

[1] The exit qualification is undefined if the access was part of the logging of a branch record or a precise-event-based-sampling (PEBS) record to the DS save area. It is recommended that software configure the paging structures so that no address in the DS save area translates to an address on the APIC-access page.

Table 23-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
4	The logical-AND of bit 1 in the EPT paging-structures entries used to translate the guest-physical address of the access causing the EPT violation (indicates that the guest-physical address was writeable).
5	The logical-AND of bit 2 in the EPT paging-structures entries used to translate the guest-physical address of the access causing the EPT violation (indicates that the guest-physical address was executable).
6	Reserved (cleared to 0).
7	Set if the guest linear-address field is valid. The guest linear-address field is valid for all EPT violations except those resulting from an attempt to load the guest PDPTEs as part of the execution of the MOV CR instruction.
8	If bit 7 is 1: ▪ Set if the access causing the EPT violation is to a guest-physical address that is the translation of a linear address. ▪ Clear if the access causing the EPT violation is to a paging-structure entry as part of a page walk or the update of an accessed or dirty bit. Reserved if bit 7 is 0 (cleared to 0).
63:9	Reserved (cleared to 0).

NOTES:

1. Bits 5:3 are cleared to 0 if any of EPT paging-structures entries used to translate the guest-physical address of the access causing the EPT violation is not present (see Section 25.3).

An EPT violation that occurs during as a result of execution of a read-modify-write operation sets bit 1 (data write). Whether it also sets bit 0 (data read) is implementation-specific and, for a given implementation, may differ for different kinds of read-modify-write operations.

- **Guest-linear address.** For some VM exits due to some instructions, this field receives the linear address of one of the instruction operands. For some VM exits due to EPT violations, this field receives a linear address associated the guest-physical address that caused the EPT violation.
  - — VM exits due to attempts to execute LMSW with a memory operand. In these cases, this field receives the linear address of that operand. On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
  - — VM exits due to attempts to execute INS or OUTS for which the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) is usable. The field receives the value of the linear address generated by

ES:(E)DI (for INS) or segment:(E)SI (for OUTS; the default segment is DS but can be overridden by a segment override prefix). (If the relevant segment is not usable, the value is undefined.) On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

  - VM exits due to EPT violations that set bit 7 of the exit qualification (see Table 23-7; these are all EPT violations except those resulting from an attempt to load the PDPTEs as of execution of the MOV CR instruction). The linear address may translate to the guest-physical address whose access caused the EPT violation. Alternatively, translation of the linear address may reference a paging-structure entry whose access caused the EPT violation.
  - For all other VM exits, the field is undefined.

- **Guest-physical address.** For a VM exit due to an EPT violation or an EPT misconfiguration, this field receives the guest-physical address that caused the EPT violation or EPT misconfiguration. For all other VM exits, the field is undefined.

23.2.2 Information for VM Exits Due to Vectored Events

Section 20.9.2 defines fields containing information for VM exits due to the following events: exceptions (including those generated by the instructions INT3, INTO, BOUND, and UD2); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs). Such VM exits include those that occur on an attempt at a task switch that causes an exception before generating the VM exit due to the task switch that causes the VM exit.

The following items detail the use of these fields:

- **VM-exit interruption information** (format given in Table 20-14). The following items detail how this field is established for VM exits due to these events:
  - For an exception, bits 7:0 receive the exception vector (at most 31). For an NMI, bits 7:0 are set to 2. For an external interrupt, bits 7:0 receive the interrupt number.
  - Bits 10:8 are set to 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), or 6 (software exception). Hardware exceptions comprise all exceptions except breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. Note that BOUND range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD2 are hardware exceptions.
  - Bit 11 is set to 1 if the VM exit is caused by a hardware exception that would have delivered an error code on the stack. If bit 11 is set to 1, the error code is placed in the VM-exit interruption error code (see below).
  - Bit 12 is undefined in any of the following cases:

        - If the “NMI exiting” VM-execution control is 1 and the “virtual NMIs” VM-execution control is 0.
        - If the VM exit sets the valid bit in the IDT-vectoring information field (see Section 23.2.3).
        - If the VM exit is due to a double fault (the interruption type is hardware exception and the vector is 8).

        Otherwise, bit 12 is defined as follows:

        - If the “virtual NMIs” VM-execution control is 0, the VM exit is due to a fault on the IRET instruction, and blocking by NMI (see Table 20-3) was in effect before execution of IRET, bit 12 is set to 1.
        - If the “virtual NMIs” VM-execution control is 1, the VM exit is due to a fault on the IRET instruction, and virtual-NMI blocking was in effect before execution of IRET, bit 12 is set to 1.
        - For all other relevant VM exits, bit 12 is cleared to 0.

    — Bits 30:13 are always set to 0.

    — Bit 31 is always set to 1.

    For other VM exits (including those due to external interrupts when the “acknowledge interrupt on exit” VM-exit control is 0), the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- VM-exit interruption error code.

    — For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the VM-exit interruption-information field, this field receives the error code that would have been pushed on the stack had the event causing the VM exit been delivered normally through the IDT. The EXT bit is set in this field exactly when it would be set normally. For exceptions that occur during the delivery of double fault (if the IDT-vectoring information field indicates a double fault), the EXT bit is set to 1, assuming that (1) that the exception would produce an error code normally (if not incident to double-fault delivery) and (2) that the error code uses the EXT bit (not for page faults, which use a different format).

    — For other VM exits, the value of this field is undefined.

23.2.3 Information for VM Exits During Event Delivery

Section 20.9.3 defined fields containing information for VM exits that occur while delivering an event through the IDT and as a result of any of the following cases:[1]

[1] This includes the case in which a VM exit occurs while delivering a software interrupt (INT *n*) through the 16-bit IVT (interrupt vector table) that is used in virtual-8086 mode with virtual-machine extensions (if RFLAGS.VM = CR4.VME = 1).

- A fault occurs during event delivery and causes a VM exit (because the bit associated with the fault is set to 1 in the exception bitmap).
- A task switch is invoked through a task gate in the IDT. Note that the VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 21.6.2).
- Event delivery causes an APIC-access VM exit (see Section 21.2).
- An EPT violation or EPT misconfiguration that occurs during event delivery.

Note that these fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 22.5.1.2).

A VM exit is not considered to occur during event delivery in any of the following circumstances:

- The original event causes the VM exit directly (for example, because the original event is a non-maskable interrupt (NMI) and the "NMI exiting" VM-execution control is 1).
- The original event results in a double-fault exception that causes the VM exit directly.
- The VM exit occurred as a result of fetching the first instruction of the handler invoked by the event delivery.
- The VM exit is caused by a triple fault.

The following items detail the use of these fields:

- IDT-vectoring information (format given in Table 20-15). The following items detail how this field is established for VM exits that occur during event delivery:
  - — If the VM exit occurred during delivery of an exception, bits 7:0 receive the exception vector (at most 31). If the VM exit occurred during delivery of an NMI, bits 7:0 are set to 2. If the VM exit occurred during delivery of an external interrupt, bits 7:0 receive the interrupt number.
  - — Bits 10:8 are set to indicate the type of event that was being delivered when the VM exit occurred: 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 4 (software interrupt), 5 (privileged software interrupt), or 6 (software exception).

    Hardware exceptions comprise all exceptions except breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. Note that BOUND range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD2 are hardware exceptions.

    Bits 10:8 may indicate privileged software interrupt if such an event was injected as part of VM entry.
  - — Bit 11 is set to 1 if the VM exit occurred during delivery of a hardware exception that would have delivered an error code on the stack. If bit 11 is set to 1, the error code is placed in the IDT-vectoring error code (see below).

- Bit 12 is undefined.
- Bits 30:13 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits, the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- IDT-vectoring error code.
  - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the IDT-vectoring information field, this field receives the error code that would have been pushed on the stack by the event that was being delivered through the IDT at the time of the VM exit. The EXT bit is set in this field when it would be set normally.
  - For other VM exits, the value of this field is undefined.

23.2.4 Information for VM Exits Due to Instruction Execution

Section 20.9.4 defined fields containing information for VM exits that occur due to instruction execution. (The VM-exit instruction length is also used for VM exits that occur during the delivery of a software interrupt or software exception.) The following items detail their use.

- **VM-exit instruction length.** This field is used in the following cases:
  - For fault-like VM exits due to attempts to execute one of the following instructions that cause VM exits unconditionally (see Section 21.1.2) or based on the settings of VM-execution controls (see Section 21.1.3): CLTS, CPUID, GETSEC, HLT, IN, INS INVD, INVLPG, LGDT, LIDT, LLDT, LMSW, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, RDMSR, RDPMC, RDTSC, RSM, SGDT, SIDT, SLDT, STR, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WRMSR, and XSETBV.[1]
  - For VM exits due to software exceptions (those generated by executions of INT3 or INTO).
  - For VM exits due to faults encountered during delivery of a software interrupt, privileged software exception, or software exception.
  - For VM exits due to attempts to effect a task switch via instruction execution. These are VM exits that produce an exit reason indicating task switch and either of the following:

[1] This item applies only to fault-like VM exits. It does not apply to trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1 or to those following executions of the WRMSR instruction when the “virtualize x2APIC mode” VM-execution control is 1.

  - An exit qualification indicating execution of CALL, IRET, or JMP instruction.
  - An exit qualification indicating a task gate in the IDT and an IDT-vectoring information field indicating that the task gate was encountered during delivery of a software interrupt, privileged software exception, or software exception.

— For APIC-access VM exits resulting from linear accesses (see Section 21.2.1) and encountered during delivery of a software interrupt, privileged software exception, or software exception.[1]

In all the above cases, this field receives the length in bytes (1–15) of the instruction (including any instruction prefixes) whose execution led to the VM exit (see the next paragraph for one exception).

The cases of VM exits encountered during delivery of a software interrupt, privileged software exception, or software exception include those encountered during delivery of events injected as part of VM entry (see Section 22.5.1.2). If the original event was injected as part of VM entry, this field receives the value of the VM-entry instruction length.

All VM exits other than those listed in the above items leave this field undefined.

- **VM-exit instruction information**.

— For VM exits due to attempts to execute INS or OUTS on some processors, this field receives information about the instruction that caused the VM exit and has the format is given in Table 23-8.[2]

Table 23-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS

Bit Position(s)	Content
1:0	Undefined.
2	Reserved (cleared to 0).
6:3	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.

[1] The VM-exit instruction-length field is not defined following APIC-access VM exits resulting from physical accesses (see Section 21.2.3) even if encountered during delivery of a software interrupt, privileged software exception, or software exception.

[2] Whether the processor provides this information on these VM exits can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix G.1).

Table 23-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS (Contd.)

Bit Position(s)	Content
10	Undefined.
14:11	Reserved (cleared to 0)
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values unused. Undefined for INS.
31:18	Undefined.

— For VM exits due to attempts to execute LIDT, LGDT, LLDT, LTR, SIDT, SGDT, SLDT, or STR, this field receives information about the instruction that caused the VM exit and has the format is given in Table 23-10.

Table 23-9. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, LLDT, LTR, SIDT, SGDT, SLDT, and STR

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
2	Reserved (cleared to 0)
6:3	Undefined.
8:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Value 3 not used.
14:9	Reserved (cleared to 0)

Table 23-9. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, LLDT, LTR, SIDT, SGDT, SLDT, and STR (Contd.)

Bit Position(s)	Content
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values unused.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined if bit 22 is set.
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined if bit 27 is set.
27	BaseReg invalid (0 = valid; 1 = invalid)
29:28	Instruction identity: 0: SGDT or SLDT 1: SIDT or STR 2: LGDT or LLDT 3: LIDT or LTR The specific instruction can be determined by consulting also the basic exit reason (see Appendix I).
31:30	Undefined.

— For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, or VMXON, this field receives information about the instruction that caused the VM exit and has the format is given in Table 23-10.

Table 23-10. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, and VMXON

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) or for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Reserved (cleared to 0)
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register) Note that VMCLEAR, VMPTRLD, VMPTRST, and VMXON are always memory instructions and thus clear this bit.
14:11	Reserved (cleared to 0)

Table 23-10. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, and VMXON (Contd.)

Bit Position(s)	Content
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values unused. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined if bit 22 is set or undefined.
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined if bit 27 is set or undefined.
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
31:28	Reg2 (same encoding as Reg1 above) Undefined on VM exits due to VMCLEAR, VMPTRLD, VMPTRST, and VMXON.

— For all other VM exits, the field is undefined.

- **I/O RCX, I/O RSI, I/O RDI, I/O RIP**. These fields are undefined except for SMM VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions. See Section 26.15.2.3.

23.3 SAVING GUEST STATE

Each field in the guest-state area of the VMCS (see Section 20.4) is written with the corresponding component of processor state. On processors that support Intel 64 architecture, the full values of each natural-width field (see Section 20.10.2) is saved regardless of the mode of the logical processor before and after the VM exit.

In general, the state saved is that which was in the logical processor at the time the VM exit commences. See Section 23.1 for a discussion of which architectural updates occur at that time.

Section 23.3.1 through Section 23.3.4 provide details for how certain components of processor state are saved. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

23.3.1 Saving Control Registers, Debug Registers, and MSRs

Contents of certain control registers, debug registers, and MSRs is saved as follows:

- The contents of CR0, CR3, CR4, and the IA32_SYSENTER_CS, IA32_SYSENTER_ESP, and IA32_SYSENTER_EIP MSRs are saved into the corresponding fields. Bits 63:32 of the IA32_SYSENTER_CS MSR are not saved. On processors that do not support Intel 64 architecture, bits 63:32 of the IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are not saved.
- If the “save debug controls” VM-exit control is 1, the contents of DR7 and the IA32_DEBUGCTL MSR are saved into the corresponding fields. Note that the first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always saved data into these fields.
- If the “save IA32_PAT” VM-exit control is 1, the contents of the IA32_PAT MSR are saved into the corresponding field.
- If the “save IA32_EFER” VM-exit control is 1, the contents of the IA32_EFER MSR are saved into the corresponding field.
- The value of the SMBASE field is undefined after all VM exits except SMM VM exits. See Section 26.15.2.

23.3.2 Saving Segment Registers and Descriptor-Table Registers

For each segment register (CS, SS, DS, ES, FS, GS, LDTR, or TR), the values saved for the base-address, segment-limit, and access rights are based on whether the register was unusable (see Section 20.4.1) before the VM exit:

- If the register was unusable, the values saved into the following fields are undefined: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in the access-rights field. The following exceptions apply:
  - — CS.
    - The base-address and segment-limit fields are saved.
    - The L, D, and G bits are saved in the access-rights field.
  - — SS.
    - DPL is saved in the access-rights field.
    - On processors that support Intel 64 architecture, bits 63:32 of the value saved for the base address are always zero.
  - — DS and ES. On processors that support Intel 64 architecture, bits 63:32 of the values saved for the base addresses are always zero.
  - — FS and GS. The base-address field is saved.

- LDTR. The value saved for the base address is always canonical.

- If the register was not unusable, the values saved into the following fields are those which were in the register before the VM exit: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in access rights.
- Bits 31:17 and 11:8 in the access-rights field are always cleared. Bit 16 is set to 1 if and only if the segment is unusable.

The contents of the GDTR and IDTR registers are saved into the corresponding base-address and limit fields.

23.3.3 Saving RIP, RSP, and RFLAGS

The contents of the RIP, RSP, and RFLAGS registers are saved as follows:

- The value saved in the RIP field is determined by the nature and cause of the VM exit:
  - If the VM exit occurs due to by an attempt to execute an instruction that causes VM exits unconditionally or that has been configured to cause a VM exit via the VM-execution controls, the value saved references that instruction.
  - If the VM exit is caused by an occurrence of an INIT signal, a start-up IPI (SIPI), or system-management interrupt (SMI), the value saved is that which was in RIP before the event occurred.
  - If the VM exit occurs due to the 1-setting of either the “interrupt-window exiting” VM-execution control or the “NMI-window exiting” VM-execution control, the value saved is that which would be in the register had the VM exit not occurred.
  - If the VM exit is due to an external interrupt, non-maskable interrupt (NMI), or hardware exception (as defined in Section 23.2.2), the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate,[1] or into the old task-state segment had the event been delivered through a task gate).
  - If the VM exit is due to a triple fault, the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate, or into the old task-state segment had the event been delivered through a task gate) had delivery of the double fault not encountered the nested exception that caused the triple fault.
  - If the VM exit is due to a software exception (due to an execution of INT3 or INTO), the value saved references the INT3 or INTO instruction that caused that exception.

[1] The reference here is to the full value of RIP before any truncation that would occur had the stack width been only 32 bits or 16 bits.

  - — Suppose that the VM exit is due to a task switch that was caused by execution of CALL, IRET, or JMP or by execution of a software interrupt (INT *n*) or software exception (due to execution of INT3 or INTO) that encountered a task gate in the IDT. The value saved references the instruction that caused the task switch (CALL, IRET, JMP, INT *n*, INT3, or INTO).
  - — Suppose that the VM exit is due to a task switch that was caused by a task gate in the IDT that was encountered for any reason except the direct access by a software interrupt or software exception. The value saved is that which would have been saved in the old task-state segment had the task switch completed normally.
  - — If the VM exit is due to an execution of MOV to CR8 or WRMSR that reduced the value of the TPR shadow[1] below that of TPR threshold VM-execution control field, the value saved references the instruction following the MOV to CR8 or WRMSR.
  - — If the VM exit was caused by a TPR-shadow update (see Section 21.5.3.3) that results from an APIC access as part of instruction execution, the value saved references the instruction following the one whose execution caused the VTPR access.
- The contents of the RSP register are saved into the RSP field.
- With the exception of the resume flag (RF; bit 16), the contents of the RFLAGS register is saved into the RFLAGS field. RFLAGS.RF is saved as follows:
  - — If the VM exit is caused directly by an event that would normally be delivered through the IDT, the value saved is that which would appear in the saved RFLAGS image (either that which would be saved on the stack had the event been delivered through a trap or interrupt gate[2] or into the old task-state segment had the event been delivered through a task gate) had the event been delivered through the IDT. See below for VM exits due to task switches caused by task gates in the IDT.
  - — If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
  - — If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.
  - — If the VM exit is caused by an attempt to execute an instruction that unconditionally causes VM exits or one that was configured to do with a VM-execution control, the value saved is 0.[3]

[1] The TPR shadow is bits 7:4 of the byte at offset 80H of the virtual-APIC page (see Section 20.6.8).

[2] The reference here is to the full value of RFLAGS before any truncation that would occur had the stack width been only 32 bits or 16 bits.

- For APIC-access VM exits and for VM exits caused by EPT violations and EPT misconfigurations, the value saved depends on whether the VM exit occurred during delivery of an event through the IDT:
    - If the VM exit stored 0 for bit 31 for IDT-vectoring information field (because the VM exit did not occur during delivery of an event through the IDT; see Section 23.2.3), the value saved is 1.
    - If the VM exit stored 1 for bit 31 for IDT-vectoring information field (because the VM exit did occur during delivery of an event through the IDT), the value saved is the value that would have appeared in the saved RFLAGS image had the event been delivered through the IDT (see above).
- For all other VM exits, the value saved is the value RFLAGS.RF had before the VM exit occurred.

23.3.4 Saving Non-Register State

Information corresponding to guest non-register state is saved as follows:

- The activity-state field is saved with the logical processor's activity state before the VM exit.[1] See Section 23.1 for details of how events leading to a VM exit may affect the activity state.
- The interruptibility-state field is saved to reflect the logical processor's interruptibility before the VM exit. See Section 23.1 for details of how events leading to a VM exit may affect this state. VM exits that end outside system-management mode (SMM) save bit 2 (blocking by SMI) as 0 regardless of the state of such blocking before the VM exit.

  Bit 3 (blocking by NMI) is treated specially if the "virtual NMIs" VM-execution control is 1. In this case, the value saved for this field does not indicate the blocking of NMIs but rather the state of virtual-NMI blocking.
- The pending debug exceptions field is saved as clear for all VM exits except the following:
    - A VM exit caused by an INIT signal, a machine-check exception, or a system-management interrupt (SMI).
    - A VM exit with basic exit reason either "TPR below threshold."[2]
    - A VM exit with basic exit reason "monitor trap flag."

[3] This is true even if RFLAGS.RF was 1 before the instruction was executed. If, in response to such a VM exit, a VM monitor re-enters the guest to re-execute the instruction that caused the VM exit (for example, after clearing the VM-execution control that caused the VM exit), the instruction may encounter a code breakpoint that has already been processed. A VM monitor can avoid this by setting the guest value of RFLAGS.RF to 1 before resuming guest software.

[1] If this activity state was an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

- — VM exits that are not caused by debug exceptions and that occur while there is MOV-SS blocking of debug exceptions.

For VM exits that do not clear the field, the value saved is determined as follows:

- — Each of bits 3:0 may be set if it corresponds to a matched breakpoint. This may be true even if the corresponding breakpoint is not enabled in DR7.
- — Suppose that a VM exit is due to an INIT signal, a machine-check exception, or an SMI; or that a VM exit has basic exit reason “TPR below threshold” or “monitor trap flag.” In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit.

  If the VM exit occurs immediately after VM entry, the value saved may match that which was loaded on VM entry (see Section 22.6.3). Otherwise, the following items apply:

  - Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 22.6.3) and the VM exit occurred before those exceptions were either delivered or lost. In other cases, bit 12 is cleared to 0.
  - Bit 14 (BS) is set if RFLAGS.TF = 1 in either of the following cases:
    - IA32_DEBUGCTL.BTF = 0 and the cause of a pending debug exception was the execution of a single instruction.
    - IA32_DEBUGCTL.BTF = 1 and the cause of a pending debug exception was a taken branch.
- — Suppose that a VM exit is due to another reason (but not a debug exception) and occurs while there is MOV-SS blocking of debug exceptions. In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit. If the VM exit occurs immediately after VM entry (no instructions were executed in VMX non-root operation), the value saved may match that which was loaded on VM entry (see Section 22.6.3). Otherwise, the following items apply:
  - Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 22.6.3) and the VM exit occurred before those exceptions were either delivered or lost. In other cases, bit 12 is cleared to 0.
  - The setting of bit 14 (BS) is implementation-specific. However, it is not set if RFLAGS.TF = 0 or IA32_DEBUGCTL.BTF = 1.
- — The reserved bits in the field are cleared.

[2] This item includes VM exits that occur after executions of MOV to CR8 or WRMSR (Section 21.1.3), TPR-shadow updates (Section 21.5.3.3), and certain VM entries (Section 22.6.7).

- If the "save VMX-preemption timer value" VM-exit control is 1, the value of timer is saved into the VMX-preemption timer-value field. This is the value loaded from this field on VM entry as subsequently decremented (see Section 21.7.1). (If the "save VMX-preemption timer value" VM-exit control is 0, VM exit does not modify the value of the VMX-preemption timer-value field.)
- If the logical processor supports the 1-setting of the "enable EPT" VM-execution control, values are saved into the four (4) PDPTE fields as follows:
  - If the "enable EPT" VM-execution control is 1 and the logical processor was using PAE paging at the time of the VM exit, the PDPTE values currently in use are saved:[1]
    - The values saved into bits 11:9 of each of the fields is undefined.
    - If the value saved into one of the fields has bit 0 (present) clear, the value saved into bits 63:1 of that field is undefined. That value need not correspond to the value that was loaded by VM entry or to any value that might have been loaded in VMX non-root operation.
    - If the value saved into one of the fields has bit 0 (present) set, the value saved into bits 63:12 of the field is a guest-physical address.
  - If the "enable EPT" VM-execution control is 0 or the logical processor was not using PAE paging at the time of the VM exit, the values saved are undefined.

23.4 SAVING MSRS

After processor state is saved to the guest-state area, values of MSRs may be stored into the VM-exit MSR-store area (see Section 20.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-store count) is processed in order by storing the value of the MSR indexed by bits 31:0 (as they would be read by RDMSR) into bits 127:64. Processing of an entry fails in either of the following cases:

- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be read only in system-management mode (SMM) and the VM exit will not end in SMM.
- The value of bits 31:0 indicates an MSR that cannot be saved on VM exits for model-specific reasons. A processor may prevent certain MSRs (based on the value of bits 31:0) from being stored on VM exits, even if they can normally be read by RDMSR. Such model-specific behavior is documented in Appendix B.

[1] A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 3.8 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. Note that "enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM exit functions as if the "enable EPT" VM-execution control were 0. See Section 20.6.2.

- Bits 63:32 of the entry are not all 0.
- An attempt to read the MSR indexed by bits 31:0 would cause a general-protection exception if executed via RDMSR with CPL = 0.

A VMX abort occurs if processing fails for any entry. See Section 23.7.

23.5 LOADING HOST STATE

Processor state is updated on VM exits in the following ways:

- Some state is loaded from or otherwise determined by the contents of the host-state area.
- Some state is determined by VM-exit controls.
- Some state is established in the same way on every VM exit.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order.

On processors that support Intel 64 architecture, the full values of each 64-bit field loaded (for example, the base address for GDTR) is loaded regardless of the mode of the logical processor before and after the VM exit.

The loading of host state is detailed in Section 23.5.1 to Section 23.5.5. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

A logical processor is in IA-32e mode after a VM exit only if the “host address-space size” VM-exit control is 1. If the logical processor was in IA-32e mode before the VM exit and this control is 0, a VMX abort occurs. See Section 23.7.

In addition to loading host state, VM exits clear address-range monitoring (Section 23.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 23.6). This loading occurs only after the state loading described in this section.

23.5.1 Loading Host Control Registers, Debug Registers, MSRs

VM exits load new values for controls registers, debug registers, and some MSRs:

- CR0, CR3, and CR4 are loaded from the CR0 field, the CR3 field, and the CR4 field, respectively, with the following exceptions:
  - — The following bits are not modified:
    - For CR0, ET, CD, NW; bits 63:32 (on processors that support Intel 64 architecture), 28:19, 17, and 15:6; and any bits that are fixed in VMX operation (see Section 19.8).[1]

    - For CR3, bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width (they are cleared to 0).[1] (This item applies only to processors that support Intel 64 architecture.)
    - For CR4, any bits that are fixed in VMX operation (see Section 19.8).
  - CR4.PAE is set to 1 if the "host address-space size" VM-exit control is 1.
- DR7 is set to 400H.
- The following MSRs are established as follows:
  - The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
  - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since that field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
  - IA32_SYSENTER_ESP MSR and IA32_SYSENTER_EIP MSR are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
  - The following are performed on processors that support Intel 64 architecture:
    - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 23.5.2).
    - The LMA and LME bits in the IA32_EFER MSR are each loaded with the setting of the "host address-space size" VM-exit control.
  - If the "load IA32_PERF_GLOBAL_CTRL" VM-exit control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field.
  - If the "load IA32_PAT" VM-exit control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field.
  - If the "load IA32_EFER" VM-exit control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field.

  With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-exit MSR-load area. See Section 23.6.

If the "enable VPID" VM-execution control is 0 and any of CR3[63:5] (CR3[31:5] on processors that do not support Intel 64 architecture), CR4.PAE, CR4.PSE, or IA32_EFER.LMA is changing, the TLBs and paging-structure caches are updated so that, after VM exit, the logical processor does not use translations that were cached before the transition. This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if CR4.PAE was 1 before and after the transition).[2] If the "enable VPID" VM-execution control is 1, the

[1] Note that bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.

[1] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

logical processor may invalidate entries in the TLBs and paging-structure caches, but it is not required to do so.

23.5.2 Loading Host Segment and Descriptor-Table Registers

Each of the registers CS, SS, DS, ES, FS, GS, and TR is loaded as follows (see below for the treatment of LDTR):

- The selector is loaded from the selector field. The segment is unusable if its selector is loaded with zero. Note that the checks specified Section 22.3.1.2 limit the selector values that may be loaded. In particular, CS and TR are never loaded with zero and are thus never unusable. SS can be loaded with zero only on processors that support Intel 64 architecture and only if the VM exit is to 64-bit mode (64-bit mode allows use of segments marked unusable).
- The base address is set as follows:
  - CS. Cleared to zero.
  - SS, DS, and ES. Undefined if the segment is unusable; otherwise, cleared to zero.
  - FS and GS. Undefined (but, on processors that support Intel 64 architecture, canonical) if the segment is unusable and the VM exit is not to 64-bit mode; otherwise, loaded from the base-address field. Note that, on processors that support Intel 64 architecture, the values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
  - TR. Loaded from the host-state area.
- The segment limit is set as follows:
  - CS. Set to FFFFFFFFH (corresponding to a descriptor limit of FFFFFH and a G-bit setting of 1).
  - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to FFFFFFFFH.
  - TR. Set to 00000067H.
- The type field and S bit are set as follows:
  - CS. Type set to 11 and S set to 1 (execute/read, accessed, non-conforming code segment).
  - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, type set to 3 and S set to 1 (read/write, accessed, expand-up data segment).
  - TR. Type set to 11 and S set to 0 (busy 32-bit task-state segment).
- The DPL is set as follows:

[2] This paragraph applies only to translations of linear addresses. It does not apply to translations of guest-physical addresses using EPT. See Section 25.6 for details regarding the caching of translations of guest-physical addresses.

  - CS, SS, and TR. Set to 0. The current privilege level (CPL) will be 0 after the VM exit completes.
  - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 0.
- The P bit is set as follows:
  - CS, TR. Set to 1.
  - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the setting of the “host address-space size” VM-exit control. Because the value of this control is also loaded into IA32_EFER.LMA (see Section 23.5.1), no VM exit is ever to compatibility mode (which requires IA32_EFER.LMA = 1 and CS.L = 0).
- D/B.
  - CS. Loaded with the inverse of the setting of the “host address-space size” VM-exit control. For example, if that control is 0, indicating a 32-bit guest, CS.D/B is set to 1.
  - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
  - TR. Set to 0.
- G.
  - CS. Set to 1.
  - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
  - TR. Set to 0.

The host-state area does not contain a selector field for LDTR. LDTR is established as follows on all VM exits: the selector is cleared to 0000H, the segment is marked unusable and is otherwise undefined (although the base address is always canonical).

The base addresses for GDTR and IDTR are loaded from the GDTR base-address field and the IDTR base-address field, respectively. The GDTR and IDTR limits are each set to FFFFH.

23.5.3 Loading Host RIP, RSP, and RFLAGS

RIP and RSP are loaded from the RIP field and the RSP field, respectively. RFLAGS is cleared, except bit 1, which is always set.

23.5.4 Checking and Loading Host Page-Directory-Pointer-Table Entries

If CR0.PG = 1 and CR4.PAE = 1, the logical processor uses the **physical-address extension** (PAE). If, in addition, IA32_EFER.LMA = 0, the logical processor uses **PAE paging**. See Section 3.8 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.[1] When in PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTEs and, if they are valid, loads them into the processor (into internal, non-architectural registers).

A VM exit is to a VMM that uses PAE paging if (1) bit 5 (corresponding to CR4.PAE) is set in the CR4 field in the host-state area of the VMCS; and (2) the "host address-space size" VM-exit control is 0. Such a VM exit may check the validity of the PDPTEs referenced by the CR3 field in the host-state area of the VMCS. Such a VM exit must check their validity if either (1) PAE paging was not in use before the VM exit; or (2) the value of CR3 is changing as a result of the VM exit. A VM exit to a VMM that does not use PAE paging must not check the validity of the PDPTEs.

A VM exit that checks the validity of the PDPTEs uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use. If MOV to CR3 would cause a general-protection exception due to the PDPTEs that would be loaded (e.g., because a reserved bit is set), a VMX abort occurs (see Section 23.7). If a VM exit to a VMM that uses PAE does not cause a VMX abort, the PDPTEs are loaded into the processor as would MOV to CR3, using the value of CR3 being load by the VM exit.

23.5.5 Updating Non-Register State

VM exits affect the non-register state of a logical processor as follows:

- A logical processor is always in the active state after a VM exit.
- Event blocking is affected as follows:
  - There is no blocking by STI or by MOV SS after a VM exit.
  - VM exits caused directly by non-maskable interrupts (NMIs) cause blocking by NMI (see Table 20-3). Other VM exits do not affect blocking by NMI. (See Section 23.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

[1] On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

23.5.6 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 7.11.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. VM exits clear any address-range monitoring that may be in effect.

23.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 20.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101H (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be read only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM exits for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Appendix B.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.[1]

If processing fails for any entry, a VMX abort occurs. See Section 23.7.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM exit, the logical processor does not use any translations that were cached before the transition.

[1] Note the following about processors that support Intel 64 architecture. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CR0.PG is always 1 in VMX operation, the IA32_EFER MSR should not be included in the VM-exit MSR-load area for the purpose of modifying the LME bit.

23.7 VMX ABORTS

A problem encountered during a VM exit leads to a **VMX abort**. A VMX abort takes a logical processor into a shutdown state as described below.

A VMX abort does not modify the VMCS data in the VMCS region of any active VMCS. The contents of these data are thus suspect after the VMX abort.

On a VMX abort, a logical processor saves a nonzero 32-bit VMX-abort indicator field at byte offset 4 in the VMCS region of the VMCS whose misconfiguration caused the failure (see Section 20.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 23.4).
2. Host checking of the page-directory-pointer-table entries (PDPTEs) failed (see Section 23.5.4).
3. The current VMCS has been corrupted (through writes to the corresponding VMCS region) in such a way that the logical processor cannot complete the VM exit properly.
4. There was a failure on loading host MSRs (see Section 23.6).
5. There was a machine check during VM exit (see Section 23.8).
6. The logical processor was in IA-32e mode before the VM exit and the “host address-space size” VM-entry control was 0 (see Section 23.5).

Some of these causes correspond to failures during the loading of state from the host-state area. Because the loading of such state may be done in any order (see Section 23.5) a VM exit that might lead to a VMX abort for multiple reasons (for example, the current VMCS may be corrupt and the host PDPTEs might not be properly configured). In such cases, the VMX-abort indicator could correspond to any one of those reasons.

A logical processor never reads the VMX-abort indicator in a VMCS region and writes it only with one of the non-zero values mentioned above. The VMX-abort indicator allows software on one logical processor to diagnose the VMX-abort on another. For this reason, it is recommended that software running in VMX root operation zero the VMX-abort indicator in the VMCS region of any VMCS that it uses.

After saving the VMX-abort indicator, operation of a logical processor experiencing a VMX abort depends on whether the logical processor is in SMX operation:[1]

- If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000DH, indicating “VMX abort.” See *Intel® Trusted Execution Technology Measured Launched Environment Programming Guide*.

[1] A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

- If the logical processor is outside SMX operation, it issues a special bus cycle (to notify the chipset) and enters the **VMX-abort shutdown state**. RESET is the only event that wakes a logical processor from the VMX-abort shutdown state. The following events do not affect a logical processor in this state: machine checks; INIT signals; external interrupts; non-maskable interrupts (NMIs); start-up IPIs (SIPIs); and system-management interrupts (SMIs).

23.8 MACHINE CHECK DURING VM EXIT

If a machine check occurs during VM exit, one of the following occurs:

- The machine check is handled normally:
  - — If CR4.MCE = 1, a machine-check exception (#MC) delivered through the guest IDT.
  - — If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:[1]
    - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine check condition.” See *Intel® Trusted Execution Technology Measured Launched Environment Programming Guide*.
    - If the logical processor is outside SMX operation, it goes to the shutdown state.
- A VMX abort is generated (see Section 23.7). The logical processor blocks events as done normally in VMX abort. The VMX abort indicator is 5, for “machine check during VM exit.”

The first option is not used if the machine check occurs after any host state has been loaded.

[1] A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, “Safer Mode Extensions Reference,” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

CHAPTER 24
VIRTUAL PROCESSOR IDENTIFIERS (VPIDS)

The original architecture for VMX operation requires a VMX transition that changes to a new linear-address space (that change any of any of CR3[63:5], CR4.PAE, CR4.PSE, or IA32_EFER.LMA) to flush the TLBs and paging-structure caches (see Section 22.3.2.1 and Section 23.5.1). This ensures that translations cached for the old linear-address space are not used after the transition.

Virtual-processor identifiers (**VPIDs**) introduce to VMX operation a facility by which a logical processor may cache information for multiple linear-address spaces and thus eliminate the need for this flushing. The following items provide an overview of this facility:

- Software can specify the 16-bit VPID that is active at a particular point in time.
- The logical processor may cache information about the paging structures that are referenced (directly or indirectly) by the current value of CR3.[1] This information is associated with the currently active VPID.
- The logical processor may translate linear addresses using the cached information. It will use only information that is associated with the currently active VPID.

24.1 INFORMATION THAT MAY BE CACHED

A logical processor may cache information from the paging structures in a variety of different manners:

- It may cache, for specific linear addresses, the addresses to which they translate, along with information about access privileges and memory typing. (This is the caching done by TLBs.)
- It may cache, for the upper portion of a linear address, information derived from the paging-structure entries that are used to translate the corresponding region of the linear address space. For example, bits 47:39 of a linear address identifies an PML4 entry and could be cached with the contents of that entry. (This is the caching done by the paging-structure caches.)

Any cached information derived from paging structures is associated with the VPID that is active at the time the information is cached. A logical processor caches such information only for the paging structures referenced (directly or indirectly) by the current value of CR3.

[1] This chapter does not apply to information in the EPT paging structures. See Section 25.6 for details of how a logical processor may cache information from the EPT paging structures.

24.2 CURRENTLY ACTIVE VPID

The currently active VPID is a 16-bit value. It is 0000H in the following situations:

- Outside VMX operation. (This includes operation in system-management mode under the default treatment of SMIs and SMM with VMX operation; see Section 26.14.)
- In VMX root operation.
- In VMX non-root operation when the “enable VPID” VM-execution control is 0.

A non-zero VPID is thus active only in VMX non-root operation when the “enable VPID” VM-execution control is 1. At this time, the currently active VPID is the value of the VPID VM-execution control field in the VMCS.

24.3 INVALIDATION OF INFORMATION FROM THE PAGING STRUCTURES

Software modifications of paging structures may result in inconsistencies between those structures and information cached by a logical processor. Certain operations invalidate information cached by a logical processor and can be used to eliminate such inconsistencies.

The following operations invalidate information cached from the paging structures as indicated:

- VM entries and VM exits that occur when the “enable VPID” VM-execution control is 0 in the active VMCS invalidate information cached for VPID 0000H.
- Execution of the INVVPID instruction invalidates information cached from the paging structures. Invalidation is based on instruction operands, called the INVVPID type and the INVVPID descriptor. Four INVVPID types are currently defined:
  - — **Individual-address.** If the INVVPID type is 0, the logical processor invalidates information cached for the VPID specified in the INVVPID descriptor. The logical processor is only required to invalidate information from paging-structure entries used to translate the linear address specified in of the INVVPID descriptor. (The instruction may invalidate information cached for other VPIDs or from paging-structure entries used to translate other linear addresses.)
  - — **Single-context.** If the INVVPID type is 1, the logical processor invalidates all information cached for the VPID specified in the INVVPID descriptor. (The instruction may invalidate information cached for other VPIDs.)

- **All-context.** If the INVVPID type is 2, the logical processor invalidates all information cached for all VPIDs.
- **Single-context-retaining-globals.** If the INVVPID type is 3, the logical processor invalidates information cached for all the VPID specified in the INVVPID descriptor. The logical processor is **not** required to invalidate information that was used for **global** translations (although it may do so). See Section 3.12, “Translation Lookaside Buffers (TLBs)” for details regarding global translations. (The instruction may invalidate information cached for other VPIDs.)

See Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B* for details of the INVVPID instruction. See Section 24.4 for guidelines regarding use of this instruction.

- No information is cached from the paging structures for any VPID following power-up or a reset.
- Operations that architecturally invalidate entries in the TLBs or paging-structure caches (e.g., the INVLPG instruction) operate normally, but they are not guaranteed to invalidate information cached for VPIDs other than the currently active one.

If the “enable VPID” VM-execution control is 1 the active VMCS, VMX transitions are not required to invalidate information cached from the paging structures, even the transition is changing to a new linear-address space.

The VMXOFF and VMXON instructions do not necessarily invalidate information cached from the paging structures.

24.4 GUIDELINES FOR USE OF THE INVVPID INSTRUCTION

The need for VMM software to use the INVVPID instruction depends on how that software is virtualizing memory (e.g., see Section 28.3, “Memory Virtualization”). If the VMM is virtualizing the guest paging structures, certain operations that would invalidate the TLBs and the paging-structure caches (e.g., the INVLPG instruction) may be configured to cause VM exits. If VMM software is emulating these operations, it may be necessary to use the INVVPID instruction to ensure that the logical processor’s TLBs and the paging-structure caches are appropriately invalidated. (If EPT is being used—see Chapter 25—many uses of the INVVPID instruction may not be required.)

Requirements of when software should use the INVVPID instruction depend on the specific algorithm being used for page-table virtualization. The following items provide guidelines for software developers:

- Emulation of the INVLPG instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is individual-address (0).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.

  - The linear address in the INVVPID descriptor is that of the operand of the INVLPG instruction being emulated.

- Many operations invalidate all entries in the TLBs and paging-structure cases—except for global translations. An example is the MOV to CR3 instruction. (See Section 3.12, “Translation Lookaside Buffers (TLBs)” for details regarding global translations.) Emulation of such an operation may require execution of the INVVPID instruction as follows:
  - The INVVPID type is single-context-retaining-globals (3).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
- Some operations invalidate all entries in the TLBs and paging-structure cases—including for global translations. An example is the MOV to CR4 instruction if the value of value of bit 4 (page global enable—PGE) is changing. Emulation of such an operation may require execution of the INVVPID instruction as follows:
  - The INVVPID type is single-context (1).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.

If EPT is in use, a VMM may be emulating none of the operations enumerated above; in that case, the INVVPID executions identified would not be required.

The following guidelines apply more generally and are appropriate even if EPT is in use:

- As detailed in Section 21.2.1.1, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the paging structures. It is recommended that software use the INVVPID instruction with the “single-context” INVVPID type and with VPID X in the INVVPID descriptor if (1) the logical processor was in VMX non-root operation earlier with VPID X; and (2) since that time, either (a) the “virtualize APIC accesses” VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.
- Software can use the INVVPID instruction with the “all-context” INVVPID type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from paging structures between separate uses of VMX operation.

CHAPTER 25
EXTENDED PAGE TABLES (EPT)

The extended page-table mechanism (**EPT**) is a feature that can be used to support the virtualization of physical memory. When EPT is in use, certain physical addresses are treated as **guest-physical addresses** and are not used to access memory directly. Instead, guest-physical addresses are translated by traversing a set of **EPT paging structures** to produce physical addresses that are used to access memory.

- Section 25.1 gives an overview of EPT.
- Section 25.2 describes operation of EPT-based address translation.
- Section 25.3 gives details of the EPT paging structures.
- Section 25.4 discusses VM exits that may be caused by EPT.
- Section 25.5 describes interactions between EPT and memory typing.
- Section 25.6 discusses how logical processors may cache information from the EPT paging structures.

25.1 EPT OVERVIEW

EPT is used when the “enable EPT” VM-execution control is 1.[1] It translates the guest-physical addresses used in VMX non-root operation and those used by VM entry for event injection.

The translation from guest-physical addresses to physical addresses is determined by a set of **EPT paging structures**. The EPT paging structures are similar to those used to translate linear addresses while the processor is in IA-32e mode. Details of the EPT paging structures are given in Section 25.3. While the “enable EPT” VM-execution control is 1, the term **guest paging structures** refers to those paging structures referenced through control register CR3.

When the “enable EPT” VM-execution control is 1, **guest-physical addresses** are those derived from the contents of control register CR3 and the guest paging structures. (This includes the values of the PDPTEs, which logical processors store in internal, non-architectural registers.) Note that the latter includes (in page-table entries and in page-directory entries for which bit 7—PS—is 1) the addresses to which linear addresses are translated by the guest paging structures.

[1] Note that “enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, the logical processor operates as if the “enable EPT” VM-execution control were 0. See Section 20.6.2.

A logical processor uses EPT to translate guest-physical addresses only when those addresses are used to access memory. This has implications for the MOV to CR3 instruction, which loads CR3 with a guest-physical address:

- If PAE paging is not being used, the MOV to CR3 instruction does not use the guest-physical address to access memory.[1] Thus, the instruction does **not** cause that address to be translated through EPT. The address will be translated through EPT on the next memory accessing using a linear address.
- If PAE paging is being used, the MOV to CR3 instruction loads the four (4) page-directory-pointer-table entries (PDPTEs) from the guest-physical address. Thus, the instruction **does** cause its address to be translated through EPT.

  The PDPTEs that are loaded also contain guest-physical addresses. The MOV to CR3 instruction does not use these addresses to access memory. Thus, the instruction does **not** cause these addresses to be translated through EPT. The guest-physical address in an PDPTE will be translated through EPT on the next memory accessing using a linear address that uses that PDPTE.

The translation of a linear address to a physical address requires multiple translations of guest-physical addresses using EPT. Assume, for example, that CR0.PG = 1 and CR4.PAE = CR4.PSE = 0. The translation of a 32-bit linear address then operates as follows:

- Bits 31:22 of the linear address select an entry in the guest page directory located at the guest-physical address in CR3. The guest-physical address of the guest page-directory entry (PDE) is translated through EPT to determine the guest PDE's physical address.
- Bits 21:12 of the linear address select an entry in the guest page table located at the guest-physical address in the guest PDE. The guest-physical address of the guest page-table entry (PTE) is translated through EPT to determine the guest PTE's physical address.
- Bits 11:0 of the linear address is the offset in the page frame located at the guest-physical address in the guest PTE. The guest-physical address determined by this offset is translated through EPT to determine the physical address to which the original linear address translates.

In addition to translating a guest-physical address to a physical address, EPT specifies the privileges that software is allowed when accessing the address. Attempts at disallowed accesses are called **EPT violations** and cause VM exits. See Section 25.4.

[1] A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 3.8 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

25.2 EPT TRANSLATION MECHANISM

The EPT translation mechanism uses only bits 47:0 of each guest-physical address.[1] It uses a page-walk length of 3, meaning that at most 3 EPT paging-structure entries are accessed to locate a (fourth) EPT paging-structure entry that contains the physical address to which the original guest-physical address translates.[2]

These 48 bits are partitioned by the logical processor to traverse the EPT paging structures:

- A 4-KByte naturally aligned EPT PML4 table is located a physical address specified in bits 51:12 of the **extended-page-table pointer** (EPTP), a VM-execution control field (see Table 20-8 in Section 20.6.11). An EPT PML4 entry is selected from this table using a physical address defined as follows:
  - Bits 63:52 are all 0.
  - Bits 51:12 are from the EPTP.
  - Bits 11:3 are bits 47:39 of the guest-physical address.
  - Bits 2:0 are all 0.
- A 4-KByte naturally aligned EPT page-directory-pointer table is located a physical address specified in bits 51:12 of the EPT PML4 entry (see Table 25-1). An EPT page-directory-pointer-table entry (PDPTE) is selected from this table using a physical address defined as follows:
  - Bits 63:52 are all 0.
  - Bits 51:12 are from the EPT PML4 entry.
  - Bits 11:3 are bits 38:30 of the guest-physical address.
  - Bits 2:0 are all 0.
- A 4-KByte naturally aligned EPT page directory is located a physical address specified in bits 51:12 of the EPT PDPTE (see Table 25-2). An EPT page-directory entry (PDE) is selected from this table using a physical address defined as follows:
  - Bits 63:52 are all 0.
  - Bits 51:12 are from the EPT PDPTE.
  - Bits 11:3 are bits 29:21 of the guest-physical address.

[1] No processors supporting the Intel 64 architecture support more than 48 physical-address bits. Thus, no such processor can produce a guest-physical address with more than 48 bits. An attempt to use such an address will cause a page fault. An attempt to load CR3 with such an address causes a general-protection fault. If PAE paging is being used, an attempt to load CR3 that would load a PDPTE with such an address causes a general-protection fault.

[2] Future processors may include support for other EPT page-walk lengths. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix G.10) to determine what EPT page-walk lengths are supported.

- Bits 2:0 are all 0.

Use of the EPT PDE depends on the value of bit 7 in that entry:

- If bit 7 of the EPT PDE is 0:
  - A 4-KByte naturally aligned EPT page table is located a physical address specified in bits 51:12 of the EPT PDE (see Table 25-3). An EPT page-table entry (PTE) is selected from this table using a physical address defined as follows:
    - Bits 63:52 are all 0.
    - Bits 51:12 are from the EPT PDE.
    - Bits 11:3 are bits 20:12 of the guest-physical address.
    - Bits 2:0 are all 0.
  - The final physical address is computed as follows:
    - Bits 63:52 are all 0.
    - Bits 51:12 are from the EPT PTE (see Table 25-5).
    - Bits 11:0 are from the original guest-physical address.
- If bit 7 of the EPT PDE is 1, the final physical address is computed as follows:
  - Bits 63:52 are all 0.
  - Bits 51:21 are from the EPT PDE (see Table 25-4).
  - Bits 20:0 are from the original guest-physical address.

If bits 2:0 of an EPT paging-structure entry are all 0, the entry is **not present** and is used neither to reference another EPT paging-structure entry nor to produce a physical address. A reference using a guest-physical address whose translation encounters an EPT paging-structure that is not present causes an EPT violation.

The discussion above describes how the EPT paging structures reference each other and how the logical processor traverses those structures when translating a guest-physical address. It does not cover all details of the translation process. Addition details are provided as follows:

- Details of the EPT paging-structure entries are given in Section 25.3.
- Situations in which the translation process may lead to VM exits (sometimes before the process completes) are described in Section 25.4.
- Interactions between the EPT translation mechanism and memory typing are described in Section 25.5.

25.3 EPT PAGING STRUCTURES

Each EPT paging structure is 4 KBytes in size and comprises 512 8-Byte entries. This section specifies the formats of these entries.

25.3.1 EPT PML4 Entries

An EPT PML4 entry is identified using bits 47:39 of the guest-physical address (see Section 25.2) and thus controls access to a 512-Gbyte region of the guest-physical address space. Table 25-1 shows the format of an EPT PML4 entry.

Table 25-1. Format of an EPT PML4 Entry

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 512-GByte region controlled by this entry
1	Write access; indicates whether writes are allowed from the 512-GByte region controlled by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 512-GByte region controlled by this entry
7:3	Reserved (must be 0)
11:8	Ignored
N–1:12	Physical address of 4-KByte aligned EPT page-directory-pointer table referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

[1] N is the physical-address width supported by the processor. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

Note that, if bits 2:0 of an EPT PML4 entry are all 0, the entry is considered to be "not present"; the logical processor ignores bits 63:3 of such an entry and will not use it to reference an EPT page-directory-pointer table.

25.3.2 EPT Page-Directory-Pointer-Table Entries

An EPT page-directory-pointer-table entry (PDPTE) is identified using bits 47:30 of the guest-physical address (see Section 25.2) and thus controls access to a 1-Gbyte region of the guest-physical address space. Table 25-2 shows the format of an EPT PDPTE.

Table 25-2. Format of an EPT Page-Directory-Pointer-Table Entry (PDPTE)

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 1-GByte region controlled by this entry
1	Write access; indicates whether writes are allowed from the 1-GByte region controlled by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 1-GByte region controlled by this entry
7:3	Reserved (must be 0)
11:8	Ignored
N–1:12	Physical address of 4-KByte aligned EPT page directory referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

[1] N is the physical-address width supported by the logical processor.

Note that, if bits 2:0 of an EPT PDPTE are all 0, the entry is considered to be “not present”; the logical processor ignores bits 63:3 of such an entry and will not use it to reference an EPT page directory.

25.3.3 EPT Page-Directory Entries

An EPT page-directory entry (PDE) is identified using bits 47:21 of the guest-physical address (see Section 25.2) and thus controls access to a 2-Mbyte region of the guest-physical address space. It may do so either by referencing an EPT page table or by mapping a single 2-Mbyte page; the value of bit 7 in the EPT PDE determines which mechanism is used.

- The value of bits 2:0 of an EPT paging-structure entry is 100b (execute-only) and this value is not supported by the logical processor. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP to determine whether this value is supported (see Appendix G.10).

If bit 7 of the EPT PDE is 0, the entry references an EPT page table. Table 25-3 shows the format of such an EPT PDE.

Table 25-3. Format of an EPT Page-Directory Entry that References an EPT Page Table

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 2-MByte region controlled by this entry
1	Write access; indicates whether writes are allowed from the 2-MByte region controlled by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 2-MByte region controlled by this entry
6:3	Reserved (must be 0)
7	Must be 0 (otherwise, this entry maps a 2-MByte page)
11:8	Ignored
N–1:12	Physical address of 4-KByte aligned EPT page table referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

[1] N is the physical-address width supported by the logical processor.

If bit 7 of the EPT PDE is 1, the entry maps a 2-MByte page. Table 25-4 shows the format of such an EPT PDE.

Table 25-4. Format of an EPT Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 2-MByte page referenced by this entry
1	Write access; indicates whether writes are allowed from the 2-MByte page referenced by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 2-MByte page referenced by this entry

Table 25-4. Format of an EPT Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
5:3	EPT memory type for this 2-MByte page
6	Ignore PAT memory type for this 2-MByte page (see Section 25.5)
7	Must be 1 (otherwise, this entry references an EPT page table)
11:8	Ignored
20:12	Reserved (must be 0)
N–1:21	Physical address of the 2-MByte page referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the logical processor.

Note that, if bits 2:0 of an EPT PDE are all 0, the entry is considered to be "not present"; the logical processor ignores bits 63:3 (including bit 7) of such an entry and will use it neither to reference an EPT page table nor to map a 2-MByte page.

25.3.4 EPT Page-Table Entries

An EPT page-table entry (PTE) is identified using bits 47:12 of the guest-physical address (see Section 25.2) and thus maps a 4-Kbyte page. Table 25-3 shows the format of an EPT PDE.

Table 25-5. Format of an EPT Page-Table Entry

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 4-KByte page referenced by this entry
1	Write access; indicates whether writes are allowed from the 4-KByte page referenced by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 4-KByte page referenced by this entry
5:3	EPT memory type for this 4-KByte page

Table 25-5. Format of an EPT Page-Table Entry (Contd.)

Bit Position(s)	Contents
6	Ignore PAT memory type for this 2-MByte page (see Section 25.5)
11:7	Ignored
N–1:12	Physical address of the 4-KByte page referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the logical processor.

Note that, if bits 2:0 of an EPT PTE are all 0, the entry is considered to be "not present"; the logical processor ignores bits 63:3 of such an entry and will not use it to map a 4-KByte page.

25.4 EPT-INDUCED VM EXITS

Accesses using guest-physical addresses may cause VM exits due to **EPT misconfigurations** and **EPT violations**. An EPT misconfiguration occurs when, in the course of translation a guest-physical address, the logical processor encounters an EPT paging-structure entry that contains an unsupported value. An EPT violation occurs when there is no EPT misconfiguration but the EPT paging-structure entries disallow an access using the guest-physical address.

Note that EPT misconfigurations and EPT violations occur only due to an attempt to access memory with a guest-physical address. Loading CR3 with a guest-physical address with the MOV to CR3 instruction can cause neither an EPT configuration nor an EPT violation until that address is used to access a paging structure.[1]

25.4.1 EPT Misconfigurations

AN EPT misconfiguration occurs if any of the following is identified while translating a guest-physical address:

[1] If the logical processor is using PAE paging—because CR4.PAE = 1 and IA32_EFER.LMA = 0—the MOV to CR3 instruction loads the PDPTEs from memory using the guest-physical address being loaded into CR3. In this case, therefore, the MOV to CR3 instruction may cause an EPT misconfiguration or an EPT violation.

- The value of bits 2:0 of an EPT paging-structure entry is either 010b (write-only) or 110b (write/execute).
- The value of bits 2:0 of an EPT paging-structure entry is 100b (execute-only) and this value is not supported by the logical processor. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP to determine whether this value is supported (see Appendix G.10).
- The value of bits 2:0 of an EPT paging-structure entry is not 000b (the entry is present) and one of the following holds:
  - A reserved bit is set. This includes the setting of a bit in the range 51:12 that is beyond the logical processor's physical-address width.[1] See Section 25.3 for details of which bits are reserved in which EPT paging-structure entries.
  - The entry is the last one used to translate a guest physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE) and the value of bits 5:3 (EPT memory type) is 2, 3, or 7 (these values are reserved).

25.4.2 EPT Violations

An EPT violation may occur during an access using a guest-physical address whose translation does not cause an EPT misconfiguration. An EPT violation occurs in any of the following situations:

- Translation of the guest-physical address encounters an EPT paging-structure entry that is not present (see Section 25.2).
- The access is a data read and bit 0 was clear in any of the EPT paging-structure entries used to translate the guest-physical address. Reads by the logical processor of guest paging structures to translate a linear address are considered to be data reads.
- The access is a data write and bit 1 was clear in any of the EPT paging-structure entries used to translate the guest-physical address. Writes by the logical processor to guest paging structures to update accessed and dirty flags are considered to be data writes.
- The access is an instruction fetch and bit 2 was clear in any of the EPT paging-structure entries used to translate the guest-physical address.

25.4.3 Prioritization of EPT-Induced VM Exits

The translation of a linear address to a physical address requires multiple translations of guest-physical addresses using EPT (see Section 25.1). This section specifies the relative priority of EPT-induced VM exits with respect to each other and to other events that may be encountered when accessing memory using a linear address.

[1] Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

For an access to a guest-physical address, determination of whether an EPT misconfiguration or an EPT violation occurs is based on an iterative process:[1]

1. An EPT paging-structure entry is read (initially, this is an EPT PML4 entry):
   a. If the entry is not present (bits 2:0 are all 0), an EPT violation occurs.
   b. If the entry is present but its contents are not configured properly (see Section 25.4.1), an EPT misconfiguration occurs.
   c. If the entry is present and its contents are configured properly, operation depends on whether the entry references another EPT paging structure (whether it is an EPT PDE with bit 7 set to 1 or an EPT PTE):
      i) If the entry does references another EPT paging structure, an entry from that structure is accessed; step 1 is executed for that other entry.
      ii) Otherwise, the entry is used to produce the ultimate physical address (the translation of the original guest-physical address); step 2 is executed.
2. Once the ultimate physical address is determined, the privileges determined by the EPT paging-structure entries are evaluated:
   a. If the access to the guest-physical address is not allowed by these privileges (see Section 25.4.2), an EPT violation occurs.
   b. If the access to the guest-physical address is allowed by these privileges, memory is accessed using the ultimate physical address.

The translation of a linear address is also an iterative process, with the processor first accessing a guest paging-structure entry referenced by the guest-physical address in CR3, then accessing another guest paging-structure entry referenced by the guest-physical address in the first guest paging-structure entry, etc. Each guest-physical address is itself translated using EPT and may cause an EPT-induced VM exit. The following items detail how page faults and EPT-induced VM exits are recognized during this iterative process:

1. An attempt is made to access a guest paging-structure entry with a guest-physical address (initially, the address is derived from the one in CR3).
   a. If the access fails because of an EPT misconfiguration or an EPT violation (see above), an EPT-induced VM exit occurs.
   b. If the access does not cause an EPT-induced VM exit, bit 0 (present) of the entry is consulted:
      i) If the present flag is 0 or any reserved bit is set, a page fault occurs.
      ii) If the present flag is 1, no reserved bit is set, operation depends on whether the entry references another guest paging-structure entry (whether it is a guest PDE with PS = 1 or a guest PTE):

[1] This is a simplification of the more detailed description given in Section 25.2.

   - If the entry does references another guest paging structure, an entry from that structure is accessed; step 1 is executed for that other entry.
   - Otherwise, the entry is used to produce the ultimate guest-physical address (the translation of the original linear address); step 2 is executed.

2. Once the ultimate guest-physical address is determined, the privileges determined by the guest paging-structure entries are evaluated:

   a. If the access to the linear address is not allowed by these privileges (e.g., it was a write to a read-only page), a page fault occurs.

   b. If the access to the linear address is allowed by these privileges, an attempt is made to access memory at the ultimate guest-physical address:

      i) If the access fails because of an EPT misconfiguration or an EPT violation (see above), an EPT-induced VM exit occurs.

      ii) If the access does not cause an EPT-induced VM exit, memory is accessed using the ultimate physical address (the translation, using EPT, of the ultimate guest-physical address).

25.5 EPT AND MEMORY TYPING

This section specifies how a logical processor determines the memory type use for a memory access while EPT is in use. (See Chapter 10, “Memory Cache Control” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A* for details of memory typing in the Intel 64 architecture.) Section 25.5.1 explains how the memory type is determined for accesses to the EPT paging structures. Section 25.5.2 explains how the memory type is determined for an access using a guest-physical address that is translated using EPT.

25.5.1 Memory Type Used for Accessing EPT Paging Structures

This section explains how the memory type is determined for accesses to the EPT paging structures. The determination is based first on the value of bit 30 (cache disable—CD) in control register CR0:

- If CR0.CD = 0, the memory type used for any such reference is the EPT paging-structure memory type, which is specified in bits 2:0 of the extended-page-table pointer (EPTP), a VM-execution control field (see Section 20.6.11). A value of 0 indicates the uncacheable type (UC), while a value of 6 indicates the write-back type (WB). Other values are reserved.
- If CR0.CD = 1, the memory type used for any such reference is uncacheable (UC).

25.5.2 Memory Type Used for Translated Guest-Physical Addresses

The **effective memory type** of a memory access using a guest-physical address (an access that is translated using EPT) is the memory type that is used to address memory. The effective memory type is based on the value of bit 30 (cache disable—CD) in control register CR0; the **last** EPT paging-structure entry used to translate the guest-physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE); and the PAT memory type (see below):

- The **PAT memory type** is the memory type selected from the IA32_PAT MSR as specified in Section 10.12.3, "Selecting a Memory Type from the PAT".[1]
- The **EPT memory type** is specified in bits 5:3 of the last EPT paging-structure entry: 0 = UC; 1 = WC; 4 = WT; 5 = WP; and 6 = WB. Other values are reserved and will cause an EPT misconfiguration (see Section 25.4).
- If CR0.CD = 0, the effective memory type depends upon the value of bit 6 of the last EPT paging-structure entry:
  - If the value is 0, the effective memory type is the combination of the EPT memory type and the PAT memory type specified in Table 10-7 in Section 10.5.2.2, using the EPT memory type in place of the MTRR memory type.
  - If the value is 1, the memory type used for the access is the EPT memory type. The PAT memory type is ignored.
- If CR0.CD = 1, the effective memory type is UC.

Note that the MTRRs have no effect on the memory type used for an access to a guest-physical address.

25.6 CACHING INFORMATION FROM THE EPT PAGING STRUCTURES

The Intel 64 architecture allows logical processors to cache information about the translation of linear addresses in translation lookaside buffers (TLBs) and paging-structure caches. (For more details, see the application note "*TLBs, Paging-Structure Caches, and Their Invalidation*." This section describes how a logical processor may cache information from the EPT paging structures.

[1] Table 10-11 in Section 10.12.3, "Selecting a Memory Type from the PAT" illustrates how the PAT memory type is selected based on the values of the PAT, PCD, and PWT bits in a page-table entry (or page-directory entry with PS = 1). For accesses to a guest paging-structure entry X, the PAT memory type is selected from the table by using a value of 0 for the PAT bit with the values of PCD and PWT from the paging-structure entry Y that references X (or from CR3 if X is in the root paging structure).

25.6.1 Information That May Be Cached

A logical processor may cache information from the EPT paging structures in a variety of different manners:

- It may cache, for specific guest-physical addresses, the physical addresses to which they translate, along with information about access privileges and memory typing. Such caching corresponds to what existing TLBs do for linear addresses.
- It may cache, for specific linear addresses, the physical addresses to which they translate, along with information about access privileges and memory typing. Such caching includes information from the EPT paging structures because each linear address is translated first to a guest-physical address, which is then translated (using EPT) to a physical address.
- It may cache, for the upper portion of a guest-physical address, information derived from the EPT paging-structure entries that are used to translate the corresponding region of the guest-physical address space. (For example, bits 47:39 of a guest-physical address identifies an EPT PML4 entry and could be cached with the contents of that entry.) This corresponds to use of existing paging-structure caches.

Any cached information derived from EPT paging structures is associated with the EPT pointer (EPTP) that references those structures. A logical processor caches such information only when EPT is in use, and it will use such information only when EPT is use, and it will use only information associated with the current EPTP value.

A logical processor does **not** cache any information derived from EPT paging-structure entries that are not present (bits 2:0 are all 0) or misconfigured (see Section 25.4.1). For example, if an EPT PTE is not present, a logical processor will not cache that information, nor will it cache translation information for any linear address that translates to such a guest-physical address whose translation would use that EPT PTE.

25.6.2 Invalidation of Information from the EPT Paging Structures

Software modifications of EPT paging structures may result in inconsistencies between those structures and information cached by a logical processor. Certain operations invalidate information cached by a logical processor and can be used to eliminate such inconsistencies.

The following operations invalidate information cached from the EPT paging structures as indicated:

- Execution of the INVEPT instruction invalidates information cached from the EPT paging structures. Invalidation is based on instruction operands, called the INVEPT type and the INVEPT descriptor. Two INVEPT types are currently defined:
  - **Single-context.** If the INVEPT type is 1, the logical processor invalidates all information cached from EPT paging structures referenced by the EPTP value specified in of the INVEPT descriptor. (The instruction may invalidate

information cached from EPT paging structures referenced by other EPTP values.)

  - **All-context.** If the INVEPT type is 2, the logical processor invalidates all information cached from EPT paging structures (for all EPTP values).

  See Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B* for details of the INVEPT instruction. See Section 25.6.3 for guidelines regarding use of this instruction.

- No information is cached from the EPT paging structures following power-up or a reset.
- EPT violations invalidate information is cached from the EPT paging structures to a limited extent. Specifically, the logical processor ensures that, if the memory access that caused an EPT violation is executed again after the EPT violation, it will not use any information that was cached from the EPT paging structures before the EPT violation. (EPT violations may invalidate more information than is required to satisfy this guarantee.)

The following operations do not necessarily invalidate information cached from the EPT paging structures: VM entries, VM exits, and executions of the INVLPG, MOV CR, VMXOFF, and VMXON instructions.[1]

25.6.3 Guidelines for Use of the INVEPT Instruction

The following items provide guidelines for use of the INVEPT instruction to invalidate information cached from the EPT paging structures.

- Software should use the INVEPT instruction with the “single-context” INVEPT type after making any of the following changes to an EPT paging-structure entry (the INVEPT descriptor should be set to the EPTP value that references the modified EPT paging structure):
  - Changing any of the privilege bits 2:0 from 1 to 0.
  - Changing the physical address in bits 51:12.
  - For an EPT PDE, changing bit 7 (which determines whether the EPT PDE maps a 2-MByte page).
  - For the **last** EPT paging-structure entry used to translate a guest-physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE), changing either bits 5:3 or bit 6. (These bits determine the effective memory of accesses using that EPT paging-structure entry; see Section 25.5.)
- Software may use the INVEPT instruction after modifying an EPT paging-structure entry to change one of the privilege bits 2:0 from 0 to 1. Failure to do

[1] Although not required to do so, these operations may invalidate some information from EPT paging structures. For example, the INVLPG instruction may invalidate information from the EPT paging structures that were used to translate the guest-physical address to which the instruction's linear-address operand translates.

so may cause an EPT violation that would not occur had the cached information been consistent with the modified EPT paging-structure entry. As noted in Section 25.6.2, the EPT violation will not recur if the original access is performed again, even if the INVEPT instruction is not executed after the EPT violation.

- Because a logical processor does not cache any information derived from EPT paging-structure entries that are not present or misconfigured (see Section 25.6.1), it is not necessary to execute INVEPT following modification of an EPT paging-structure entry that had been not present or misconfigured.
- As detailed in Section 21.2.1.1 and Section 21.2.2.1, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the EPT paging structures. It is recommended that software use the INVEPT instruction with the “single-context” INVEPT type with value X for the INVEPT descriptor if (1) EPT was in use earlier with EPTP value X; and (2) since that time, either (a) the “virtualize APIC accesses” VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.
- Software can use the INVEPT instruction with the “all-context” INVEPT type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from EPT paging structures between separate uses of VMX operation.

In a system containing more than one logical processor, software must account for the fact that information from an EPT paging-structure entry may be cached on logical processors other than the one that modifies that entry. The process of propagating the changes to a paging-structure entry is commonly referred to as “TLB shootdown.” A discussion of TLB shootdown appears in Section 9 of the application note “*TLBs, Paging-Structure Caches, and Their Invalidation*.”

CHAPTER 26 SYSTEM MANAGEMENT

This chapter describes aspects of IA-64 and IA-32 architecture used in system management mode (SMM).

SMM provides an alternate operating environment that can be used to monitor and manage various system resources for more efficient energy usage, to control system hardware, and/or to run proprietary code. It was introduced into the IA-32 architecture in the Intel386 SL processor (a mobile specialized version of the Intel386 processor). It is also available in the Pentium M, Pentium 4, Intel Xeon, P6 family, and Pentium and Intel486 processors (beginning with the enhanced versions of the Intel486 SL and Intel486 processors).

26.1 SYSTEM MANAGEMENT MODE OVERVIEW

SMM is a special-purpose operating mode provided for handling system-wide functions like power management, system hardware control, or proprietary OEM-designed code. It is intended for use only by system firmware, not by applications software or general-purpose systems software. The main benefit of SMM is that it offers a distinct and easily isolated processor environment that operates transparently to the operating system or executive and software applications.

When SMM is invoked through a system management interrupt (SMI), the processor saves the current state of the processor (the processor's context), then switches to a separate operating environment contained in system management RAM (SMRAM). While in SMM, the processor executes SMI handler code to perform operations such as powering down unused disk drives or monitors, executing proprietary code, or placing the whole system in a suspended state. When the SMI handler has completed its operations, it executes a resume (RSM) instruction. This instruction causes the processor to reload the saved context of the processor, switch back to protected or real mode, and resume executing the interrupted application or operating-system program or task.

The following SMM mechanisms make it transparent to applications programs and operating systems:

- The only way to enter SMM is by means of an SMI.
- The processor executes SMM code in a separate address space (SMRAM) that can be made inaccessible from the other operating modes.
- Upon entering SMM, the processor saves the context of the interrupted program or task.

- All interrupts normally handled by the operating system are disabled upon entry into SMM.
- The RSM instruction can be executed only in SMM.

SMM is similar to real-address mode in that there are no privilege levels or address mapping. An SMM program can address up to 4 GBytes of memory and can execute all I/O and applicable system instructions. See Section 26.5 for more information about the SMM execution environment.

NOTES

The physical address extension (PAE) mechanism introduced in the P6 family processors is not supported when a processor is in SMM.

The IA-32e mode address-translation mechanism is not supported in SMM. See Section 3.10 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

26.1.1 System Management Mode and VMX Operation

Traditionally, SMM services system management interrupts and then resumes program execution (back to the software stack consisting of executive and application software; see Section 26.2 through Section 26.13).

A virtual machine monitor (VMM) using VMX can act as a host to multiple virtual machines and each virtual machine can support its own software stack of executive and application software. On processors that support VMX, virtual-machine extensions may use system-management interrupts (SMIs) and system-management mode (SMM) in one of two ways:

- **Default treatment.** System firmware handles SMIs. The processor saves architectural states and critical states relevant to VMX operation upon entering SMM. When the firmware completes servicing SMIs, it uses RSM to resume VMX operation.
- **Dual-monitor treatment.** Two VM monitors collaborate to control the servicing of SMIs: one VMM operates outside of SMM to provide basic virtualization in support for guests; the other VMM operates inside SMM (while in VMX operation) to support system-management functions. The former is referred to as **executive monitor**, the latter **SMM monitor**.[1]

The default treatment is described in Section 26.14, “Default Treatment of SMIs and SMM with VMX Operation and SMX Operation”. Dual-monitor treatment of SMM is described in Section 26.15, “Dual-Monitor Treatment of SMIs and SMM”.

[1] The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix G.1) to determine whether it is supported.

26.2 SYSTEM MANAGEMENT INTERRUPT (SMI)

The only way to enter SMM is by signaling an SMI through the SMI# pin on the processor or through an SMI message received through the APIC bus. The SMI is a nonmaskable external interrupt that operates independently from the processor's interrupt- and exception-handling mechanism and the local APIC. The SMI takes precedence over an NMI and a maskable interrupt. SMM is non-reentrant; that is, the SMI is disabled while the processor is in SMM.

NOTES

In the Pentium 4, Intel Xeon, and P6 family processors, when a processor that is designated as an application processor during an MP initialization sequence is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked. However if a SMI is received while an application processor is in the wait for SIPI mode, the SMI will be pended. The processor then responds on receipt of a SIPI by immediately servicing the pended SMI and going into SMM before handling the SIPI.

An SMI may be blocked for one macroinstruction following an STI, MOVSS or POPSS.

26.3 SWITCHING BETWEEN SMM AND THE OTHER PROCESSOR OPERATING MODES

Figure 2-3 shows how the processor moves between SMM and the other processor operating modes (protected, real-address, and virtual-8086). Signaling an SMI while the processor is in real-address, protected, or virtual-8086 modes always causes the processor to switch to SMM. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.

26.3.1 Entering SMM

The processor always handles an SMI on an architecturally defined "interruptible" point in program execution (which is commonly at an IA-32 architecture instruction boundary). When the processor receives an SMI, it waits for all instructions to retire and for all stores to complete. The processor then saves its current context in SMRAM (see Section 26.4), enters SMM, and begins to execute the SMI handler.

Upon entering SMM, the processor signals external hardware that SMM handling has begun. The signaling mechanism used is implementation dependent. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is asserted each time a bus transaction is generated while the processor is in SMM. For the Pentium and Intel486 processors, the SMIACT# pin is asserted.

An SMI has a greater priority than debug exceptions and external interrupts. Thus, if an NMI, maskable hardware interrupt, or a debug exception occurs at an instruction boundary along with an SMI, only the SMI is handled. Subsequent SMI requests are not acknowledged while the processor is in SMM. The first SMI interrupt request that occurs while the processor is in SMM (that is, after SMM has been acknowledged to external hardware) is latched and serviced when the processor exits SMM with the RSM instruction. The processor will latch only one SMI while in SMM.

See Section 26.5 for a detailed description of the execution environment when in SMM.

26.3.2 Exiting From SMM

The only way to exit SMM is to execute the RSM instruction. The RSM instruction is only available to the SMI handler; if the processor is not in SMM, attempts to execute the RSM instruction result in an invalid-opcode exception (#UD) being generated.

The RSM instruction restores the processor's context by loading the state save image from SMRAM back into the processor's registers. The processor then returns an SMIACK transaction on the system bus and returns program control back to the interrupted program.

Upon successful completion of the RSM instruction, the processor signals external hardware that SMM has been exited. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is no longer generated on bus cycles. For the Pentium and Intel486 processors, the SMIACT# pin is deserted.

If the processor detects invalid state information saved in the SMRAM, it enters the shutdown state and generates a special bus cycle to indicate it has entered shutdown state. Shutdown happens only in the following situations:

- A reserved bit in control register CR4 is set to 1 on a write to CR4. This error should not happen unless SMI handler code modifies reserved areas of the SMRAM saved state map (see Section 26.4.1). Note that CR4 is saved in the state map in a reserved location and cannot be read or modified in its saved state.
- An illegal combination of bits is written to control register CR0, in particular PG set to 1 and PE set to 0, or NW set to 1 and CD set to 0.
- (For the Pentium and Intel486 processors only.) If the address stored in the SMBASE register when an RSM instruction is executed is not aligned on a 32-KByte boundary. This restriction does not apply to the P6 family processors.

In the shutdown state, Intel processors stop executing instructions until a RESET#, INIT# or NMI# is asserted. While Pentium family processors recognize the SMI# signal in shutdown state, P6 family and Intel486 processors do not. Intel does not support using SMI# to recover from shutdown states for any processor family; the response of processors in this circumstance is not well defined. On Pentium 4 and later processors, shutdown will inhibit INTR and A20M but will not change any of the other inhibits. On these processors, NMIs will be inhibited if no action is taken in the SMM handler to uninhibit them (see Section 26.8).

If the processor is in the HALT state when the SMI is received, the processor handles the return from SMM slightly differently (see Section 26.10). Also, the SMBASE address can be changed on a return from SMM (see Section 26.11).

26.4 SMRAM

While in SMM, the processor executes code and stores data in the SMRAM space. The SMRAM space is mapped to the physical address space of the processor and can be up to 4 GBytes in size. The processor uses this space to save the context of the processor and to store the SMI handler code, data and stack. It can also be used to store system management information (such as the system configuration and specific information about powered-down devices) and OEM-specific information.

The default SMRAM size is 64 KBytes beginning at a base physical address in physical memory called the SMBASE (see Figure 26-1). The SMBASE default value following a hardware reset is 30000H. The processor looks for the first instruction of the SMI handler at the address [SMBASE + 8000H]. It stores the processor's state in the area from [SMBASE + FE00H] to [SMBASE + FFFFH]. See Section 26.4.1 for a description of the mapping of the state save area.

The system logic is minimally required to decode the physical address range for the SMRAM from [SMBASE + 8000H] to [SMBASE + FFFFH]. A larger area can be decoded if needed. The size of this SMRAM can be between 32 KBytes and 4 GBytes.

The location of the SMRAM can be changed by changing the SMBASE value (see Section 26.11). It should be noted that all processors in a multiple-processor system are initialized with the same SMBASE value (30000H). Initialization software must sequentially place each processor in SMM and change its SMBASE so that it does not overlap those of other processors.

The actual physical location of the SMRAM can be in system memory or in a separate RAM memory. The processor generates an SMI acknowledge transaction (P6 family processors) or asserts the SMIACT# pin (Pentium and Intel486 processors) when the processor receives an SMI (see Section 26.3.1).

System logic can use the SMI acknowledge transaction or the assertion of the SMIACT# pin to decode accesses to the SMRAM and redirect them (if desired) to specific SMRAM memory. If a separate RAM memory is used for SMRAM, system logic should provide a programmable method of mapping the SMRAM into system memory space when the processor is not in SMM. This mechanism will enable start-up procedures to initialize the SMRAM space (that is, load the SMI handler) before executing the SMI handler during SMM.

26.4.1 SMRAM State Save Map

When an IA-32 processor that does not support Intel 64 architecture initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area begins at [SMBASE + 8000H + 7FFFH] and extends down to [SMBASE + 8000H + 7E00H]. Table 26-1 shows the state save map. The offset in column 1 is relative to the SMBASE value plus 8000H. Reserved spaces should not be used by software.

Some of the registers in the SMRAM state save area (marked YES in column 3) may be read and changed by the SMI handler, with the changed values restored to the processor registers by the RSM instruction. Some register images are read-only, and must not be modified (modifying these registers will result in unpredictable behavior). An SMI handler should not rely on any values stored in an area that is marked as reserved.

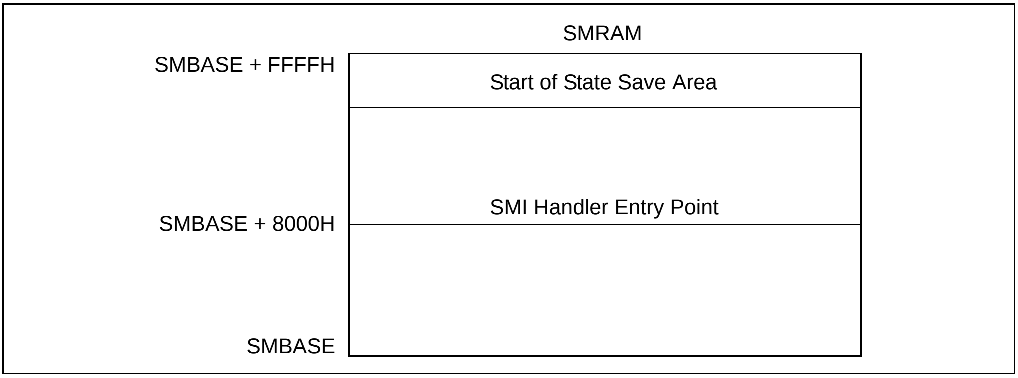

Figure 26-1. SMRAM Usage

Table 26-1. SMRAM State Save Map

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FFCH	CR0	No
7FF8H	CR3	No
7FF4H	EFLAGS	Yes
7FF0H	EIP	Yes
7FECH	EDI	Yes
7FE8H	ESI	Yes
7FE4H	EBP	Yes
7FE0H	ESP	Yes

Table 26-1. SMRAM State Save Map (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FDCH	EBX	Yes
7FD8H	EDX	Yes
7FD4H	ECX	Yes
7FD0H	EAX	Yes
7FCCH	DR6	No
7FC8H	DR7	No
7FC4H	TR[1]	No
7FC0H	Reserved	No
7FBCH	GS[1]	No
7FB8H	FS[1]	No
7FB4H	DS[1]	No
7FB0H	SS[1]	No
7FACH	CS[1]	No
7FA8H	ES[1]	No
7FA4H	I/O State Field, see Section 26.7	No
7FA0H	I/O Memory Address Field, see Section 26.7	No
7F9FH-7F03H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7E00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

The following registers are saved (but not readable) and restored upon exiting SMM:

- Control register CR4. (This register is cleared to all 0s while in SMM).
- The hidden segment descriptor information stored in segment registers CS, DS, ES, FS, GS, and SS.

If an SMI request is issued for the purpose of powering down the processor, the values of all reserved locations in the SMM state save must be saved to nonvolatile memory.

The following state is not automatically saved and restored following an SMI and the RSM instruction, respectively:

- Debug registers DR0 through DR3.
- The x87 FPU registers.
- The MTRRs.
- Control register CR2.
- The model-specific registers (for the P6 family and Pentium processors) or test registers TR3 through TR7 (for the Pentium and Intel486 processors).
- The state of the trap controller.
- The machine-check architecture registers.
- The APIC internal interrupt state (ISR, IRR, etc.).
- The microcode update state.

If an SMI is used to power down the processor, a power-on reset will be required before returning to SMM, which will reset much of this state back to its default values. So an SMI handler that is going to trigger power down should first read these registers listed above directly, and save them (along with the rest of RAM) to nonvolatile storage. After the power-on reset, the continuation of the SMI handler should restore these values, along with the rest of the system's state. Anytime the SMI handler changes these registers in the processor, it must also save and restore them.

NOTES

A small subset of the MSRs (such as, the time-stamp counter and performance-monitoring counters) are not arbitrarily writable and therefore cannot be saved and restored. SMM-based power-down and restoration should only be performed with operating systems that do not use or rely on the values of these registers.

Operating system developers should be aware of this fact and insure that their operating-system assisted power-down and restoration software is immune to unexpected changes in these register values.

26.4.1.1 SMRAM State Save Map and Intel 64 Architecture

When the processor initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area on an Intel 64 processor at [SMBASE + 8000H + 7FFFH] and extends to [SMBASE + 8000H + 7C00H].

Support for Intel 64 architecture is reported by CPUID.80000001:EDX[29] = 1. The layout of the SMRAM state save map is shown in Table 26-3.

Additionally, the SMRAM state save map shown in Table 26-3 also applies to processors with the following CPUID signatures listed in Table 26-2, irrespective of the value in CPUID.80000001:EDX[29].

Table 26-2. Processor Signatures and 64-bit SMRAM State Save Map Format

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_17H	Intel Xeon Processor 5200, 5400 series, Intel Core 2 Quad processor Q9xxx, Intel Core 2 Duo processors E8000, T9000,
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad, Intel Core 2 Extreme, Intel Core 2 Duo processors, Intel Pentium dual-core processors
06_1CH	Intel Atom processors

Table 26-3. SMRAM State Save Map for Intel 64 Architecture

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FF8H	CR0	No
7FF0H	CR3	No
7FE8H	RFLAGS	Yes
7FE0H	IA32_EFER	Yes
7FD8H	RIP	Yes
7FD0H	DR6	No
7FC8H	DR7	No
7FC4H	TR SEL[1]	No
7FC0H	LDTR SEL[1]	No
7FBCH	GS SEL[1]	No
7FB8H	FS SEL[1]	No
7FB4H	DS SEL[1]	No
7FB0H	SS SEL[1]	No
7FACH	CS SEL[1]	No
7FA8H	ES SEL[1]	No
7FA4H	IO_MISC	No
7F9CH	IO_MEM_ADDR	No

Table 26-3. SMRAM State Save Map for Intel 64 Architecture (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7F94H	RDI	Yes
7F8CH	RSI	Yes
7F84H	RBP	Yes
7F7CH	RSP	Yes
7F74H	RBX	Yes
7F6CH	RDX	Yes
7F64H	RCX	Yes
7F5CH	RAX	Yes
7F54H	R8	Yes
7F4CH	R9	Yes
7F44H	R10	Yes
7F3CH	R11	Yes
7F34H	R12	Yes
7F2CH	R13	Yes
7F24H	R14	Yes
7F1CH	R15	Yes
7F1BH-7F04H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7EE4H	Reserved	No
7EE0H	Setting of “enable EPT” VM-execution control	No
7ED8H	Value of EPTP VM-execution control field	No
7ED7H - 7EA8H	Reserved	No
7EA4H	LDT Info	No
7EA0H	LDT Limit	No
7E9CH	LDT Base (lower 32 bits)	No
7E98H	IDT Limit	No

Table 26-3. SMRAM State Save Map for Intel 64 Architecture (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7E94H	IDT Base (lower 32 bits)	No
7E90H	GDT Limit	No
7E8CH	GDT Base (lower 32 bits)	No
7E8BH - 7E44H	Reserved	No
7E40H	CR4	No
7E3FH - 7DF0H	Reserved	No
7DE8H	IO_EIP	Yes
7DE7H - 7DDCH	Reserved	No
7DD8H	IDT Base (Upper 32 bits)	No
7DD4H	LDT Base (Upper 32 bits)	No
7DD0H	GDT Base (Upper 32 bits)	No
7DCFH - 7C00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

26.4.2 SMRAM Caching

An IA-32 processor does not automatically write back and invalidate its caches before entering SMM or before exiting SMM. Because of this behavior, care must be taken in the placement of the SMRAM in system memory and in the caching of the SMRAM to prevent cache incoherence when switching back and forth between SMM and protected mode operation. Either of the following three methods of locating the SMRAM in system memory will guarantee cache coherency:

- Place the SRAM in a dedicated section of system memory that the operating system and applications are prevented from accessing. Here, the SRAM can be designated as cacheable (WB, WT, or WC) for optimum processor performance, without risking cache incoherence when entering or exiting SMM.
- Place the SRAM in a section of memory that overlaps an area used by the operating system (such as the video memory), but designate the SMRAM as uncacheable (UC). This method prevents cache access when in SMM to maintain cache coherency, but the use of uncacheable memory reduces the performance of SMM code.
- Place the SRAM in a section of system memory that overlaps an area used by the operating system and/or application code, but explicitly flush (write back and invalidate) the caches upon entering and exiting SMM mode. This method

maintains cache coherency, but the incurs the overhead of two complete cache flushes.

For Pentium 4, Intel Xeon, and P6 family processors, a combination of the first two methods of locating the SMRAM is recommended. Here the SMRAM is split between an overlapping and a dedicated region of memory. Upon entering SMM, the SMRAM space that is accessed overlaps video memory (typically located in low memory). This SMRAM section is designated as UC memory. The initial SMM code then jumps to a second SMRAM section that is located in a dedicated region of system memory (typically in high memory). This SMRAM section can be cached for optimum processor performance.

For systems that explicitly flush the caches upon entering SMM (the third method described above), the cache flush can be accomplished by asserting the FLUSH# pin at the same time as the request to enter SMM (generally initiated by asserting the SMI# pin). The priorities of the FLUSH# and SMI# pins are such that the FLUSH# is serviced first. To guarantee this behavior, the processor requires that the following constraints on the interaction of FLUSH# and SMI# be met. In a system where the FLUSH# and SMI# pins are synchronous and the set up and hold times are met, then the FLUSH# and SMI# pins may be asserted in the same clock. In asynchronous systems, the FLUSH# pin must be asserted at least one clock before the SMI# pin to guarantee that the FLUSH# pin is serviced first.

Upon leaving SMM (for systems that explicitly flush the caches), the WBINVD instruction should be executed prior to leaving SMM to flush the caches.

NOTES

In systems based on the Pentium processor that use the FLUSH# pin to write back and invalidate cache contents before entering SMM, the processor will prefetch at least one cache line in between when the Flush Acknowledge cycle is run and the subsequent recognition of SMI# and the assertion of SMIACT#.

It is the obligation of the system to ensure that these lines are not cached by returning KEN# inactive to the Pentium processor.

26.5 SMI HANDLER EXECUTION ENVIRONMENT

After saving the current context of the processor, the processor initializes its core registers to the values shown in Table 26-4. Upon entering SMM, the PE and PG flags in control register CR0 are cleared, which places the processor is in an environment similar to real-address mode. The differences between the SMM execution environment and the real-address mode execution environment are as follows:

- The addressable SMRAM address space ranges from 0 to FFFFFFFFH (4 GBytes). (The physical address extension (enabled with the PAE flag in control register CR4) is not supported in SMM.)
- The normal 64-KByte segment limit for real-address mode is increased to 4 GBytes.

- The default operand and address sizes are set to 16 bits, which restricts the addressable SMRAM address space to the 1-MByte real-address mode limit for native real-address-mode code. However, operand-size and address-size override prefixes can be used to access the address space beyond the 1-MByte.

Table 26-4. Processor Register Initialization in SMM

Register	Contents
General-purpose registers	Undefined
EFLAGS	00000002H
EIP	00008000H
CS selector	SMM Base shifted right 4 bits (default 3000H)
CS base	SMM Base (default 30000H)
DS, ES, FS, GS, SS Selectors	0000H
DS, ES, FS, GS, SS Bases	000000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFFH
CR0	PE, EM, TS, and PG flags set to 0; others unmodified
CR4	Cleared to zero
DR6	Undefined
DR7	00000400H

- Near jumps and calls can be made to anywhere in the 4-GByte address space if a 32-bit operand-size override prefix is used. Due to the real-address-mode style of base-address formation, a far call or jump cannot transfer control to a segment with a base address of more than 20 bits (1 MByte). However, since the segment limit in SMM is 4 GBytes, offsets into a segment that go beyond the 1-MByte limit are allowed when using 32-bit operand-size override prefixes. Any program control transfer that does not have a 32-bit operand-size override prefix truncates the EIP value to the 16 low-order bits.
- Data and the stack can be located anywhere in the 4-GByte address space, but can be accessed only with a 32-bit address-size override if they are located above 1 MByte. As with the code segment, the base address for a data or stack segment cannot be more than 20 bits.

The value in segment register CS is automatically set to the default of 30000H for the SMBASE shifted 4 bits to the right; that is, 3000H. The EIP register is set to 8000H. When the EIP value is added to shifted CS value (the SMBASE), the resulting linear address points to the first instruction of the SMI handler.

The other segment registers (DS, SS, ES, FS, and GS) are cleared to 0 and their segment limits are set to 4 GBytes. In this state, the SMRAM address space may be treated as a single flat 4-GByte linear address space. If a segment register is loaded with a 16-bit value, that value is then shifted left by 4 bits and loaded into the segment base (hidden part of the segment register). The limits and attributes are not modified.

Maskable hardware interrupts, exceptions, NMI interrupts, SMI interrupts, A20M interrupts, single-step traps, breakpoint traps, and INIT operations are inhibited when the processor enters SMM. Maskable hardware interrupts, exceptions, single-step traps, and breakpoint traps can be enabled in SMM if the SMM execution environment provides and initializes an interrupt table and the necessary interrupt and exception handlers (see Section 26.6).

26.6 EXCEPTIONS AND INTERRUPTS WITHIN SMM

When the processor enters SMM, all hardware interrupts are disabled in the following manner:

- The IF flag in the EFLAGS register is cleared, which inhibits maskable hardware interrupts from being generated.
- The TF flag in the EFLAGS register is cleared, which disables single-step traps.
- Debug register DR7 is cleared, which disables breakpoint traps. (This action prevents a debugger from accidentally breaking into an SMM handler if a debug breakpoint is set in normal address space that overlays code or data in SMRAM.)
- NMI, SMI, and A20M interrupts are blocked by internal SMM logic. (See Section 26.8 for more information about how NMIs are handled in SMM.)

Software-invoked interrupts and exceptions can still occur, and maskable hardware interrupts can be enabled by setting the IF flag. Intel recommends that SMM code be written in so that it does not invoke software interrupts (with the INT *n*, INTO, INT 3, or BOUND instructions) or generate exceptions.

If the SMM handler requires interrupt and exception handling, an SMM interrupt table and the necessary exception and interrupt handlers must be created and initialized from within SMM. Until the interrupt table is correctly initialized (using the LIDT instruction), exceptions and software interrupts will result in unpredictable processor behavior.

The following restrictions apply when designing SMM interrupt and exception-handling facilities:

- The interrupt table should be located at linear address 0 and must contain real-address mode style interrupt vectors (4 bytes containing CS and IP).
- Due to the real-address mode style of base address formation, an interrupt or exception cannot transfer control to a segment with a base address of more that 20 bits.
- An interrupt or exception cannot transfer control to a segment offset of more than 16 bits (64 KBytes).
- When an exception or interrupt occurs, only the 16 least-significant bits of the return address (EIP) are pushed onto the stack. If the offset of the interrupted procedure is greater than 64 KBytes, it is not possible for the interrupt/exception handler to return control to that procedure. (One solution to this problem is for a handler to adjust the return address on the stack.)

- The SMBASE relocation feature affects the way the processor will return from an interrupt or exception generated while the SMI handler is executing. For example, if the SMBASE is relocated to above 1 MByte, but the exception handlers are below 1 MByte, a normal return to the SMI handler is not possible. One solution is to provide the exception handler with a mechanism for calculating a return address above 1 MByte from the 16-bit return address on the stack, then use a 32-bit far call to return to the interrupted procedure.
- If an SMI handler needs access to the debug trap facilities, it must insure that an SMM accessible debug handler is available and save the current contents of debug registers DR0 through DR3 (for later restoration). Debug registers DR0 through DR3 and DR7 must then be initialized with the appropriate values.
- If an SMI handler needs access to the single-step mechanism, it must insure that an SMM accessible single-step handler is available, and then set the TF flag in the EFLAGS register.
- If the SMI design requires the processor to respond to maskable hardware interrupts or software-generated interrupts while in SMM, it must ensure that SMM accessible interrupt handlers are available and then set the IF flag in the EFLAGS register (using the STI instruction). Software interrupts are not blocked upon entry to SMM, so they do not need to be enabled.

26.7 MANAGING SYNCHRONOUS AND ASYNCHRONOUS SYSTEM MANAGEMENT INTERRUPTS

When coding for a multiprocessor system or a system with Intel HT Technology, it was not always possible for an SMI handler to distinguish between a synchronous SMI (triggered during an I/O instruction) and an asynchronous SMI. To facilitate the discrimination of these two events, incremental state information has been added to the SMM state save map.

Processors that have an SMM revision ID of 30004H or higher have the incremental state information described below.

26.7.1 I/O State Implementation

Within the extended SMM state save map, a bit (IO_SMI) is provided that is set only when an SMI is either taken immediately after a *successful* I/O instruction or is taken after a *successful* iteration of a REP I/O instruction (note that the *successful* notion pertains to the processor point of view; not necessarily to the corresponding platform function). When set, the IO_SMI bit provides a strong indication that the corresponding SMI was synchronous. In this case, the SMM State Save Map also supplies the port address of the I/O operation. The IO_SMI bit and the I/O Port Address may be used in conjunction with the information logged by the platform to confirm that the SMI was indeed synchronous.

Note that the IO_SMI bit by itself is a strong indication, not a guarantee, that the SMI is synchronous. This is because an asynchronous SMI might coincidentally be taken after an I/O instruction. In such a case, the IO_SMI bit would still be set in the SMM state save map.

Information characterizing the I/O instruction is saved in two locations in the SMM State Save Map (Table 26-5). Note that the IO_SMI bit also serves as a valid bit for the rest of the I/O information fields. The contents of these I/O information fields are not defined when the IO_SMI bit is not set.

Table 26-5. I/O Instruction Information in the SMM State Save Map

State (SMM Rev. ID: 30004H or higher)	**Format**				
	31 16	15 8	7 4	3 1	0
I/O State Field SMRAM offset 7FA4	I/O Port	Reserved	I/O Type	I/O Length	IO_SMI
	31				0
I/O Memory Address Field SMRAM offset 7FA0	I/O Memory Address				

When IO_SMI is set, the other fields may be interpreted as follows:

- I/O length:
  - 001 – Byte
  - 010 – Word
  - 100 – Dword
- I/O instruction type (Table 26-6)

Table 26-6. I/O Instruction Type Encodings

Instruction	**Encoding**
IN Immediate	1001
IN DX	0001
OUT Immediate	1000
OUT DX	0000
INS	0011
OUTS	0010
REP INS	0111
REP OUTS	0110

26.8 NMI HANDLING WHILE IN SMM

NMI interrupts are blocked upon entry to the SMI handler. If an NMI request occurs during the SMI handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMI handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence. This assumes that NMIs were not blocked before the SMI occurred. If NMIs were blocked before the SMI occurred, they are blocked after execution of RSM.

Although NMI requests are blocked when the processor enters SMM, they may be enabled through software by executing an IRET instruction. If the SMM handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET instruction. Once an IRET instruction is executed, NMI interrupt requests are serviced in the same “real mode” manner in which they are handled outside of SMM.

A special case can occur if an SMI handler nests inside an NMI handler and then another NMI occurs. During NMI interrupt handling, NMI interrupts are disabled, so normally NMI interrupts are serviced and completed with an IRET instruction one at a time. When the processor enters SMM while executing an NMI handler, the processor saves the SMRAM state save map but does not save the attribute to keep NMI interrupts disabled. Potentially, an NMI could be latched (while in SMM or upon exit) and serviced upon exit of SMM even though the previous NMI handler has still not completed. One or more NMIs could thus be nested inside the first NMI handler. The NMI interrupt handler should take this possibility into consideration.

Also, for the Pentium processor, exceptions that invoke a trap or fault handler will enable NMI interrupts from inside of SMM. This behavior is implementation specific for the Pentium processor and is not part of the IA-32 architecture.

26.9 SMM REVISION IDENTIFIER

The SMM revision identifier field is used to indicate the version of SMM and the SMM extensions that are supported by the processor (see Figure 26-2). The SMM revision identifier is written during SMM entry and can be examined in SMRAM space at offset 7EFCH. The lower word of the SMM revision identifier refers to the version of the base SMM architecture.

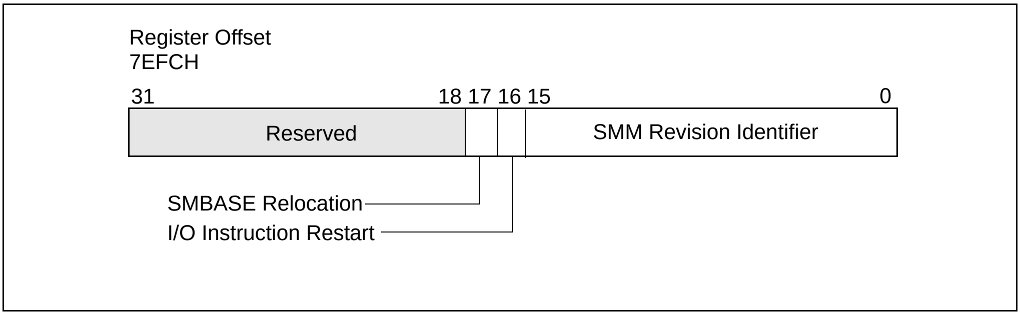

Figure 26-2. SMM Revision Identifier

The upper word of the SMM revision identifier refers to the extensions available. If the I/O instruction restart flag (bit 16) is set, the processor supports the I/O instruction restart (see Section 26.12); if the SMBASE relocation flag (bit 17) is set, SMRAM base address relocation is supported (see Section 26.11).

26.10 AUTO HALT RESTART

If the processor is in a HALT state (due to the prior execution of a HLT instruction) when it receives an SMI, the processor records the fact in the auto HALT restart flag in the saved processor state (see Figure 26-3). (This flag is located at offset 7F02H and bit 0 in the state save area of the SMRAM.)

If the processor sets the auto HALT restart flag upon entering SMM (indicating that the SMI occurred when the processor was in the HALT state), the SMI handler has two options:

- It can leave the auto HALT restart flag set, which instructs the RSM instruction to return program control to the HLT instruction. This option in effect causes the processor to re-enter the HALT state after handling the SMI. (This is the default operation.)
- It can clear the auto HALT restart flag, with instructs the RSM instruction to return program control to the instruction following the HLT instruction.

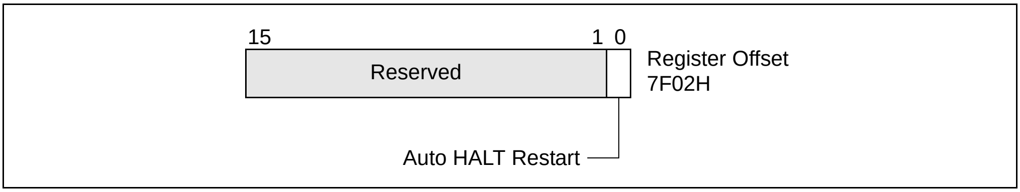

Figure 26-3. Auto HALT Restart Field

These options are summarized in Table 26-7. Note that if the processor was not in a HALT state when the SMI was received (the auto HALT restart flag is cleared), setting the flag to 1 will cause unpredictable behavior when the RSM instruction is executed.

Table 26-7. Auto HALT Restart Flag Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
0	0	Returns to next instruction in interrupted program or task.
0	1	Unpredictable.
1	0	Returns to next instruction after HLT instruction.
1	1	Returns to HALT state.

If the HLT instruction is restarted, the processor will generate a memory access to fetch the HLT instruction (if it is not in the internal cache), and execute a HLT bus transaction. This behavior results in multiple HLT bus transactions for the same HLT instruction.

26.10.1 Executing the HLT Instruction in SMM

The HLT instruction should not be executed during SMM, unless interrupts have been enabled by setting the IF flag in the EFLAGS register. If the processor is halted in SMM, the only event that can remove the processor from this state is a maskable hardware interrupt or a hardware reset.

26.11 SMBASE RELOCATION

The default base address for the SMRAM is 30000H. This value is contained in an internal processor register called the SMBASE register. The operating system or executive can relocate the SMRAM by setting the SMBASE field in the saved state map (at offset 7EF8H) to a new value (see Figure 26-4). The RSM instruction reloads the internal SMBASE register with the value in the SMBASE field each time it exits SMM. All subsequent SMI requests will use the new SMBASE value to find the starting address for the SMI handler (at SMBASE + 8000H) and the SMRAM state save area (from SMBASE + FE00H to SMBASE + FFFFH). (The processor resets the value in its internal SMBASE register to 30000H on a RESET, but does not change it on an INIT.)

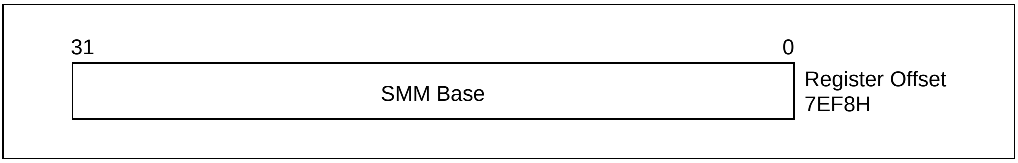

Figure 26-4. SMBASE Relocation Field

In multiple-processor systems, initialization software must adjust the SMBASE value for each processor so that the SMRAM state save areas for each processor do not overlap. (For Pentium and Intel486 processors, the SMBASE values must be aligned on a 32-KByte boundary or the processor will enter shutdown state during the execution of a RSM instruction.)

If the SMBASE relocation flag in the SMM revision identifier field is set, it indicates the ability to relocate the SMBASE (see Section 26.9).

26.11.1 Relocating SMRAM to an Address Above 1 MByte

In SMM, the segment base registers can only be updated by changing the value in the segment registers. The segment registers contain only 16 bits, which allows only 20 bits to be used for a segment base address (the segment register is shifted left 4 bits to determine the segment base address). If SMRAM is relocated to an address above 1 MByte, software operating in real-address mode can no longer initialize the segment registers to point to the SMRAM base address (SMBASE).

The SMRAM can still be accessed by using 32-bit address-size override prefixes to generate an offset to the correct address. For example, if the SMBASE has been relocated to FFFFFFH (immediately below the 16-MByte boundary) and the DS, ES, FS, and GS registers are still initialized to 0H, data in SMRAM can be accessed by using 32-bit displacement registers, as in the following example:

```
mov     esi,00FFxxxxH; 64K segment immediately below 16M
mov     ax,ds:[esi]
```

A stack located above the 1-MByte boundary can be accessed in the same manner.

26.12 I/O INSTRUCTION RESTART

If the I/O instruction restart flag in the SMM revision identifier field is set (see Section 26.9), the I/O instruction restart mechanism is present on the processor. This mechanism allows an interrupted I/O instruction to be re-executed upon returning from SMM mode. For example, if an I/O instruction is used to access a powered-down I/O device, a chip set supporting this device can intercept the access and respond by asserting SMI#. This action invokes the SMI handler to power-up the device. Upon returning from the SMI handler, the I/O instruction restart mechanism can be used to re-execute the I/O instruction that caused the SMI.

The I/O instruction restart field (at offset 7F00H in the SMM state-save area, see Figure 26-5) controls I/O instruction restart. When an RSM instruction is executed, if this field contains the value FFH, then the EIP register is modified to point to the I/O instruction that received the SMI request. The processor will then automatically re-execute the I/O instruction that the SMI trapped. (The processor saves the necessary machine state to insure that re-execution of the instruction is handled coherently.)

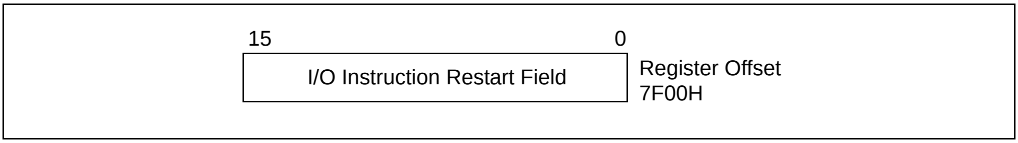

Figure 26-5. I/O Instruction Restart Field

If the I/O instruction restart field contains the value 00H when the RSM instruction is executed, then the processor begins program execution with the instruction following the I/O instruction. (When a repeat prefix is being used, the next instruction may be the next I/O instruction in the repeat loop.) Not re-executing the interrupted I/O instruction is the default behavior; the processor automatically initializes the I/O instruction restart field to 00H upon entering SMM. Table 26-8 summarizes the states of the I/O instruction restart field.

Table 26-8. I/O Instruction Restart Field Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
00H	00H	Does not re-execute trapped I/O instruction.
00H	FFH	Re-executes trapped I/O instruction.

Note that the I/O instruction restart mechanism does not indicate the cause of the SMI. It is the responsibility of the SMI handler to examine the state of the processor to determine the cause of the SMI and to determine if an I/O instruction was interrupted and should be restarted upon exiting SMM. If an SMI interrupt is signaled on a non-I/O instruction boundary, setting the I/O instruction restart field to FFH prior to executing the RSM instruction will likely result in a program error.

26.12.1 Back-to-Back SMI Interrupts When I/O Instruction Restart Is Being Used

If an SMI interrupt is signaled while the processor is servicing an SMI interrupt that occurred on an I/O instruction boundary, the processor will service the new SMI request before restarting the originally interrupted I/O instruction. If the I/O instruction restart field is set to FFH prior to returning from the second SMI handler, the EIP will point to an address different from the originally interrupted I/O instruction, which will likely lead to a program error. To avoid this situation, the SMI handler must be able to recognize the occurrence of back-to-back SMI interrupts when I/O instruction restart is being used and insure that the handler sets the I/O instruction restart field to 00H prior to returning from the second invocation of the SMI handler.

26.13 SMM MULTIPLE-PROCESSOR CONSIDERATIONS

The following should be noted when designing multiple-processor systems:

- Any processor in a multiprocessor system can respond to an SMM.
- Each processor needs its own SMRAM space. This space can be in system memory or in a separate RAM.
- The SMRAMs for different processors can be overlapped in the same memory space. The only stipulation is that each processor needs its own state save area and its own dynamic data storage area. (Also, for the Pentium and Intel486 processors, the SMBASE address must be located on a 32-KByte boundary.) Code and static data can be shared among processors. Overlapping SMRAM spaces can be done more efficiently with the P6 family processors because they do not require that the SMBASE address be on a 32-KByte boundary.
- The SMI handler will need to initialize the SMBASE for each processor.
- Processors can respond to local SMIs through their SMI# pins or to SMIs received through the APIC interface. The APIC interface can distribute SMIs to different processors.
- Two or more processors can be executing in SMM at the same time.
- When operating Pentium processors in dual processing (DP) mode, the SMIACT# pin is driven only by the MRM processor and should be sampled with ADS#. For additional details, see Chapter 14 of the *Pentium Processor Family User's Manual, Volume 1*.

SMM is not re-entrant, because the SMRAM State Save Map is fixed relative to the SMBASE. If there is a need to support two or more processors in SMM mode at the same time then each processor should have dedicated SMRAM spaces. This can be done by using the SMBASE Relocation feature (see Section 26.11).

26.14 DEFAULT TREATMENT OF SMIS AND SMM WITH VMX OPERATION AND SMX OPERATION

Under the default treatment, the interactions of SMIs and SMM with VMX operation are few. This section details those interactions. It also explains how this treatment affects SMX operation.

26.14.1 Default Treatment of SMI Delivery

Ordinary SMI delivery saves processor state into SMRAM and then loads state based on architectural definitions. Under the default treatment, processors that support VMX operation perform SMI delivery as follows:

```
Enter SMM;
save the following internal to the processor:
    CR4.VMXE
    an indication of whether the logical processor was in VMX operation (root or non-root)
IF the logical processor is in VMX operation
    THEN
        save current VMCS pointer internal to the processor;
        leave VMX operation;
        save VMX-critical state defined below;
FI;
IF the logical processor supports SMX operation
    THEN
        save internal to the logical processor an indication of whether the Intel® TXT private space
is locked;
        IF the TXT private space is unlocked
            THEN lock the TXT private space;
        FI;
FI;
CR4.VMXE ← 0;
perform ordinary SMI delivery:
    save processor state in SMRAM;
    set processor state to standard SMM values;[1]
    invalidate information in the TLBs and paging-structure caches for VPID 0000H (see Chapter
24);
```

The pseudocode above makes reference to the saving of **VMX-critical state**. This state consists of the following: (1) SS.DPL (the current privilege level); (2) RFLAGS.VM[2]; (3) the state of blocking by STI and by MOV SS (see Table 20-3 in Section 20.4.2); (4) the state of virtual-NMI blocking (only if the processor is in VMX non-root operation and the “virtual NMIs” VM-execution control is 1); and (5) an indication of whether an MTF VM exit is pending (see Section 21.7.2). These data may be saved internal to the processor or in the VMCS region of the current VMCS.

[1] This causes the logical processor to block INIT signals, NMIs, and SMIs.

[2] Section 26.14 and Section 26.15 use the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of these registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to the lower 32 bits of the register.

Note that processors that do not support SMI recognition while there is blocking by STI or by MOV SS need not save the state of such blocking.

If the logical processor supports the 1-setting of the “enable EPT” VM-execution control and the logical processor was in VMX non-root operation at the time of an SMI, it saves the value of that control into bit 0 of the 32-bit field at offset SMBASE + 8000H + 7EE0H (SMBASE + FEE0H; see Table 26-3).[1] If the logical processor was not in VMX non-root operation at the time of the SMI, it saves 0 into that bit. If the logical processor saves 1 into that bit (it was in VMX non-root operation and the “enable EPT” VM-execution control was 1), it saves the value of the EPT pointer (EPTP) into the 64-bit field at offset SMBASE + 8000H + 7ED8H (SMBASE + FED8H).

Because SMI delivery causes a logical processor to leave VMX operation, all the controls associated with VMX non-root operation are disabled in SMM and thus cannot cause VM exits while the logical processor in SMM.

26.14.2 Default Treatment of RSM

Ordinary execution of RSM restores processor state from SMRAM. Under the default treatment, processors that support VMX operation perform RSM as follows:

```
IF VMXE = 1 in CR4 image in SMRAM
   THEN fail and enter shutdown state;
   ELSE
       restore state normally from SMRAM;
       invalidate information in the TLBs and paging-structure caches for all VPIDs (see Chapter 24);
       IF the logical processor supports SMX operation andthe Intel® TXT private space was unlocked at the time of the last SMI (as saved)
           THEN unlock the TXT private space;
       FI;
       CR4.VMXE ← value stored internally;
       IF internal storage indicates that the logical processor
       had been in VMX operation (root or non-root)
           THEN
               enter VMX operation (root or non-root);
               restore VMX-critical state as defined in Section 26.14.1;
               set to their fixed values any bits in CR0 and CR4 whose values must be fixed in VMX operation (see Section 19.8);
               IF RFLAGS.VM = 0
                   THEN
                       CS.RPL ← SS.DPL;
```

[1] Note that “enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, SMI functions as the “enable EPT” VM-execution control were 0. See Section 20.6.2.

```
                        SS.RPL ← SS.DPL;
                FI;
                restore current VMCS pointer;
        FI;
        Leave SMM;
        IF logical processor will be in VMX operation or in SMX operation after RSM
            THEN block A20M and leave A20M mode;
        FI;
FI;
```

RSM unblocks SMIs. It restores the state of blocking by NMI (see Table 20-3 in Section 20.4.2) as follows:

- If the RSM is not to VMX non-root operation or if the “virtual NMIs” VM-execution control will be 0, the state of NMI blocking is restored normally.
- If the RSM is to VMX non-root operation and the “virtual NMIs” VM-execution control will be 1, NMIs are not blocked after RSM. The state of virtual-NMI blocking is restored as part of VMX-critical state.

INIT signals are blocked after RSM if and only if the logical processor will be in VMX root operation.

If RSM returns a logical processor to VMX non-root operation, it re-establishes the controls associated with the current VMCS. If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs immediately after RSM if the enabling conditions apply. The same is true for the “NMI-window exiting” VM-execution control. Such VM exits occur with their normal priority. See Section 21.3.

If an MTF VM exit was pending at the time of the previous SMI, an MTF VM exit is pending on the instruction boundary following execution of RSM. The following items detail the treatment of MTF VM exits that may be pending following RSM:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these MTF VM exits. These MTF VM exits take priority over debug-trap exceptions and lower priority events.
- These MTF VM exits wake the logical processor if RSM caused the logical processor to enter the HLT state (see Section 26.10). They do not occur if the logical processor just entered the shutdown state.

26.14.3 Protection of CR4.VMXE in SMM

Under the default treatment, CR4.VMXE is treated as a reserved bit while a logical processor is in SMM. Any attempt by software running in SMM to set this bit causes a general-protection exception. In addition, software cannot use VMX instructions or enter VMX operation while in SMM.

26.15 DUAL-MONITOR TREATMENT OF SMIs AND SMM

Dual-monitor treatment is activated through the cooperation of the **executive monitor** (the VMM that operates outside of SMM to provide basic virtualization) and the **SMM monitor** (the VMM that operates inside SMM—while in VMX operation—to support system-management functions). Control is transferred to the SMM monitor through VM exits; VM entries are used to return from SMM.

The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix G.1) to determine whether it is supported.

26.15.1 Dual-Monitor Treatment Overview

The dual-monitor treatment uses an executive monitor and an SMM monitor. Transitions from the executive monitor or its guests to the SMM monitor are called **SMM VM exits** and are discussed in Section 26.15.2. SMM VM exits are caused by SMIs as well as executions of VMCALL in VMX root operation. The latter allow the executive monitor to call the SMM monitor for service.

The SMM monitor runs in VMX root operation and uses VMX instructions to establish a VMCS and perform VM entries to its own guests. This is done all inside SMM (see Section 26.15.3). The SMM monitor returns from SMM, not by using the RSM instruction, but by using a VM entry that returns from SMM. Such VM entries are described in Section 26.15.4.

Initially, there is no SMM monitor and the default treatment (Section 26.14) is used. The dual-monitor treatment is not used until it is enabled and activated. The steps to do this are described in Section 26.15.5 and Section 26.15.6.

It is not possible to leave VMX operation under the dual-monitor treatment; VMXOFF will fail if executed. The dual-monitor treatment must be deactivated first. The SMM monitor deactivates dual-monitor treatment using a VM entry that returns from SMM with the “deactivate dual-monitor treatment” VM-entry control set to 1 (see Section 26.15.7).

The executive monitor configures any VMCS that it uses for VM exits to the executive monitor. SMM VM exits, which transfer control to the SMM monitor, use a different VMCS. Under the dual-monitor treatment, each logical processor uses a separate VMCS called the **SMM-transfer VMCS**. When the dual-monitor treatment is active, the logical processor maintains another VMCS pointer called the **SMM-transfer VMCS pointer**. The SMM-transfer VMCS pointer is established when the dual-monitor treatment is activated.

26.15.2 SMM VM Exits

An SMM VM exit is a VM exit that begins outside SMM and that ends in SMM.

Unlike other VM exits, SMM VM exits can begin in VMX root operation. SMM VM exits result from the arrival of an SMI outside SMM or from execution of VMCALL in VMX root operation outside SMM. Execution of VMCALL in VMX root operation causes an SMM VM exit only if the valid bit is set in the IA32_SMM_MONITOR_CTL MSR (see Section 26.15.5).

Execution of VMCALL in VMX root operation causes an SMM VM exit even under the default treatment. This SMM VM exit activates the dual-monitor treatment (see Section 26.15.6).

Differences between SMM VM exits and other VM exits are detailed in Sections 26.15.2.1 through 26.15.2.5. Differences between SMM VM exits that activate the dual-monitor treatment and other SMM VM exits are described in Section 26.15.6.

26.15.2.1 Architectural State Before a VM Exit

System-management interrupts (SMIs) that cause SMM VM exits always do so directly. They do not save state to SMRAM as they do under the default treatment.

26.15.2.2 Updating the Current-VMCS and Executive-VMCS Pointers

SMM VM exits begin by performing the following steps:

1. The executive-VMCS pointer field in the SMM-transfer VMCS is loaded as follows:
   - If the SMM VM exit commenced in VMX non-root operation, it receives the current-VMCS pointer.
   - If the SMM VM exit commenced in VMX root operation, it receives the VMXON pointer.
2. The current-VMCS pointer is loaded with the value of the SMM-transfer VMCS pointer.

The last step ensures that the current VMCS is the SMM-transfer VMCS. VM-exit information is recorded in that VMCS, and VM-entry control fields in that VMCS are updated. State is saved into the guest-state area of that VMCS. The VM-exit controls and host-state area of that VMCS determine how the VM exit operates.

26.15.2.3 Recording VM-Exit Information

SMM VM exits differ from other VM exit with regard to the way they record VM-exit information. The differences follow.

- **Exit reason**.
  - Bits 15:0 of this field contain the basic exit reason. The field is loaded with the reason for the SMM VM exit: I/O SMI (an SMI arrived immediately after retirement of an I/O instruction), other SMI, or VMCALL. See Appendix I, “VMX Basic Exit Reasons”.

- — SMM VM exits are the only VM exits that may occur in VMX root operation. Because the SMM monitor may need to know whether it was invoked from VMX root or VMX non-root operation, this information is stored in bit 29 of the exit-reason field (see Table 20-13 in Section 20.9.1). The bit is set by SMM VM exits from VMX root operation.
- — If the SMM VM exit occurred in VMX non-root operation and an MTF VM exit was pending, bit 28 of the exit-reason field is set; otherwise, it is cleared.
- — Bits 27:16 and bits 31:30 are cleared.

- **Exit qualification.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, the exit qualification contains information about the I/O instruction that retired immediately before the SMI.It has the format given in Table 26-9.

Table 26-9. Exit Qualification for SMIs That Arrive Immediately After the Retirement of an I/O Instruction

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used.
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in the I/O instruction)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

- **Guest linear address.** This field is used for VM exits due to SMIs that arrive immediately after the retirement of an INS or OUTS instruction for which the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) is usable. The field receives the value of the linear address generated by ES:(E)DI (for INS) or segment:(E)SI (for OUTS; the default segment is DS but can be overridden by a segment override prefix) at the time the instruction started. If the relevant segment is not usable, the value is undefined. On processors that support Intel 64 architecture, bits 63:32 are clear if the logical processor was not in 64-bit mode before the VM exit.

- **I/O RCX, I/O RSI, I/O RDI, and I/O RIP.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, these fields receive the values that were in RCX, RSI, RDI, and RIP, respectively, before the I/O instruction executed. Thus, the value saved for I/O RIP addresses the I/O instruction.

26.15.2.4 Saving Guest State

SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area.

The value of the VMX-preemption timer is saved into the corresponding field in the guest-state area if the "save VMX-preemption timer value" VM-exit control is 1. That field becomes undefined if, in addition, either the SMM VM exit is from VMX root operation or the SMM VM exit is from VMX non-root operation and the "activate VMX-preemption timer" VM-execution control is 0.

26.15.2.5 Updating Non-Register State

SMM VM exits affect the non-register state of a logical processor as follows:

- SMM VM exits cause non-maskable interrupts (NMIs) to be blocked; they may be unblocked through execution of IRET or through a VM entry (depending on the value loaded for the interruptibility state and the setting of the "virtual NMIs" VM-execution control).
- SMM VM exits cause SMIs to be blocked; they may be unblocked by a VM entry that returns from SMM (see Section 26.15.4).

SMM VM exits invalidate information in the TLBs and paging-structure caches for VPID 0000H (see Chapter 24).

26.15.3 Operation of an SMM Monitor

Once invoked, an SMM monitor is in VMX root operation and can use VMX instructions to configure VMCSs and to cause VM entries to virtual machines supported by those structures. As noted in Section 26.15.1, the VMXOFF instruction cannot be used under the dual-monitor treatment and thus cannot be used by an SMM monitor.

The RSM instruction also cannot be used under the dual-monitor treatment. As noted in Section 21.1.3, it causes a VM exit if executed in SMM in VMX non-root operation. If executed in VMX root operation, it causes an invalid-opcode exception. SMM monitor uses VM entries to return from SMM (see Section 26.15.4).

26.15.4 VM Entries that Return from SMM

The SMM monitor returns from SMM using a VM entry with the “entry to SMM” VM-entry control clear. VM entries that return from SMM reverse the effects of an SMM VM exit (see Section 26.15.2).

VM entries that return from SMM may differ from other VM entries in that they do not necessarily enter VMX non-root operation. If the executive-VMCS pointer field in the current VMCS contains the VMXON pointer, the logical processor remains in VMX root operation after VM entry.

For differences between VM entries that return from SMM and other VM entries see Sections 26.15.4.1 through 26.15.4.10.

26.15.4.1 Checks on the Executive-VMCS Pointer Field

VM entries that return from SMM perform the following checks on the executive-VMCS pointer field in the current VMCS:

- Bits 11:0 must be 0.
- On processors that support Intel 64 architecture, the pointer must not set any bits beyond the processor’s physical-address width.[1] On processors that do not support Intel 64 architecture, it must not set any bits in the range 63:32.
- The 32 bits located in memory referenced by the physical address in the pointer must contain the processor’s VMCS revision identifier (see Section 20.2).

The checks above are performed before the checks described in Section 26.15.4.2 and before any of the following checks:

- If the “deactivate dual-monitor treatment” VM-entry control is 0, the launch state of the executive VMCS (the VMCS referenced by the executive-VMCS pointer field) must be launched (see Section 20.11).
- If the “deactivate dual-monitor treatment” VM-entry control is 1, the executive-VMCS pointer field must contain the VMXON pointer (see Section 26.15.7).[2]

26.15.4.2 Checks on VM-Execution Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-execution control fields specified in Section 22.2.1.1. They do not apply the checks to the current VMCS. Instead, VM-entry behavior depends on whether the executive-VMCS pointer field contains the VMXON pointer:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are not performed at all.

[1] Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

[2] An SMM monitor can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the checks are performed on the VM-execution control fields in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS). These checks are performed after checking the executive-VMCS pointer field itself (for proper alignment).

Other VM entries ensure that, if “activate VMX-preemption timer” VM-execution control is 0, the “save VMX-preemption timer value” VM-exit control is also 0. This check is not performed by VM entries that return from SMM.

26.15.4.3 Checks on VM-Entry Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-entry control fields specified in Section 22.2.1.3.

Specifically, if the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the following must **not** all hold for the VM-entry interruption-information field:

- the valid bit (bit 31) in the VM-entry interruption-information field is 1
- the interruption type (bits 10:8) is not 7 (other event); and
- the vector (bits 7:0) is not 0 (pending MTF VM exit).

26.15.4.4 Checks on Guest Non-Register State

For VM entries that return from SMM, the activity-state field must not indicate the wait-for-SIPI state if the executive-VMCS pointer field contains the VMXON pointer (the VM entry is to VMX root operation).

Section 22.3.1.5 includes the following check on the interruptibility-state field: bit 3 (blocking by NMI) must be 0 if the “virtual NMIs” VM-execution control is 1, the valid bit (bit 31) in the VM-entry interruption-information field is 1, and the interruption type (bits 10:8) in that field has value 2 (indicating NMI). VM entries that return from SMM modify this check based on whether the executive-VMCS pointer field contains the VMXON pointer:[1]

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), this check is not performed at all.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), this check is performed based on the setting of the “virtual NMIs” VM-execution control in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS).

[1] An SMM monitor can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

26.15.4.5 Loading Guest State

VM entries that return from SMM load the SMBASE register from the SMBASE field.

VM entries that return from SMM invalidate information in the TLBs and paging-structure caches for all VPIDs (see Chapter 24).

26.15.4.6 VMX-Preemption Timer

A VM entry that returns from SMM activates the VMX-preemption timer only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation) and the “activate VMX-preemption timer” VM-entry control is 1 in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field). In this case, VM entry starts the VMX-preemption timer with the value in the VMX-preemption timer-value field in the current VMCS.

26.15.4.7 Updating the Current-VMCS and SMM-Transfer VMCS Pointers

Successful VM entries (returning from SMM) load the SMM-transfer VMCS pointer with the current-VMCS pointer. Following this, they load the current-VMCS pointer from a field in the current VMCS:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the current-VMCS pointer is loaded from the VMCS-link pointer field.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the current-VMCS pointer is loaded with the value of the executive-VMCS pointer field.

If the VM entry successfully enters VMX non-root operation, the VM-execution controls in effect after the VM entry are those from the new current VMCS. This includes any structures external to the VMCS referenced by VM-execution control fields.

The updating of these VMCS pointers occurs before event injection. Event injection is determined, however, by the VM-entry control fields in the VMCS that was current when the VM entry commenced.

26.15.4.8 VM Exits Induced by VM Entry

Section 22.5.1.2 describes how the event-delivery process invoked by event injection may lead to a VM exit. Section 22.6.3 to Section 22.6.7 describe other situations that may cause a VM exit to occur immediately after a VM entry.

Whether these VM exits occur is determined by the VM-execution control fields in the current VMCS. For VM entries that return from SMM, they can occur only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation).

In this case, determination is based on the VM-execution control fields in the VMCS that is current after the VM entry. This is the VMCS referenced by the value of the executive-VMCS pointer field at the time of the VM entry (see Section 26.15.4.7). This VMCS also controls the delivery of such VM exits. Thus, VM exits induced by a VM entry returning from SMM are to the executive monitor and not to the SMM monitor.

26.15.4.9 SMI Blocking

VM entries that return from SMM determine the blocking of system-management interrupts (SMIs) as follows:

- If the "deactivate dual-monitor treatment" VM-entry control is 0, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.
- If the "deactivate dual-monitor treatment" VM-entry control is 1, the blocking of SMIs depends on whether the logical processor is in SMX operation:[1]
  - If the logical processor is in SMX operation, SMIs are blocked after VM entry.
  - If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

VM entries that return from SMM and that do not deactivate the dual-monitor treatment may leave SMIs blocked. This feature exists to allow an SMM monitor to invoke functionality outside of SMM without unblocking SMIs.

26.15.4.10 Failures of VM Entries That Return from SMM

Section 22.7 describes the treatment of VM entries that fail during or after loading guest state. Such failures record information in the VM-exit information fields and load processor state as would be done on a VM exit. The VMCS used is the one that was current before the VM entry commenced. Control is thus transferred to the SMM monitor and the logical processor remains in SMM.

26.15.5 Enabling the Dual-Monitor Treatment

Code and data for the SMM monitor reside in a region of SMRAM called the **monitor segment** (MSEG). Code running in SMM determines the location of MSEG and establishes its content. This code is also responsible for enabling the dual-monitor treatment.

[1] A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

SMM code enables the dual-monitor treatment and determines the location of MSEG by writing to IA32_SMM_MONITOR_CTL MSR (index 9BH). The MSR has the following format:

- Bit 0 is the register's valid bit. The SMM monitor may be invoked using VMCALL only if this bit is 1. Because VMCALL is used to activate the dual-monitor treatment (see Section 26.15.6), the dual-monitor treatment cannot be activated if the bit is 0. This bit is cleared when the logical processor is reset.
- Bits 11:1 are reserved.
- Bits 31:12 contain a value that, when shifted right 12 bits, is the physical address of MSEG (the MSEG base address).
- Bits 63:32 are reserved.

The following items detail use of this MSR:

- The IA32_SMM_MONITOR_CTL MSR is supported only on processors that support the dual-monitor treatment.[1] On other processors, accesses to the MSR using RDMSR or WRMSR generate a general-protection fault (#GP(0)).
- A write to the IA32_SMM_MONITOR_CTL MSR using WRMSR generates a general-protection fault (#GP(0)) if executed outside of SMM or if an attempt is made to set any reserved bit. An attempt to write to IA32_SMM_MONITOR_CTL MSR fails if made as part of a VM exit that does not end in SMM or part of a VM entry that does not begin in SMM.
- Reads from IA32_SMM_MONITOR_CTL MSR using RDMSR are allowed any time RDMSR is allowed. The MSR may be read as part of any VM exit.
- The dual-monitor treatment can be activated only if the valid bit in the MSR is set to 1.

The 32 bytes located at the MSEG base address are called the **MSEG header**. The format of the MSEG header is given in Table 26-10 (each field is 32 bits).

Table 26-10. Format of MSEG Header

Byte Offset	Field
0	MSEG-header revision identifier
4	SMM-monitor features
8	GDTR limit
12	GDTR base offset
16	CS selector
20	EIP offset

[1] Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix G.1) to determine whether the dual-monitor treatment is supported.

Table 26-10. Format of MSEG Header (Contd.)

Byte Offset	Field
24	ESP offset
28	CR3 offset

To ensure proper behavior in VMX operation, software should maintain the MSEG header in writeback cacheable memory. Future implementations may allow or require a different memory type.[1] Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix G.1).

SMM code should enable the dual-monitor treatment (by setting the valid bit in IA32_SMM_MONITOR_CTL MSR) only after establishing the content of the MSEG header as follows:

- Bytes 3:0 contain the **MSEG revision identifier**. Different processors may use different MSEG revision identifiers. These identifiers enable software to avoid using an MSEG header formatted for one processor on a processor that uses a different format. Software can discover the MSEG revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_MISC (see Appendix G.6).
- Bytes 7:4 contain the **SMM-monitor features** field. Bits 31:1 of this field are reserved and must be zero. Bit 0 of the field is the **IA-32e mode SMM feature bit**.[2] It indicates whether the logical processor will be in IA-32e mode after the SMM monitor is activated (see Section 26.15.6).
- Bytes 31:8 contain fields that determine how processor state is loaded when the SMM monitor is activated (see Section 26.15.6.4). SMM code should establish these fields so that activating of the SMM monitor invokes the SMM monitor's initialization code.

26.15.6 Activating the Dual-Monitor Treatment

The dual-monitor treatment may be enabled by SMM code as described in Section 26.15.5. The dual-monitor treatment is activated only if it is enabled and only by the executive monitor. The executive monitor activates the dual-monitor treatment by executing VMCALL in VMX root operation.

[1] Alternatively, software may map the MSEG header with the UC memory type; this may be necessary, depending on how memory is organized. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix G.1.

[2] Note that use of IA-32e mode address-translation mechanism is not currently supported in SMM. Thus, setting the IA-32e mode SMM feature bit to 1 is not currently supported. See note in Section 26.1.

When VMCALL activates the dual-monitor treatment, it causes an SMM VM exit. Differences between this SMM VM exit and other SMM VM exits are discussed in Sections 26.15.6.1 through 26.15.6.5. See also “VMCALL—Call to VM Monitor” in Chapter 6 of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*.

26.15.6.1 Initial Checks

An execution of VMCALL attempts to activate the dual-monitor treatment if (1) the processor supports the dual-monitor treatment;[1] (2) the logical processor is in VMX root operation; (3) the logical processor is outside SMM and the valid bit is set in the IA32_SMM_MONITOR_CTL MSR; (4) the logical processor is not in virtual-8086 mode and not in compatibility mode; (5) CPL = 0; and (6) the dual-monitor treatment is not active.

The VMCS that manages SMM VM exit caused by this VMCALL is the current VMCS established by the executive monitor. The VMCALL performs the following checks on the current VMCS in the order indicated:

1. There must be a current VMCS pointer.
2. The launch state of the current VMCS must be clear.
3. The VM-exit control fields must be valid:
   - Reserved bits in the VM-exit controls must be set properly. Software may consult the VMX capability MSR IA32_VMX_EXIT_CTLS to determine the proper settings (see Appendix G.4).
   - The following checks are performed for the VM-exit MSR-store address if the VM-exit MSR-store count field is non-zero:
     - The lower 4 bits of the VM-exit MSR-store address must be 0. On processors that support Intel 64 architecture, the address should not set any bits beyond the processor’s physical-address width.[2] On processors that do not support Intel 64 architecture, the address should not set any bits in the range 63:32.
     - On processors that support Intel 64 architecture, the address of the last byte in the VM-exit MSR-store area should not set any bits beyond the processor’s physical-address width. On processors that do not support Intel 64 architecture, the address of the last byte in the VM-exit MSR-store area should not set any bits in the range 63:32. The address of this last byte is VM-exit MSR-store address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)

[1] Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix G.1) to determine whether the dual-monitor treatment is supported.

[2] Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

If any of these checks fail, subsequent checks are skipped and VMCALL fails. If all these checks succeed, the logical processor uses the IA32_SMM_MONITOR_CTL MSR to determine the base address of MSEG. The following checks are performed in the order indicated:

1. The logical processor reads the 32 bits at the base of MSEG and compares them to the processor's MSEG revision identifier.
2. The logical processor reads the SMM-monitor features field:
   - Bit 0 of the field is the IA-32e mode SMM feature bit, and it indicates whether the logical processor will be in IA-32e mode after the SMM monitor is activated.
     - If the VMCALL is executed on a processor that does not support Intel 64 architecture, the IA-32e mode SMM feature bit must be 0.
     - If the VMCALL is executed in 64-bit mode, the IA-32e mode SMM feature bit must be 1.
   - Bits 31:1 of this field are currently reserved and must be zero.

If any of these checks fail, subsequent checks are skipped and the VMCALL fails.

26.15.6.2 MSEG Checking

SMM VM exits that activate the dual-monitor treatment check the following before updating the current-VMCS pointer and the executive-VMCS pointer field (see Section 26.15.2.2):

- The 32 bits at the MSEG base address (used as a physical address) must contain the processor's MSEG revision identifier.
- Bits 31:1 of the SMM-monitor features field in the MSEG header (see Table 26-10) must be 0. Bit 0 of the field (the IA-32e mode SMM feature bit) must be 0 if the processor does not support Intel 64 architecture.

If either of these checks fail, execution of VMCALL fails.

26.15.6.3 Updating the Current-VMCS and Executive-VMCS Pointers

Before performing the steps in Section 26.15.2.2, SMM VM exits that activate the dual-monitor treatment begin by loading the SMM-transfer VMCS pointer with the value of the current-VMCS pointer.

26.15.6.4 Loading Host State

The VMCS that is current during an SMM VM exit that activates the dual-monitor treatment was established by the executive monitor. It does not contain the VM-exit controls and host state required to initialize the SMM monitor. For this reason, such SMM VM exits do not load processor state as described in Section 23.5. Instead, state is set to fixed values or loaded based on the content of the MSEG header (see Table 26-10):

- CR0 is set to as follows:
  - — PG, NE, ET, MP, and PE are all set to 1.
  - — CD and NW are left unchanged.
  - — All other bits are cleared to 0.
- CR3 is set as follows:
  - — Bits 63:32 are cleared on processors that supports IA-32e mode.
  - — Bits 31:12 are set to bits 31:12 of the sum of the MSEG base address and the CR3-offset field in the MSEG header.
  - — Bits 11:5 and bits 2:0 are cleared (the corresponding bits in the CR3-offset field in the MSEG header are ignored).
  - — Bits 4:3 are set to bits 4:3 of the CR3-offset field in the MSEG header.
- CR4 is set as follows:
  - — MCE and PGE are cleared.
  - — PAE is set to the value of the IA-32e mode SMM feature bit.
  - — If the IA-32e mode SMM feature bit is clear, PSE is set to 1 if supported by the processor; if the bit is set, PSE is cleared.
  - — All other bits are unchanged.
- DR7 is set to 400H.
- The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
- The registers CS, SS, DS, ES, FS, and GS are loaded as follows:
  - — All registers are usable.
  - — CS.selector is loaded from the corresponding fields in the MSEG header (the high 16 bits are ignored), with bits 2:0 cleared to 0. If the result is 0000H, CS.selector is set to 0008H.
  - — The selectors for SS, DS, ES, FS, and GS are set to CS.selector+0008H. If the result is 0000H (if the CS selector was 0xFFF8), these selectors are instead set to 0008H.
  - — The base addresses of all registers are cleared to zero.
  - — The segment limits for all registers are set to FFFFFFFFH.
  - — The AR bytes for the registers are set as follows:
    - CS.Type is set to 11 (execute/read, accessed, non-conforming code segment).
    - For SS, DS, FS, and GS, the Type is set to 3 (read/write, accessed, expand-up data segment).

  - The S bits for all registers are set to 1.
  - The DPL for each register is set to 0.
  - The P bits for all registers are set to 1.
  - On processors that support Intel 64 architecture, CS.L is loaded with the value of the IA-32e mode SMM feature bit.
  - CS.D is loaded with the inverse of the value of the IA-32e mode SMM feature bit.
  - For each of SS, DS, FS, and GS, the D/B bit is set to 1.
  - The G bits for all registers are set to 1.
- LDTR is unusable. The LDTR selector is cleared to 0000H, and the register is otherwise undefined (although the base address is always canonical)
- GDTR.base is set to the sum of the MSEG base address and the GDTR base-offset field in the MSEG header (bits 63:32 are always cleared on processors that supports IA-32e mode). GDTR.limit is set to the corresponding field in the MSEG header (the high 16 bits are ignored).
- IDTR.base is unchanged. IDTR.limit is cleared to 0000H.
- RIP is set to the sum of the MSEG base address and the value of the RIP-offset field in the MSEG header (bits 63:32 are always cleared on logical processors that support IA-32e mode).
- RSP is set to the sum of the MSEG base address and the value of the RSP-offset field in the MSEG header (bits 63:32 are always cleared on logical processor that supports IA-32e mode).
- RFLAGS is cleared, except bit 1, which is always set.
- The logical processor is left in the active state.
- Event blocking after the SMM VM exit is as follows:
  - — There is no blocking by STI or by MOV SS.
  - — There is blocking by non-maskable interrupts (NMIs) and by SMIs.
- There are no pending debug exceptions after the SMM VM exit.
- For processors that support IA-32e mode, the IA32_EFER MSR is modified so that LME and LMA both contain the value of the IA-32e mode SMM feature bit.

If any of CR3[63:5], CR4.PAE, CR4.PSE, or IA32_EFER.LMA is changing, the TLBs are updated so that, after VM exit, the logical processor does not use translations that were cached before the transition. This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if IA32_EFER.LMA was 1 before and after the transition).

26.15.6.5 Loading MSRs

The VM-exit MSR-load area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are loaded from that area.

26.15.7 Deactivating the Dual-Monitor Treatment

An SMM monitor may deactivate the dual monitor treatment and return the processor to default treatment of SMIs and SMM (see Section 26.14). It does this by executing a VM entry with the “deactivate dual-monitor treatment” VM-entry control set to 1.

As noted in Section 22.2.1.3 and Section 26.15.4.1, an attempt to deactivate the dual-monitor treatment fails in the following situations: (1) the processor is not in SMM; (2) the “entry to SMM” VM-entry control is 1; or (3) the executive-VMCS pointer does not contain the VMXON pointer (the VM entry is to VMX non-root operation).

As noted in Section 26.15.4.9, VM entries that deactivate the dual-monitor treatment ignore the SMI bit in the interruptibility-state field of the guest-state area. Instead, such a VM entry unconditionally unmasks SMIs.

26.16 SMI AND PROCESSOR EXTENDED STATE MANAGEMENT

On processors that support processor extended states using XSAVE/XRSTOR (see Chapter 12, “System Programming for Instruction Set Extensions and Processor Extended States”), the processor does not save any XSAVE/XRSTOR related state on an SMI. It is the responsibility of the SMM handler code to properly preserve the state information (including CR4.OSXSAVE, the XFEATURE_ENABLED_MASK register, and possibly processor extended states using XSAVE/XRSTOR). Therefore, the SMM handler must follow the rules described in Chapter 12.

CHAPTER 27
VIRTUAL-MACHINE MONITOR PROGRAMMING CONSIDERATIONS

27.1 VMX SYSTEM PROGRAMMING OVERVIEW

The Virtual Machine Monitor (VMM) is a software class used to manage virtual machines (VM). This chapter describes programming considerations for VMMs.

Each VM behaves like a complete physical machine and can run operating system (OS) and applications. The VMM software layer runs at the most privileged level and has complete ownership of the underlying system hardware. The VMM controls creation of a VM, transfers control to a VM, and manages situations that can cause transitions between the guest VMs and host VMM. The VMM allows the VMs to share the underlying hardware and yet provides isolation between the VMs. The guest software executing in a VM is unaware of any transitions that might have occurred between the VM and its host.

27.2 SUPPORTING PROCESSOR OPERATING MODES IN GUEST ENVIRONMENTS

Typically, VMMs transfer control to a VM using VMX transitions referred to as VM entries. The boundary conditions that define what a VM is allowed to execute in isolation are specified in a virtual-machine control structure (VMCS).

As noted in Section 19.8, processors may fix certain bits in CR0 and CR4 to specific values and not support other values. The first processors to support VMX operation require that CR0.PE and CR0.PG be 1 in VMX operation. Thus, a VM entry is allowed only to guests with paging enabled that are in protected mode or in virtual-8086 mode. Guest execution in other processor operating modes need to be specially handled by the VMM.

One example of such a condition is guest execution in real-mode. A VMM could support guest real-mode execution using at least two approaches:

- By using a fast instruction set emulator in the VMM.

- By using the similarity between real-mode and virtual-8086 mode to support real-mode guest execution in a virtual-8086 container. The virtual-8086 container may be implemented as a virtual-8086 container task within a monitor that emulates real-mode guest state and instructions, or by running the guest VM as the virtual-8086 container (by entering the guest with RFLAGS.VM[1] set). Attempts by real-mode code to access privileged state outside the virtual-8086 container would trap to the VMM and would also need to be emulated.

Another example of such a condition is guest execution in protected mode with paging disabled. A VMM could support such guest execution by using “identity” page tables to emulate unpaged protected mode.

27.2.1 Emulating Guest Execution

In certain conditions, VMMs may resort to using a virtual-8086 container to support guest execution in operating modes not supported by VMX. But for other conditions, VMMs may need to resort to emulating guest execution.

These are example conditions that require guest emulation in the VMM:

- Programming conditions that are not allowed by the VMX consistency checks. Examples of this include transient conditions introduced when switching between real-mode and protected mode (where some segment may not be consistent with the operating mode).
- Conditions of guest task switching. Task switches always cause VM exits. To correctly advance the guest state, the monitor needs to emulate the guest task-switching behavior.
- When a SMM monitor is configured, conditions where the SMRAM is relocated to an address above 1 MByte (HSEG).
- When executing SMM code in a guest container by an SMM monitor. SMM processor operation allows address space ranges from 0-4 GBytes compared to the 1 MByte address space in real-mode operation. Also, the 64-KByte segment limit of real-mode is increased to 4 GBytes in SMM).

27.3 MANAGING VMCS REGIONS AND POINTERS

A VMM must observe necessary procedures when working with a VMCS, the associated VMCS pointer, and the VMCS region. It must also not assume the state of persistency for VMCS regions in memory or cache.

[1] This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.).

Before entering VMX operation, the host VMM allocates a VMXON region. A VMM can host several virtual machines and have many VMCSs active under its management. A unique VMCS region is required for each virtual machine; a VMXON region is required for the VMM itself.

A VMM determines the VMCS region size by reading IA32_VMX_BASIC MSR; it creates VMCS regions of this size using a 4-KByte-aligned area of physical memory. Each VMCS region needs to be initialized with a VMCS revision identifier (at byte offset 0) identical to the revision reported by the processor in the VMX capability MSR.

NOTE

Software must not read or write directly to the VMCS data region as the format is not architecturally defined. Consequently, we recommend that the VMM remove any linear-address mappings to VMCS regions before loading.

System software does not need to do special preparation to the VMXON region before entering into VMX operation. The address of the VMXON region for the VMM is provided as an operand to VMXON instruction. Once in VMX root operation, the VMM needs to prepare data fields in the VMCS that control the execution of a VM upon a VM entry. The VMM can make a VMCS the current VMCS by using the VMPTRLD instruction. VMCS data fields must be read or written only through VMREAD and VMWRITE commands respectively.

Every component of the VMCS is identified by a 32-bit encoding that is provided as an operand to VMREAD and VMWRITE. Appendix H provides the encodings. A VMM must properly initialize all fields in a VMCS before using the current VMCS for VM entry.

A VMCS is referred to as a controlling VMCS if it is the current VMCS on a logical processor in VMX non-root operation. A current VMCS for controlling a logical processor in VMX non-root operation may be referred to as a working VMCS if the logical processor is not in VMX non-root operation. The relationship of active, current (i.e. working) and controlling VMCS during VMX operation is shown in Figure 27-1.

The VMX capability MSR IA32_VMX_BASIC reports the memory type used by the processor for accessing a VMCS or any data structures referenced through pointers in the VMCS. Software must maintain the VMCS structures in cache-coherent memory. Software must always map the regions hosting the I/O bitmaps, MSR bitmaps, VM-exit MSR-store area, VM-exit MSR-load area, and VM-entry MSR-load area to the write-back (WB) memory type. Mapping these regions to uncacheable (UC) memory type is supported, but strongly discouraged due to negative impact on performance.

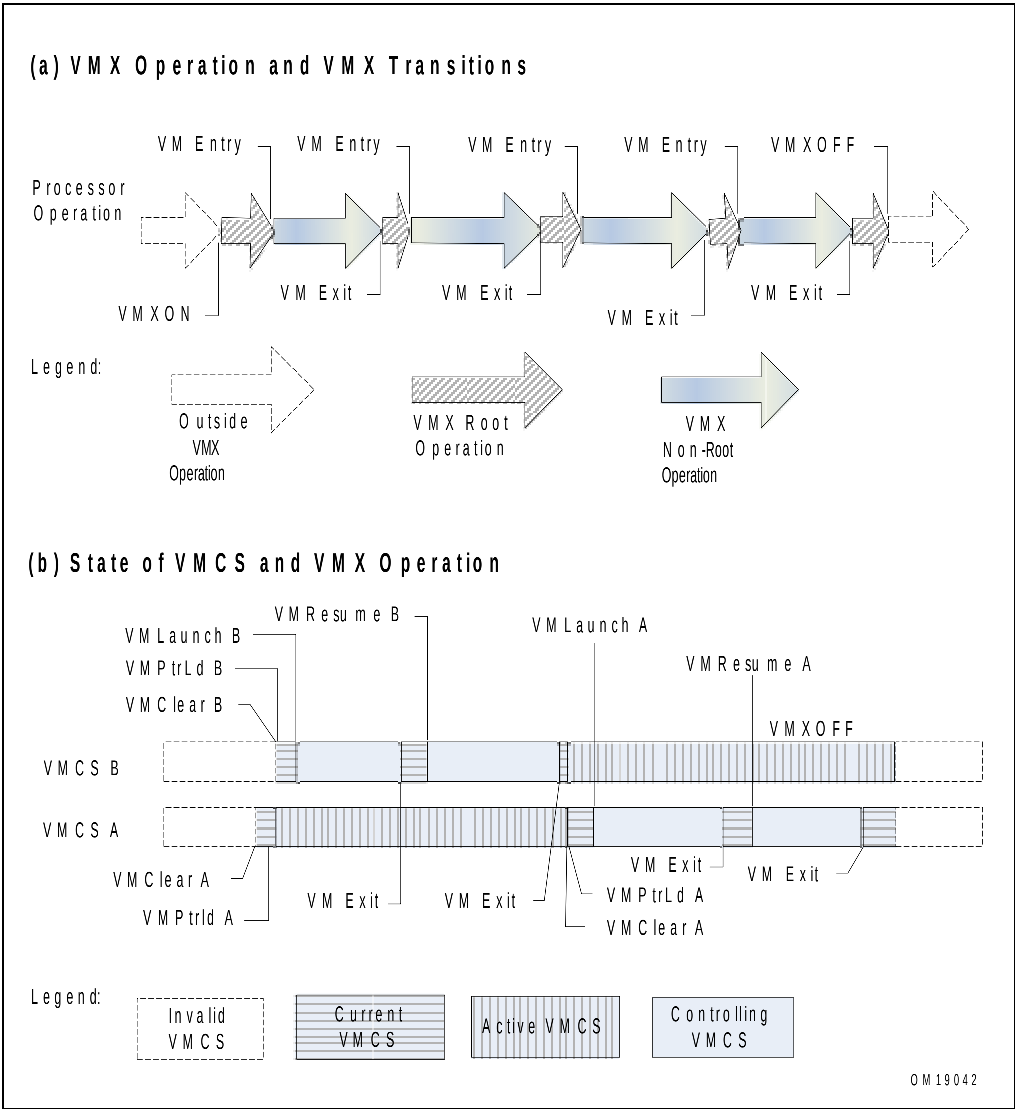

Figure 27-1. VMX Transitions and States of VMCS in a Logical Processor

27.4 USING VMX INSTRUCTIONS

VMX instructions are allowed only in VMX root operation. An attempt to execute a VMX instruction in VMX non-root operation causes a VM exit.

Processors perform various checks while executing any VMX instruction. They follow well-defined error handling on failures. VMX instruction execution failures detected before loading of a guest state are handled by the processor as follows:

- If the working-VMCS is not valid, the instruction fails by setting RFLAGS.CF = 1.
- If the working-VMCS pointer is valid, RFLAGS.ZF is set to value 1 and the proper error-code is saved in the VM-instruction error field of the working-VMCS.

Software is required to check RFLAGS.CF and RFLAGS.ZF to determine the success or failure of VMX instruction executions.

After a VM-entry instruction (VMRESUME or VMLAUNCH) successfully completes the general checks and checks on VMX controls and the host-state area (see Section 22.2), any errors encountered while loading of guest-state (due to bad guest-state or bad MSR loading) causes the processor to load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 27.7).

This failure behavior differs from that of VM exits in that no guest-state is saved to the guest-state area. A VMM can detect its VM-exit handler was invoked by such a failure by checking bit 31 (for 1) in the exit reason field of the working VMCS and further identify the failure by using the exit qualification field.

27.5 VMM SETUP & TEAR DOWN

VMMs need to ensure that the processor is running in protected mode with paging before entering VMX operation. The following list describes the minimal steps required to enter VMX root operation with a VMM running at CPL = 0.

- Check VMX support in processor using CPUID.
- Determine the VMX capabilities supported by the processor through the VMX capability MSRs. See Section 27.5.1 and Appendix G.
- Create a VMXON region in non-pageable memory of a size specified by IA32_VMX_BASIC MSR and aligned to a 4-KByte boundary. Software should read the capability MSRs to determine width of the physical addresses that may be used for the VMXON region and ensure the entire VMXON region can be addressed by addresses with that width. Also, software must ensure that the VMXON region is hosted in cache-coherent memory.
- Initialize the version identifier in the VMXON region (the first 32 bits) with the VMCS revision identifier reported by capability MSRs.
- Ensure the current processor operating mode meets the required CR0 fixed bits (CR0.PE = 1, CR0.PG = 1). Other required CR0 fixed bits can be detected through the IA32_VMX_CR0_FIXED0 and IA32_VMX_CR0_FIXED1 MSRs.

- Enable VMX operation by setting CR4.VMXE = 1. Ensure the resultant CR4 value supports all the CR4 fixed bits reported in the IA32_VMX_CR4_FIXED0 and IA32_VMX_CR4_FIXED1 MSRs.
- Ensure that the IA32_FEATURE_CONTROL MSR (MSR index 3AH) has been properly programmed and that its lock bit is set (Bit 0 = 1). This MSR is generally configured by the BIOS using WRMSR.
- Execute VMXON with the physical address of the VMXON region as the operand. Check successful execution of VMXON by checking if RFLAGS.CF = 0.

Upon successful execution of the steps above, the processor is in VMX root operation.

A VMM executing in VMX root operation and CPL = 0 leaves VMX operation by executing VMXOFF and verifies successful execution by checking if RFLAGS.CF = 0 and RFLAGS.ZF = 0.

If an SMM monitor has been configured to service SMIs while in VMX operation (see Section 26.15), the SMM monitor needs to be torn down before the executive monitor can leave VMX operation (see Section 26.15.7). VMXOFF fails for the executive monitor (a VMM that entered VMX operation by way of issuing VMXON) if SMM monitor is configured.

27.5.1 Algorithms for Determining VMX Capabilities

As noted earlier, a VMM should determine the VMX capabilities supported by the processor by reading the VMX capability MSRs. The architecture for these MSRs is detailed in Appendix G.

As noted in Chapter 20, “Virtual-Machine Control Structures”, certain VMX controls are reserved and must be set to a specific value (0 or 1) determined by the processor. The specific value to which a reserved control must be set is its **default setting**. Most controls have a default setting of 0; Appendix G.2 identifies those controls that have a default setting of 1. The term **default1** describes the class of controls whose default setting is 1. The are controls in this class from the pin-based VM-execution controls, the primary processor-based VM-execution controls, the VM-exit controls, and the VM-entry controls. There are no secondary processor-based VM-execution controls in the default1 class.

Future processors may define new functionality for one or more reserved controls. Such processors would allow each newly defined control to be set either to 0 or to 1. Software that does not desire a control’s new functionality should set the control to its default setting.

The capability MSRs IA32_VMX_PINBASED_CTLS, IA32_VMX_PROCBASED_CTLS, IA32_VMX_EXIT_CTLS, and IA32_VMX_ENTRY_CTLS report, respectively, on the allowed settings of most of the pin-based VM-execution controls, the primary processor-based VM-execution controls, the VM-exit controls, and the VM-entry controls. However, they will always report that any control in the default1 class must be 1. If a logical processor allows any control in the default1 class to be 0, it indicates this fact by returning 1 for the value of bit 55 of the IA32_VMX_BASIC MSR. If this bit is 1, the logical processor supports the capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. These capability MSRs report, respectively, on the allowed settings of all of the pin-based VM-execution controls, the primary processor-based VM-execution controls, the VM-exit controls, and the VM-entry controls.

Software may use one of the following high-level algorithms to determine the correct default control settings:[1]

1. The following algorithm does not use the details given in Appendix G.2:
   a. Ignore bit 55 of the IA32_VMX_BASIC MSR.
   b. Using RDMSR, read the VMX capability MSRs IA32_VMX_PINBASED_CTLS, IA32_VMX_PROCBASED_CTLS, IA32_VMX_EXIT_CTLS, and IA32_VMX_ENTRY_CTLS.
   c. Set the VMX controls as follows:
      i) If the relevant VMX capability MSR reports that a control has a single setting, use that setting.
      ii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is known to the VMM; then set the control based on functionality desired.
      iii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is not known to the VMM; then set the control to 0.

   A VMM using this algorithm will set to 1 all controls in the default1 class (in step (c)(i)). It will operate correctly even on processors that allow some controls in the default1 class to be 0. However, such a VMM will not be able to use the new features enabled by the 0-setting of such controls. For that reason, this algorithm is not recommended.

2. The following algorithm uses the details given in Appendix G.2. This algorithm requires software to know the identity of the controls in the default1 class:
   a. Using RDMSR, read the IA32_VMX_BASIC MSR.
   b. Use bit 55 of that MSR as follows:
      i) If bit 55 is 0, use RDMSR to read the VMX capability MSRs IA32_VMX_PINBASED_CTLS, IA32_VMX_PROCBASED_CTLS, IA32_VMX_EXIT_CTLS, and IA32_VMX_ENTRY_CTLS.
      ii) If bit 55 is 1, use RDMSR to read the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS,

[1] These algorithms apply only to the pin-based VM-execution controls, the primary processor-based VM-execution controls, the VM-exit controls, and the VM-entry controls. Because there are no secondary processor-based VM-execution controls in the default1 class, a VMM can always set to 0 any such control whose meaning is unknown to it.

IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS.

c. Set the VMX controls as follows:

   i) If the relevant VMX capability MSR reports that a control has a single setting, use that setting.

   ii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is known to the VMM; then set the control based on functionality desired.

   iii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; (2) the control's meaning is not known to the VMM; and (3) the control is not in the default1 class; then set the control to 0.

   iv) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; (2) the control's meaning is not known to the VMM; and (3) the control is in the default1 class; then set the control to 1.

A VMM using this algorithm will set to 1 all controls in default1 class whose meaning it does not know (either in step (c)(i) or step (c)(iv)). It will operate correctly even on processors that allow some controls in the default1 class to be 0. Unlike a VMM using Algorithm 1, a VMM using Algorithm 2 will be able to use the new features enabled by the 0-setting of such controls.

3. The following algorithm uses the details given in Appendix G.2. This algorithm does not require software to know the identity of the controls in the default1 class:

   a. Using RDMSR, read the VMX capability MSRs IA32_VMX_BASIC, IA32_VMX_PINBASED_CTLS, IA32_VMX_PROCBASED_CTLS, IA32_VMX_EXIT_CTLS, and IA32_VMX_ENTRY_CTLS.

   b. If bit 55 of the IA32_VMX_BASIC MSR is 0, set the VMX controls as follows:

      i) If the relevant VMX capability MSR reports that a control has a single setting, use that setting.

      ii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is known to the VMM; then set the control based on functionality desired.

      iii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is not known to the VMM; then set the control to 0.

   c. If bit 55 of the IA32_VMX_BASIC MSR is 1, use RDMSR to read the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. Set the VMX controls as follows:

      i) If the relevant VMX capability MSR just read reports that a control has a single setting, use that setting.

ii) If (1) the relevant VMX capability MSR just read reports that a control can be set to 0 or 1; and (2) the control's meaning is known to the VMM; then set the control based on functionality desired.

iii) If (1) the relevant VMX capability MSR just read reports that a control can be set to 0 or 1; (2) the control's meaning is not known to the VMM; and (3) the relevant VMX capability MSR as read in step (a) reports that a control can be set to 0; then set the control to 0.

iv) If (1) the relevant VMX capability MSR just read reports that a control can be set to 0 or 1; (2) the control's meaning is not known to the VMM; and (3) the relevant VMX capability MSR as read in step (a) reports that a control must be 1; then set the control to 1.

A VMM using this algorithm will set to 1 all controls in the default1 class whose meaning it does not know (in step (b)(i), step (c)(i), or step (c)(iv)). It will operate correctly even on processors that allow some controls in the default1 class to be 0. Unlike a VMM using Algorithm 1, a VMM using Algorithm 3 will be able to use the new features enabled by the 0-setting of such controls. Unlike a VMM using Algorithm 2, a VMM using Algorithm 3 need not know the identities of the controls in the default1 class.

27.6 PREPARATION AND LAUNCHING A VIRTUAL MACHINE

The following list describes the minimal steps required by the VMM to set up and launch a guest VM.

- Create a VMCS region in non-pageable memory of size specified by the VMX capability MSR IA32_VMX_BASIC and aligned to 4-KBytes. Software should read the capability MSRs to determine width of the physical addresses that may be used for a VMCS region and ensure the entire VMCS region can be addressed by addresses with that width. The term “guest-VMCS address” refers to the physical address of the new VMCS region for the following steps.
- Initialize the version identifier in the VMCS (first 32 bits) with the VMCS revision identifier reported by the VMX capability MSR IA32_VMX_BASIC.
- Execute the VMCLEAR instruction by supplying the guest-VMCS address. This will initialize the new VMCS region in memory and set the launch state of the VMCS to “clear”. This action also invalidates the working-VMCS pointer register to FFFFFFFF_FFFFFFFFH. Software should verify successful execution of VMCLEAR by checking if RFLAGS.CF = 0 and RFLAGS.ZF = 0.
- Execute the VMPTRLD instruction by supplying the guest-VMCS address. This initializes the working-VMCS pointer with the new VMCS region's physical address.
- Issue a sequence of VMWRITEs to initialize various host-state area fields in the working VMCS. The initialization sets up the context and entry-points to the VMM

upon subsequent VM exits from the guest. Host-state fields include control registers (CR0, CR3 and CR4), selector fields for the segment registers (CS, SS, DS, ES, FS, GS and TR), and base-address fields (for FS, GS, TR, GDTR and IDTR; RSP, RIP and the MSRs that control fast system calls).

Chapter 22 describes the host-state consistency checking done by the processor for VM entries. The VMM is required to set up host-state that comply with these consistency checks. For example, VMX requires the host-area to have a task register (TR) selector with TI and RPL fields set to 0 and pointing to a valid TSS.

- Use VMWRITEs to set up the various VM-exit control fields, VM-entry control fields, and VM-execution control fields in the VMCS. Care should be taken to make sure the settings of individual fields match the allowed 0 and 1 settings for the respective controls as reported by the VMX capability MSRs (see Appendix G). Any settings inconsistent with the settings reported by the capability MSRs will cause VM entries to fail.
- Use VMWRITE to initialize various guest-state area fields in the working VMCS. This sets up the context and entry-point for guest execution upon VM entry. Chapter 22 describes the guest-state loading and checking done by the processor for VM entries to protected and virtual-8086 guest execution.
- The VMM is required to set up guest-state that complies with these consistency checks:
  - — If the VMM design requires the initial VM launch to cause guest software (typically the guest virtual BIOS) execution from the guest's reset vector, it may need to initialize the guest execution state to reflect the state of a physical processor at power-on reset (described in Chapter 8, *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).
  - — The VMM may need to initialize additional guest execution state that is not captured in the VMCS guest-state area by loading them directly on the respective processor registers. Examples include general purpose registers, the CR2 control register, debug registers, floating point registers and so forth. VMM may support lazy loading of FPU, MMX, SSE, and SSE2 states with CR0.TS = 1 (described in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).
- Execute VMLAUNCH to launch the guest VM. If VMLAUNCH fails due to any consistency checks before guest-state loading, RFLAGS.CF or RFLAGS.ZF will be set and the VM-instruction error field (see Section 20.9.5) will contain the error-code. If guest-state consistency checks fail upon guest-state loading, the processor loads state from the host-state area as if a VM exit had occurred (see Section 27.6).

VMLAUNCH updates the controlling-VMCS pointer with the working-VMCS pointer and saves the old value of controlling-VMCS as the parent pointer. In addition, the launch state of the guest VMCS is changed to "launched" from "clear". Any programmed exit conditions will cause the guest to VM exit to the VMM. The VMM should execute VMRESUME instruction for subsequent VM entries to guests in a "launched" state.

27.7 HANDLING OF VM EXITS

This section provides examples of software steps involved in a VMM's handling of VM-exit conditions:

- Determine the exit reason through a VMREAD of the exit-reason field in the working-VMCS. Appendix I describes exit reasons and their encodings.
- VMREAD the exit-qualification from the VMCS if the exit-reason field provides a valid qualification. The exit-qualification field provides additional details on the VM-exit condition. For example, in case of page faults, the exit-qualification field provides the guest linear address that caused the page fault.
- Depending on the exit reason, fetch other relevant fields from the VMCS. Appendix I lists the various exit reasons.
- Handle the VM-exit condition appropriately in the VMM. This may involve the VMM emulating one or more guest instructions, programming the underlying host hardware resources, and then re-entering the VM to continue execution.

27.7.1 Handling VM Exits Due to Exceptions

As noted in Section 21.3, an exception causes a VM exit if the bit corresponding to the exception's vector is set in the exception bitmap. (For page faults, the error code also determines whether a VM exit occurs.) This section provide some guidelines of how a VMM might handle such exceptions.

Exceptions result when a logical processor encounters an unusual condition that software may not have expected. When guest software encounters an exception, it may be the case that the condition was caused by the guest software. For example, a guest application may attempt to access a page that is restricted to supervisor access. Alternatively, the condition causing the exception may have been established by the VMM. For example, a guest OS may attempt to access a page that the VMM has chosen to make not present.

When the condition causing an exception was established by guest software, the VMM may choose to **reflect** the exception to guest software. When the condition was established by the VMM itself, the VMM may choose to **resume** guest software after removing the condition.

27.7.1.1 Reflecting Exceptions to Guest Software

If the VMM determines that a VM exit was caused by an exception due to a condition established by guest software, it may reflect that exception to guest software. The VMM would cause the exception to be delivered to guest software, where it can be handled as it would be if the guest were running on a physical machine. This section describes how that may be done.

In general, the VMM can deliver the exception to guest software using VM-entry event injection as described in Section 22.5. The VMM can copy (using VMREAD and

VMWRITE) the contents of the VM-exit interruption-information field (which is valid, since the VM exit was caused by an exception) to the VM-entry interruption-information field (which, if valid, will cause the exception to be delivered as part of the next VM entry). The VMM would also copy the contents of the VM-exit interruption error-code field to the VM-entry exception error-code field; this need not be done if bit 11 (error code valid) is clear in the VM-exit interruption-information field. After this, the VMM can execute VMRESUME.

The following items provide details that may qualify the general approach:

- Care should be taken to ensure that reserved bits 30:12 in the VM-entry interruption-information field are 0. In particular, some VM exits may set bit 12 in the VM-exit interruption-information field to indicate NMI unblocking due to IRET. If this bit is copied as 1 into the VM-entry interruption-information field, the next VM entry will fail because that bit should be 0.
- Bit 31 (valid) of the IDT-vectoring information field indicates, if set, that the exception causing the VM exit occurred while another event was being delivered to guest software. If this is the case, it may not be appropriate simply to reflect that exception to guest software. To provide proper virtualization of the exception architecture, a VMM should handle nested events as a physical processor would. Processor handling is described in Chapter 5, “Interrupt 8—Double Fault Exception (#DF)” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.
    - — The VMM should reflect the exception causing the VM exit to guest software in any of the following cases:
        - The value of bits 10:8 (interruption type) of the IDT-vectoring information field is anything other than 3 (hardware exception).
        - The value of bits 7:0 (vector) of the IDT-vectoring information field indicates a benign exception (1, 2, 3, 4, 5, 6, 7, 9, 16, 17, 18, or 19).
        - The value of bits 7:0 (vector) of the VM-exit interruption-information field indicates a benign exception.
        - The value of bits 7:0 of the IDT-vectoring information field indicates a contributory exception (0, 10, 11, 12, or 13) and the value of bits 7:0 of the VM-exit interruption-information field indicates a page fault (14).
    - — If the value of bits 10:8 of the IDT-vectoring information field is 3 (hardware exception), the VMM should reflect a double-fault exception to guest software in any of the following cases:
        - The value of bits 7:0 of the IDT-vectoring information field and the value of bits 7:0 of the VM-exit interruption-information field each indicates a contributory exception.
        - The value of bits 7:0 of the IDT-vectoring information field indicates a page fault and the value of bits 7:0 of the VM-exit interruption-information field indicates either a contributory exception or a page fault.

A VMM can reflect a double-fault exception to guest software by setting the VM-entry interruption-information and VM-entry exception error-code fields as follows:

- Set bits 7:0 (vector) of the VM-entry interruption-information field to 8 (#DF).
- Set bits 10:8 (interruption type) of the VM-entry interruption-information field to 3 (hardware exception).
- Set bit 11 (deliver error code) of the VM-entry interruption-information field to 1.
- Clear bits 30:12 (reserved) of VM-entry interruption-information field.
- Set bit 31 (valid) of VM-entry interruption-information field.
- Set the VM-entry exception error-code field to zero.

— If the value of bits 10:8 of the IDT-vectoring information field is 3 (hardware exception) and the value of bits 7:0 is 8 (#DF), guest software would have encountered a triple fault. Event injection should not be used in this case. The VMM may choose to terminate the guest, or it might choose to enter the guest in the shutdown activity state.

27.7.1.2 Resuming Guest Software after Handling an Exception

If the VMM determines that a VM exit was caused by an exception due to a condition established by the VMM itself, it may choose to resume guest software after removing the condition. The approach for removing the condition may be specific to the VMM's software architecture. and algorithms This section describes how guest software may be resumed after removing the condition.

In general, the VMM can resume guest software simply by executing VMRESUME. The following items provide details of cases that may require special handling:

- If the “NMI exiting” VM-execution control is 0, bit 12 of the VM-exit interruption-information field indicates that the VM exit was due to a fault encountered during an execution of the IRET instruction that unblocked non-maskable interrupts (NMIs). In particular, it provides this indication if the following are both true:

  — Bit 31 (valid) in the IDT-vectoring information field is 0.

  — The value of bits 7:0 (vector) of the VM-exit interruption-information field is not 8 (the VM exit is not due to a double-fault exception).

  If both are true and bit 12 of the VM-exit interruption-information field is 1, NMIs were blocked before guest software executed the IRET instruction that caused the fault that caused the VM exit. The VMM should set bit 3 (blocking by NMI) in the interruptibility-state field (using VMREAD and VMWRITE) before resuming guest software.

- If the “virtual NMIs” VM-execution control is 1, bit 12 of the VM-exit interruption-information field indicates that the VM exit was due to a fault encountered during an execution of the IRET instruction that removed virtual-NMI blocking. In particular, it provides this indication if the following are both true:

- — Bit 31 (valid) in the IDT-vectoring information field is 0.
- — The value of bits 7:0 (vector) of the VM-exit interruption-information field is not 8 (the VM exit is not due to a double-fault exception).

If both are true and bit 12 of the VM-exit interruption-information field is 1, there was virtual-NMI blocking before guest software executed the IRET instruction that caused the fault that caused the VM exit. The VMM should set bit 3 (blocking by NMI) in the interruptibility-state field (using VMREAD and VMWRITE) before resuming guest software.

- Bit 31 (valid) of the IDT-vectoring information field indicates, if set, that the exception causing the VM exit occurred while another event was being delivered to guest software. The VMM should ensure that the other event is delivered when guest software is resumed. It can do so using the VM-entry event injection described in Section 22.5 and detailed in the following paragraphs:
  - — The VMM can copy (using VMREAD and VMWRITE) the contents of the IDT-vectoring information field (which is presumed valid) to the VM-entry interruption-information field (which, if valid, will cause the exception to be delivered as part of the next VM entry).
    - The VMM should ensure that reserved bits 30:12 in the VM-entry interruption-information field are 0. In particular, the value of bit 12 in the IDT-vectoring information field is undefined after all VM exits. If this bit is copied as 1 into the VM-entry interruption-information field, the next VM entry will fail because the bit should be 0.
    - If the "virtual NMIs" VM-execution control is 1 and the value of bits 10:8 (interruption type) in the IDT-vectoring information field is 2 (indicating NMI), the VM exit occurred during delivery of an NMI that had been injected as part of the previous VM entry. In this case, bit 3 (blocking by NMI) will be 1 in the interruptibility-state field in the VMCS. The VMM should clear this bit; otherwise, the next VM entry will fail (see Section 22.3.1.5).
  - — The VMM can also copy the contents of the IDT-vectoring error-code field to the VM-entry exception error-code field. This need not be done if bit 11 (error code valid) is clear in the IDT-vectoring information field.
  - — The VMM can also copy the contents of the VM-exit instruction-length field to the VM-entry instruction-length field. This need be done only if bits 10:8 (interruption type) in the IDT-vectoring information field indicate either software interrupt, privileged software exception, or software exception.

27.8 MULTI-PROCESSOR CONSIDERATIONS

The most common VMM design will be the symmetric VMM. This type of VMM runs the same VMM binary on all logical processors. Like a symmetric operating system, the symmetric VMM is written to ensure all critical data is updated by only one processor at a time, IO devices are accessed sequentially, and so forth. Asymmetric VMM designs are possible. For example, an asymmetric VMM may run its scheduler on one processor and run just enough of the VMM on other processors to allow the correct execution of guest VMs. The remainder of this section focuses on the multi-processor considerations for a symmetric VMM.

A symmetric VMM design does not preclude asymmetry in its operations. For example, a symmetric VMM can support asymmetric allocation of logical processor resources to guests. Multiple logical processors can be brought into a single guest environment to support an MP-aware guest OS. Because an active VMCS can not control more than one logical processor simultaneously, a symmetric VMM must make copies of its VMCS to control the VM allocated to support an MP-aware guest OS. Care must be taken when accessing data structures shared between these VMCSs. See Section 27.8.4.

Although it may be easier to develop a VMM that assumes a fully-symmetric view of hardware capabilities (with all processors supporting the same processor feature sets, including the same revision of VMX), there are advantages in developing a VMM that comprehends different levels of VMX capability (reported by VMX capability MSRs). One possible advantage of such an approach could be that an existing software installation (VMM and guest software stack) could continue to run without requiring software upgrades to the VMM, when the software installation is upgraded to run on hardware with enhancements in the processor's VMX capabilities. Another advantage could be that a single software installation image, consisting of a VMM and guests, could be deployed to multiple hardware platforms with varying VMX capabilities. In such cases, the VMM could fall back to a common subset of VMX features supported by all VMX revisions, or choose to understand the asymmetry of the VMX capabilities and assign VMs accordingly.

This section outlines some of the considerations to keep in mind when developing an MP-aware VMM.

27.8.1 Initialization

Before enabling VMX, an MP-aware VMM must check to make sure that all processors in the system are compatible and support features required. This can be done by:

- Checking the CPUID on each logical processor to ensure VMX is supported and that the overall feature set of each logical processor is compatible.
- Checking VMCS revision identifiers on each logical processor.
- Checking each of the “allowed-1” or “allowed-0” fields of the VMX capability MSR’s on each processor.

27.8.2 Moving a VMCS Between Processors

An MP-aware VMM is free to assign any logical processor to a VM. But for performance considerations, moving a guest VMCS to another logical processor is slower than resuming that guest VMCS on the same logical processor. Certain VMX performance features (such as caching of portions of the VMCS in the processor) are optimized for a guest VMCS that runs on the same logical processor.

The reasons are:

- To restart a guest on the same logical processor, a VMM can use VMRESUME. VMRESUME is expected to be faster than VMLAUNCH in general.
- To migrate a VMCS to another logical processor, a VMM must use the sequence of VMCLEAR, VMPTRLD and VMLAUNCH.
- Operations involving VMCLEAR can impact performance negatively. See Section 20.11.

A VMM scheduler should make an effort to schedule a guest VMCS to run on the logical processor where it last ran. Such a scheduler might also benefit from doing lazy VMCLEARs (that is: performing a VMCLEAR on a VMCS only when the scheduler knows the VMCS is being moved to a new logical processor). The remainder of this section describes the steps a VMM must take to move a VMCS from one processor to another.

A VMM must check the VMCS revision identifier in the VMX capability MSR IA32_VMX_BASIC to determine if the VMCS regions are identical between all logical processors. If the VMCS regions are identical (same revision ID) the following sequence can be used to move or copy the VMCS from one logical processor to another:

- Perform a VMCLEAR operation on the source logical processor. This ensures that all VMCS data that may be cached by the processor are flushed to memory.
- Copy the VMCS region from one memory location to another location. This is an optional step assuming the VMM wishes to relocate the VMCS or move the VMCS to another system.
- Perform a VMPTRLD of the physical address of VMCS region on the destination processor to establish its current VMCS pointer.

If the revision identifiers are different, each field must be copied to an intermediate structure using individual reads (VMREAD) from the source fields and writes (VMWRITE) to destination fields. Care must be taken on fields that are hard-wired to certain values on some processor implementations.

27.8.3 Paired Index-Data Registers

A VMM may need to virtualize hardware that is visible to software using paired index-data registers. Paired index-data register interfaces, such as those used in PCI (CF8, CFC), require special treatment in cases where a VM performing writes to these pairs can be moved during execution. In this case, the index (e.g. CF8) should be part of the virtualized state. If the VM is moved during execution, writes to the index should be redone so subsequent data reads/writes go to the right location.

27.8.4 External Data Structures

Certain fields in the VMCS point to external data structures (for example: the MSR bitmap, the I/O bitmaps). If a logical processor is in VMX non-root operation, none of the external structures referenced by that logical processor's current VMCS should be modified by any logical processor or DMA. Before updating one of these structures, the VMM must ensure that no logical processor whose current VMCS references the structure is in VMX non-root operation.

If a VMM uses multiple VMCS with each VMCS using separate external structures, and these structures must be kept synchronized, the VMM must apply the same care to updating these structures.

27.8.5 CPUID Emulation

CPUID reports information that is used by OS and applications to detect hardware features. It also provides multi-threading/multi-core configuration information. For example, MP-aware OSs rely on data reported by CPUID to discover the topology of logical processors in a platform (see Section 7.10, “Programming Considerations for Hardware Multi-Threading Capable Processors,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*).

If a VMM is to support asymmetric allocation of logical processor resources to guest OSs that are MP aware, then the VMM must emulate CPUID for its guests. The emulation of CPUID by the VMM must ensure the guest’s view of CPUID leaves are consistent with the logical processor allocation committed by the VMM to each guest OS.

27.9 32-BIT AND 64-BIT GUEST ENVIRONMENTS

For the most part, extensions provided by VMX to support virtualization are orthogonal to the extensions provided by Intel 64 architecture. There are considerations that impact VMM designs. These are described in the following subsections.

27.9.1 Operating Modes of Guest Environments

For Intel 64 processors, VMX operation supports host and guest environments that run in IA-32e mode or without IA-32e mode. VMX operation also supports host and guest environments on IA-32 processors.

A VMM entering VMX operation while IA-32e mode is active is considered to be an IA-32e mode host. A VMM entering VMX operation while IA-32e mode is not activated or not available is referred to as a 32-bit VMM. The type of guest operations such VMMs support are summarized in Table 27-1.

Table 27-1. Operating Modes for Host and Guest Environments

Capability	Guest Operation in IA-32e mode	Guest Operation Not Requiring IA-32e Mode
IA-32e mode VMM	Yes	Yes
32-bit VMM	Not supported	Yes

A VM exit may occur to an IA-32e mode guest in either 64-bit sub-mode or compatibility sub-mode of IA-32e mode. VMMs may resume guests in either mode. The sub-mode in which an IA-32e mode guest resumes VMX non-root operation is determined by the attributes of the code segment which experienced the VM exit. If CS.L = 1, the guest is executing in 64-bit mode; if CS.L = 0, the guest is executing in compatibility mode (see Section 27.9.5).

Not all of an IA-32e mode VMM must run in 64-bit mode. While some parts of an IA-32e mode VMM must run in 64-bit mode, there are only a few restrictions preventing a VMM from executing in compatibility mode. The most notable restriction is that most VMX instructions cause exceptions when executed in compatibility mode.

27.9.2 Handling Widths of VMCS Fields

Individual VMCS control fields must be accessed using VMREAD or VMWRITE instructions. Outside of 64-Bit mode, VMREAD and VMWRITE operate on 32 bits of data. The widths of VMCS control fields may vary depending on whether a processor supports Intel 64 architecture.

Many VMCS fields are architected to extend transparently on processors supporting Intel 64 architecture (64 bits on processors that support Intel 64 architecture, 32 bits on processors that do not). Some VMCS fields are 64-bits wide regardless of whether the processor supports Intel 64 architecture or is in IA-32e mode.

27.9.2.1 Natural-Width VMCS Fields

Many VMCS fields operate using natural width. Such fields return (on reads) and set (on writes) 32-bits when operating in 32-bit mode and 64-bits when operating in 64-bit mode. For the most part, these fields return the naturally expected data widths. The “Guest RIP” field in the VMCS guest-state area is an example of this type of field.

27.9.2.2 64-Bit VMCS Fields

Unlike natural width fields, these fields are fixed to 64-bit width on all processors. When in 64-bit mode, reads of these fields return 64-bit wide data and writes to these fields write 64-bits. When outside of 64-bit mode, reads of these fields return the low 32-bits and writes to these fields write the low 32-bits and zero the upper 32-bits. Should a non-IA-32e mode host require access to the upper 32-bits of these fields, a separate VMCS encoding is used when issuing VMREAD/VMWRITE instructions.

The VMCS control field “MSR bitmap address” (which contains the physical address of a region of memory which specifies which MSR accesses should generate VM-exits) is an example of this type of field. Specifying encoding 00002004H to VMREAD returns the lower 32-bits to non-IA-32e mode hosts and returns 64-bits to 64-bit hosts. The separate encoding 00002005H returns only the upper 32-bits.

27.9.3 IA-32e Mode Hosts

An IA-32e mode host is required to support 64-bit guest environments. Because activating IA-32e mode currently requires that paging be disabled temporarily and VMX entry requires paging to be enabled, IA-32e mode must be enabled before entering VMX operation. For this reason, it is not possible to toggle in and out of IA-32e mode in a VMM.

Section 27.5 describes the steps required to launch a VMM. An IA-32e mode host is also required to set the “Host Address-Space Size” VMCS VM-exit control to 1. The value of this control is then loaded in the IA32_EFER.LME/LMA and CS.L bits on each VM exit. This establishes a 64-bit host environment as execution transfers to the VMM entry point. At a minimum, the entry point is required to be in a 64-bit code segment. Subsequently, the VMM can, if it chooses, switch to 32-bit compatibility mode on a code-segment basis (see Section 27.9.1). Note, however, that VMX instructions other than VMCALL are not supported in compatibility mode; they generate an invalid opcode exception if used.

The following VMCS controls determine the value of IA32_EFER when a VM exit occurs: the “Host Address-Space Size” control (described above), the “VM-exit MSR-load count,” and the “VM-exit MSR-load address” (see Section 23.3). The loading of IA32_EFER.LME/LMA and CS.L bits established by the “Host Address-Space Size” control precede any loading of the IA32_EFER MSR due from the VM-exit MSR-load area. If IA32_EFER is specified in the VM-exit MSR-load area, the value of the LME bit in the load image of IA32_EFER should match the setting of the “Host Address-Space Size” control. Otherwise the attempt to modify the LME bit (while paging is enabled) will lead to a VMX-abort.

On the other hand, the IA32_EFER.LMA bit is always set by the processor (determined by the value of the LME bit, the CR0.PG bit, and the CR4.PAE bit) regardless of any value specified in the load image of the IA32_EFER MSR. For these and performance reasons, VMM writers may choose to not use the VM-exit/entry MSR-load/save areas for IA32_EFER.

On a VMM teardown, VMX operation should be exited before deactivating IA-32e mode if the latter is required.

27.9.4 IA-32e Mode Guests

A 32-bit guest can be launched by either IA-32e-mode hosts or non-IA-32e-mode hosts. A 64-bit guests can only be launched by a IA-32e-mode host.

In addition to the steps outlined in Section 27.6, VMM writers need to:

- Set the “IA-32e-Mode Guest” VM-entry control to 1 in the VMCS to assure VM-entry (VMLAUNCH or VMRESUME) will establish a 64-bit (or 32-bit compatible) guest operating environment.
- Enable paging (CR0.PG) and PAE mode (CR4.PAE) to assure VM-entry to a 64-bit guest will succeed.
- Ensure that the host to be in IA-32e mode (the IA32_EFER.LMA must be set to 1) and the setting of the VM-exit “Host Address-Space Size” control bit in the VMCS must also be set to 1.

If each of the above conditions holds true, then VM-entry will copy the value of the VM-entry “IA-32e-Mode Guest” control bit into the guests IA32_EFER.LME bit which will result in subsequent activation of IA-32e mode. If any of the above conditions is false, the VM-entry will fail and load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 22.7).

The following VMCS controls determine the value of IA32_EFER on a VM entry: the “IA-32e-Mode Guest” VM-entry control (described above), the “VM-entry MSR-load count,” and the “VM-entry MSR-load address” (see Section 22.4).

The loading of IA32_EFER.LME bit (described above) precedes any loading of the IA32_EFER MSR from the VM-entry MSR-load area of the VMCS. If loading of IA32_EFER is specified in the VM-entry MSR-load area, the value of the LME bit in the load image should be match the setting of the “IA-32e-Mode Guest” VM-entry control. Otherwise, the attempt to modify the LME bit (while paging is enabled) results in a failed VM entry.

On the other hand, the IA32_EFER.LMA bit is always set by the processor (determined by the value of the LME bit, the CR0.PG bit, and the CR4.PAE bit) regardless of any value specified in the load image of IA32_EFER. For these and performance reasons, VMM writers may choose to not use the VM-exit/entry MSR-load/save areas for IA32_EFER MSR.

Note that the VMM can control the processor’s architectural state when transferring control to a VM. VMM writers may choose to launch guests in protected mode and subsequently allow the guest to activate IA-32e mode or they may allow guests to toggle in and out of IA-32e mode. In this case, the VMM should require VM exit on accesses to the IA32_EFER MSR to detect changes in the operating mode and modify the VM-entry “IA-32e-Mode Guest” control accordingly.

A VMM should save/restore the extended (full 64-bit) contents of the guest general-purpose registers, the new general-purpose registers (R8-R15) and the SIMD registers introduced in 64-bit mode should it need to modify these upon VM exit.

27.9.5 32-Bit Guests

To launch or resume a 32-bit guest, VMM writers can follow the steps outlined in Section 27.6, making sure that the “IA-32e-Mode Guest” VM-entry control bit is set to 0. Then the “IA-32e-Mode Guest” control bit is copied into the guest IA32_EFER.LME bit, establishing IA32_EFER.LMA as 0.

27.10 HANDLING MODEL SPECIFIC REGISTERS

Model specific registers (MSR) provide a wide range of functionality. They affect processor features, control the programming interfaces, or are used in conjunction with specific instructions. As part of processor virtualization, a VMM may wish to protect some or all MSR resources from direct guest access.

VMX operation provides the following features to virtualize processor MSRs.

27.10.1 Using VM-Execution Controls

Processor-based VM-execution controls provide two levels of support for handling guest access to processor MSRs using RDMSR and WRMSR:

- **MSR bitmaps**: In VMX implementations that support a 1-setting (see Appendix G) of the user-MSR-bitmaps execution control bit, MSR bitmaps can be used to provide flexibility in managing guest MSR accesses. The MSR-bitmap-address in the guest VMCS can be programmed by VMM to point to a bitmap region which specifies VM-exit behavior when reading and writing individual MSRs.

  MSR bitmaps form a 4-KByte region in physical memory and are required to be aligned to a 4-KByte boundary. The first 1-KByte region manages read control of MSRs in the range 00000000H-00001FFFH; the second 1-KByte region covers read control of MSR addresses in the range C0000000H-C0001FFFH. The bitmaps for write control of these MSRs are located in the 2-KByte region immediately following the read control bitmaps. While the MSR bitmap address is part of VMCS, the MSR bitmaps themselves are not. This implies MSR bitmaps are not accessible through VMREAD and VMWRITE instructions but rather by using ordinary memory writes. Also, they are not specially cached by the processor and may be placed in normal cache-coherent memory by the VMM.

  When MSR bitmap addresses are properly programmed and the use-MSR-bitmap control (see Section 20.6.2) is set, the processor consults the associated bit in the appropriate bitmap on guest MSR accesses to the corresponding MSR and causes a VM exit if the bit in the bitmap is set. Otherwise, the access is permitted to proceed. This level of protection may be utilized by VMMs to selectively allow guest access to some MSRs while virtualizing others.

- **Default MSR protection**: If the use-MSR-bitmap control is not set, an attempt by a guest to access any MSR causes a VM exit. This also occurs for any attempt to access an MSR outside the ranges identified above (even if the use-MSR-bitmap control is set).

VM exits due to guest MSR accesses may be identified by the VMM through VM-exit reason codes. The MSR-read exit reason implies guest software attempted to read an MSR protected either by default or through MSR bitmaps. The MSR-write exit reason implies guest software attempting to write a MSR protected through the VM-execution controls. Upon VM exits caused by MSR accesses, the VMM may virtualize the guest MSR access through emulation of RDMSR/WRMSR.

27.10.2 Using VM-Exit Controls for MSRs

If a VMM allows its guest to access MSRs directly, the VMM may need to store guest MSR values and load host MSR values for these MSRs on VM exits. This is especially true if the VMM uses the same MSRs while in VMX root operation.

A VMM can use the VM-exit MSR-store-address and the VM-exit MSR-store-count exit control fields (see Section 20.7.2) to manage how MSRs are stored on VM exits. The VM-exit MSR-store-address field contains the physical address (16-byte aligned) of the VM-exit MSR-store area (a table of entries with 16 bytes per entry). Each table entry specifies an MSR whose value needs to be stored on VM exits. The VM-exit MSR-store-count contains the number of entries in the table.

Similarly the VM-exit MSR-load-address and VM-exit MSR-load-count fields point to the location and size of the VM-exit MSR load area. The entries in the VM-exit MSR-load area contain the host expected values of specific MSRs when a VM exit occurs.

Upon VM-exit, bits 127:64 of each entry in the VM-exit MSR-store area is updated with the contents of the MSR indexed by bits 31:0. Also, bits 127:64 of each entry in the VM-exit MSR-load area is updated by loading with values from bits 127:64 the contents of the MSR indexed by bits 31:0.

27.10.3 Using VM-Entry Controls for MSRs

A VMM may require specific MSRs to be loaded explicitly on VM entries while launching or resuming guest execution. The VM-entry MSR-load-address and VM-entry MSR-load-count entry control fields determine how MSRs are loaded on VM-entries. The VM-entry MSR-load-address and count fields are similar in structure and function to the VM-exit MSR-load address and count fields, except the MSR loading is done on VM-entries.

27.10.4 Handling Special-Case MSRs and Instructions

A number of instructions make use of designated MSRs in their operation. The VMM may need to consider saving the states of those MSRs. Instructions that merit such consideration include SYSENTER/SYSEXIT, SYSCALL/SYSRET, SWAPGS.

27.10.4.1 Handling IA32_EFER MSR

The IA32_EFER MSR provides bit fields that allow system software to enable processor features. For example: the SCE bit enables SYSCALL/SYSRET and the NXE bit enables Execute-Disable-Bit functionality.

VMX provides hardware support to preserve the values of these bits upon a VM entry after a VM exit, such that it does not require VMM to modify these bits in IA32_EFER.

27.10.4.2 Handling the SYSENTER and SYSEXIT Instructions

The SYSENTER and SYSEXIT instructions use three dedicated MSRs (i.e. IA32_SYSENTER_CS, IA32_SYSENTER_ESP and IA32_SYSENTER_EIP) to manage fast system calls. These MSRs may be utilized by both the VMM and the guest OS to manage system calls in VMX root operation and VMX non-root operation respectively.

VMX provides special handling of these MSRs on VM exits and VM entries:

- The save-SYSENTER-MSRs VM-Exit control field can be set to 1 to save these MSRs to guest-state area in VMCS on VM-exits.
- The load-SYSENTER-MSRs VM-exit control field allows the processor to load these MSRs from values saved in the host-state area of the VMCS on VM-exits.

The load-SYSENTER-MSRs VM-Entry control field allows loading of the SYSENTER MSRs from guest-state area of the VMCS on VM entries.

27.10.4.3 Handling the SYSCALL and SYSRET Instructions

The SYSCALL/SYSRET instructions are similar to SYSENTER/SYSEXIT but are designed to operate within the context of a 64-bit flat code segment. They are available only in 64-bit mode and only when the SCE bit of the IA32_EFER MSR is set. SYSCALL/SYSRET invocations can occur from either 32-bit compatibility mode application code or from 64-bit application code. Three related MSR registers (IA32_STAR, IA32_LSTAR, IA32_FMASK) are used in conjunction with fast system calls/returns that use these instructions.

64-Bit hosts which make use of these instructions in the VMM environment will need to save the guest state of the above registers on VM exit, load the host state, and restore the guest state on VM entry. One possible approach is to use the VM-exit MSR-save and MSR-load areas and the VM-entry MSR-load area defined by controls in the VMCS. A disadvantage to this approach, however, is that the approach results in the unconditional saving, loading, and restoring of MSR registers on each VM exit or VM entry.

Depending on the design of the VMM, it is likely that many VM-exits will require no fast system call support but the VMM will be burdened with the additional overhead of saving and restoring MSRs if the VMM chooses to support fast system call uniformly. Further, even if the host intends to support fast system calls during a VM-exit, some of the MSR values (such as the setting of the SCE bit in IA32_EFER) may not require modification as they may already be set to the appropriate value in the guest.

For performance reasons, a VMM may perform lazy save, load, and restore of these MSR values on certain VM exits when it is determined that this is acceptable. The lazy-save-load-restore operation can be carried out “manually” using RDMSR and WRMSR.

27.10.4.4 Handling the SWAPGS Instruction

The SWAPGS instruction is available only in 64-bit mode. It swaps the contents of two specific MSRs (IA32_GSBASE and IA32_KERNEL_GSBASE). The IA32_GSBASE MSR shadows the base address portion of the GS descriptor register; the IA32_KERNEL_GSBASE MSR holds the base address of the GS segment used by the kernel (typically it houses kernel structures). SWAPGS is intended for use with fast system calls when in 64-bit mode to allow immediate access to kernel structures on transition to kernel mode.

Similar to SYSCALL/SYSRET, IA-32e mode hosts which use fast system calls may need to save, load, and restore these MSR registers on VM exit and VM entry using the guidelines discussed in previous paragraphs.

27.10.4.5 Implementation Specific Behavior on Writing to Certain MSRs

As noted in Section 22.4 and Section 23.4, a processor may prevent writing to certain MSRs when loading guest states on VM entries or storing guest states on VM exits. This is done to ensure consistent operation. The subset and number of MSRs subject to restrictions are implementation specific. For initial VMX implementations, there are two MSRs: IA32_BIOS_UPDT_TRIG and IA32_BIOS_SIGN_ID (see Appendix B).

27.10.5 Handling Accesses to Reserved MSR Addresses

Privileged software (either a VMM or a guest OS) can access a model specific register by specifying addresses in MSR address space. VMMs, however, must prevent a guest from accessing reserved MSR addresses in MSR address space.

Consult Appendix B for lists of supported MSRs and their usage. Use the MSR bitmap control to cause a VM exit when a guest attempts to access a reserved MSR address. The response to such a VM exit should be to reflect #GP(0) back to the guest.

27.11 HANDLING ACCESSES TO CONTROL REGISTERS

Bit fields in control registers (CR0, CR4) control various aspects of processor operation. The VMM must prevent guests from modifying bits in CR0 or CR4 that are reserved at the time the VMM is written.

Guest/host masks should be used by the VMM to cause VM exits when a guest attempts to modify reserved bits. Read shadows should be used to ensure that the guest always reads the reserved value (usually 0) for such bits. The VMM response to VM exits due to attempts from a guest to modify reserved bits should be to emulate the response which the processor would have normally produced (usually a #GP(0)).

27.12 PERFORMANCE CONSIDERATIONS

VMX provides hardware features that may be used for improving processor virtualization performance. VMMs must be designed to use this support properly. The basic idea behind most of these performance optimizations of the VMM is to reduce the number of VM exits while executing a guest VM.

This section lists ways that VMMs can take advantage of the performance enhancing features in VMX.

- **Read Access to Control Registers.** Analysis of common client workloads with common PC operating systems in a virtual machine shows a large number of VM-exits are caused by control register read accesses (particularly CR0). Reads of CR0 and CR4 does not cause VM exits. Instead, they return values from the CR0/CR4 read-shadows configured by the VMM in the guest controlling-VMCS with the guest-expected values.
- **Write Access to Control Registers.** Most VMM designs require only certain bits of the control registers to be protected from direct guest access. Write access to CR0/CR4 registers can be reduced by defining the host-owned and guest-owned bits in them through the CR0/CR4 host/guest masks in the VMCS. CR0/CR4 write values by the guest are qualified with the mask bits. If they change only guest-owned bits, they are allowed without causing VM exits. Any write that cause changes to host-owned bits cause VM exits and need to be handled by the VMM.
- **Access Rights based Page Table protection.** For VMM that implement access-rights-based page table protection, the VMCS provides a CR3 target value list that can be consulted by the processor to determine if a VM exit is required. Loading of CR3 with a value matching an entry in the CR3 target-list are allowed to proceed without VM exits. The VMM can utilize the CR3 target-list to save page-table hierarchies whose state is previously verified by the VMM.
- **Page-fault handling.** Another common cause for a VM exit is due to page-faults induced by guest address remapping done through virtual memory virtualization. VMX provides page-fault error-code mask and match fields in the VMCS to filter VM exits due to page-faults based on their cause (reflected in the error-code).

CHAPTER 28
VIRTUALIZATION OF SYSTEM RESOURCES

28.1 OVERVIEW

When a VMM is hosting multiple guest environments (VMs), it must monitor potential interactions between software components using the same system resources. These interactions can require the virtualization of resources. This chapter describes the virtualization of system resources. These include: debugging facilities, address translation, physical memory, and microcode update facilities.

28.2 VIRTUALIZATION SUPPORT FOR DEBUGGING FACILITIES

The Intel 64 and IA-32 debugging facilities (see Chapter 18) provide breakpoint instructions, exception conditions, register flags, debug registers, control registers and storage buffers for functions related to debugging system and application software. In VMX operation, a VMM can support debugging system and application software from within virtual machines if the VMM properly virtualizes debugging facilities. The following list describes features relevant to virtualizing these facilities.

- The VMM can program the exception-bitmap (see Section 20.6.3) to ensure it gets control on debug functions (like breakpoint exceptions occurring while executing guest code such as INT3 instructions). Normally, debug exceptions modify debug registers (such as DR6, DR7, IA32_DEBUGCTL). However, if debug exceptions cause VM exits, exiting occurs before register modification.
- The VMM may utilize the VM-entry event injection facilities described in Section 22.5 to inject debug or breakpoint exceptions to the guest. See Section 28.2.1 for a more detailed discussion.
- The MOV-DR exiting control bit in the processor-based VM-execution control field (see Section 20.6.2) can be enabled by the VMM to cause VM exits on explicit guest access of various processor debug registers (for example, MOV to/from DR0-DR7). These exits would always occur on guest access of DR0-DR7 registers regardless of the values in CPL, DR4.DE or DR7.GD. Since all guest task switches cause VM exits, a VMM can control any indirect guest access or modification of debug registers during guest task switches.
- Guest software access to debug-related model-specific registers (such as IA32_DEBUGCTL MSR) can be trapped by the VMM through MSR access control features (such as the MSR-bitmaps that are part of processor-based VM-execution controls). See Section 27.10 for details on MSR virtualization.

- Debug registers such as DR7 and the IA32_DEBUGCTL MSR may be explicitly modified by the guest (through MOV-DR or WRMSR instructions) or modified implicitly by the processor as part of generating debug exceptions. The current values of DR7 and the IA32_DEBUGCTL MSR are saved to guest-state area of VMCS on every VM exit. Pending debug exceptions are debug exceptions that are recognized by the processor but not yet delivered. See Section 22.6.3 for details on pending debug exceptions.
- DR7 and the IA32-DEBUGCTL MSR are loaded from values in the guest-state area of the VMCS on every VM entry. This allows the VMM to properly virtualize debug registers when injecting debug exceptions to guest. Similarly, the RFLAGS[1] register is loaded on every VM entry (or pushed to stack if injecting a virtual event) from guest-state area of the VMCS. Pending debug exceptions are also loaded from guest-state area of VMCS so that they may be delivered after VM entry is completed.

28.2.1 Debug Exceptions

If a VMM emulates a guest instruction that would encounter a debug trap (single step or data or I/O breakpoint), it should cause that trap to be delivered. The VMM should not inject the debug exception using VM-entry event injection, but should set the appropriate bits in the pending debug exceptions field. This method will give the trap the right priority with respect to other events. (If the exception bitmap was programmed to cause VM exits on debug exceptions, the debug trap will cause a VM exit. At this point, the trap can be injected during VM entry with the proper priority.)

There is a valid pending debug exception if the BS bit (see Table 20-4) is set, regardless of the values of RFLAGS.TF or IA32_DEBUGCTL.BTF. The values of these bits do not impact the delivery of pending debug exceptions.

VMMs should exercise care when emulating a guest write (attempted using WRMSR) to IA32_DEBUGCTL to modify BTF if this is occurring with RFLAGS.TF = 1 and after a MOV SS or POP SS instruction (for example: while debug exceptions are blocked). Note the following:

- Normally, if WRMSR clears BTF while RFLAGS.TF = 1 and with debug exceptions blocked, a single-step trap will occur after WRMSR. A VMM emulating such an instruction should set the BS bit (see Table 20-4) in the pending debug exceptions field before VM entry.
- Normally, if WRMSR sets BTF while RFLAGS.TF = 1 and with debug exceptions blocked, neither a single-step trap nor a taken-branch trap can occur after WRMSR. A VMM emulating such an instruction should clear the BS bit (see Table 20-4) in the pending debug exceptions field before VM entry.

[1] This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.).

28.3 MEMORY VIRTUALIZATION

VMMs must control physical memory to ensure VM isolation and to remap guest physical addresses in host physical address space for virtualization. Memory virtualization allows the VMM to enforce control of physical memory and yet support guest OSs' expectation to manage memory address translation.

28.3.1 Processor Operating Modes & Memory Virtualization

Memory virtualization is required to support guest execution in various processor operating modes. This includes: protected mode with paging, protected mode with no paging, real-mode and any other transient execution modes. VMX allows guest operation in protected-mode with paging enabled and in virtual-8086 mode (with paging enabled) to support guest real-mode execution. Guest execution in transient operating modes (such as in real mode with one or more segment limits greater than 64-KByte) must be emulated by the VMM.

Since VMX operation requires processor execution in protected mode with paging (through CR0 and CR4 fixed bits), the VMM may utilize paging structures to support memory virtualization. To support guest real-mode execution, the VMM may establish a simple flat page table for guest linear to host physical address mapping. Memory virtualization algorithms may also need to capture other guest operating conditions (such as guest performing A20M# address masking) to map the resulting 20-bit effective guest physical addresses.

28.3.2 Guest & Host Physical Address Spaces

Memory virtualization provides guest software with contiguous guest physical address space starting zero and extending to the maximum address supported by the guest virtual processor's physical address width. The VMM utilizes guest physical to host physical address mapping to locate all or portions of the guest physical address space in host memory. The VMM is responsible for the policies and algorithms for this mapping which may take into account the host system physical memory map and the virtualized physical memory map exposed to a guest by the VMM. The memory virtualization algorithm needs to accommodate various guest memory uses (such as: accessing DRAM, accessing memory-mapped registers of virtual devices or core logic functions and so forth). For example:

- To support guest DRAM access, the VMM needs to map DRAM-backed guest physical addresses to host-DRAM regions. The VMM also requires the guest to host memory mapping to be at page granularity.
- Virtual devices (I/O devices or platform core logic) emulated by the VMM may claim specific regions in the guest physical address space to locate memory-mapped registers. Guest access to these virtual registers may be configured to cause page-fault induced VM-exits by marking these regions as always not

present. The VMM may handle these VM exits by invoking appropriate virtual device emulation code.

28.3.3 Virtualizing Virtual Memory by Brute Force

VMX provides the hardware features required to fully virtualize guest virtual memory accesses. VMX allows the VMM to trap guest accesses to the PAT (Page Attribute Table) MSR and the MTRR (Memory Type Range Registers). This control allows the VMM to virtualize the specific memory type of a guest memory. The VMM may control caching by controlling the guest CR0.CRD and CR0.NW bits, as well as by trapping guest execution of the INVD instruction. The VMM can trap guest CR3 loads and stores, and it may trap guest execution of INVLPG.

Because a VMM must retain control of physical memory, it must also retain control over the processor's address-translation mechanisms. Specifically, this means that only the VMM can access CR3 (which contains the base of the page directory) and can execute INVLPG (the only other instruction that directly manipulates the TLB).

At the same time that the VMM controls address translation, a guest operating system will also expect to perform normal memory management functions. It will access CR3, execute INVLPG, and modify (what it believes to be) page directories and page tables. Virtualization of address translation must tolerate and support guest attempts to control address translation.

A simple-minded way to do this would be to ensure that all guest attempts to access address-translation hardware trap to the VMM where such operations can be properly emulated. It must ensure that accesses to page directories and page tables also get trapped. This may be done by protecting these in-memory structures with conventional page-based protection. The VMM can do this because it can locate the page directory because its base address is in CR3 and the VMM receives control on any change to CR3; it can locate the page tables because their base addresses are in the page directory.

Such a straightforward approach is not necessarily desirable. Protection of the in-memory translation structures may be cumbersome. The VMM may maintain these structures with different values (e.g., different page base addresses) than guest software. This means that there must be traps on guest attempt to read these structures and that the VMM must maintain, in auxiliary data structures, the values to return to these reads. There must also be traps on modifications to these structures even if the translations they effect are never used. All this implies considerable overhead that should be avoided.

28.3.4 Alternate Approach to Memory Virtualization

Guest software is allowed to freely modify the guest page-table hierarchy without causing traps to the VMM. Because of this, the active page-table hierarchy might not always be consistent with the guest hierarchy. Any potential problems arising from inconsistencies can be solved using techniques analogous to those used by the processor and its TLB.

This section describes an alternative approach that allows guest software to freely access page directories and page tables. Traps occur on CR3 accesses and executions of INVLPG. They also occur when necessary to ensure that guest modifications to the translation structures actually take effect. The software mechanisms to support this approach are collectively called virtual TLB. This is because they emulate the functionality of the processor's physical translation look-aside buffer (TLB).

The basic idea behind the virtual TLB is similar to that behind the processor TLB. While the page-table hierarchy defines the relationship between physical to linear address, it does not directly control the address translation of each memory access. Instead, translation is controlled by the TLB, which is occasionally filled by the processor with translations derived from the page-table hierarchy. With a virtual TLB, the page-table hierarchy established by guest software (specifically, the guest operating system) does not control translation, either directly or indirectly. Instead, translation is controlled by the processor (through its TLB) and by the VMM (through a page-table hierarchy that it maintains).

Specifically, the VMM maintains an alternative page-table hierarchy that effectively caches translations derived from the hierarchy maintained by guest software. The remainder of this document refers to the former as the active page-table hierarchy (because it is referenced by CR3 and may be used by the processor to load its TLB) and the latter as the guest page-table hierarchy (because it is maintained by guest software). The entries in the active hierarchy may resemble the corresponding entries in the guest hierarchy in some ways and may differ in others.

Guest software is allowed to freely modify the guest page-table hierarchy without causing VM exits to the VMM. Because of this, the active page-table hierarchy might not always be consistent with the guest hierarchy. Any potential problems arising from any inconsistencies can be solved using techniques analogous to those used by the processor and its TLB. Note the following:

- Suppose the guest page-table hierarchy allows more access than active hierarchy (for example: there is a translation for a linear address in the guest hierarchy but not in the active hierarchy); this is analogous to a situation in which the TLB allows less access than the page-table hierarchy. If an access occurs that would be allowed by the guest hierarchy but not the active one, a page fault occurs; this is analogous to a TLB miss. The VMM gains control (as it handles all page faults) and can update the active page-table hierarchy appropriately; this corresponds to a TLB fill.
- Suppose the guest page-table hierarchy allows less access than the active hierarchy; this is analogous to a situation in which the TLB allows more access than the page-table hierarchy. This situation can occur only if the guest operating system has modified a page-table entry to reduce access (for example: by marking it not-present). Because the older, more permissive translation may have been cached in the TLB, the processor is architecturally permitted to use the older translation and allow more access. Thus, the VMM may (through the active page-table hierarchy) also allow greater access. For the new, less permissive

translation to take effect, guest software should flush any older translations from the TLB either by executing INVLPG or by loading CR3. Because both these operations will cause a trap to the VMM, the VMM will gain control and can remove from the active page-table hierarchy the translations indicated by guest software (the translation of a specific linear address for INVLPG or all translations for a load of CR3).

As noted previously, the processor reads the page-table hierarchy to cache translations in the TLB. It also writes to the hierarchy to main the accessed (A) and dirty (D) bits in the PDEs and PTEs. The virtual TLB emulates this behavior as follows:

- When a page is accessed by guest software, the A bit in the corresponding PTE (or PDE for a 4-MByte page) in the active page-table hierarchy will be set by the processor (the same is true for PDEs when active page tables are accessed by the processor). For guest software to operate properly, the VMM should update the A bit in the guest entry at this time. It can do this reliably if it keeps the active PTE (or PDE) marked not-present until it has set the A bit in the guest entry.
- When a page is written by guest software, the D bit in the corresponding PTE (or PDE for a 4-MByte page) in the active page-table hierarchy will be set by the processor. For guest software to operate properly, the VMM should update the D bit in the guest entry at this time. It can do this reliably if it keeps the active PTE (or PDE) marked read-only until it has set the D bit in the guest entry. This solution is valid for guest software running at privilege level 3; support for more privileged guest software is described in Section 28.3.5.

28.3.5 Details of Virtual TLB Operation

This section describes in more detail how a VMM could support a virtual TLB. It explains how an active page-table hierarchy is initialized and how it is maintained in response to page faults, uses of INVLPG, and accesses to CR3. The mechanisms described here are the minimum necessary. They may not result in the best performance.

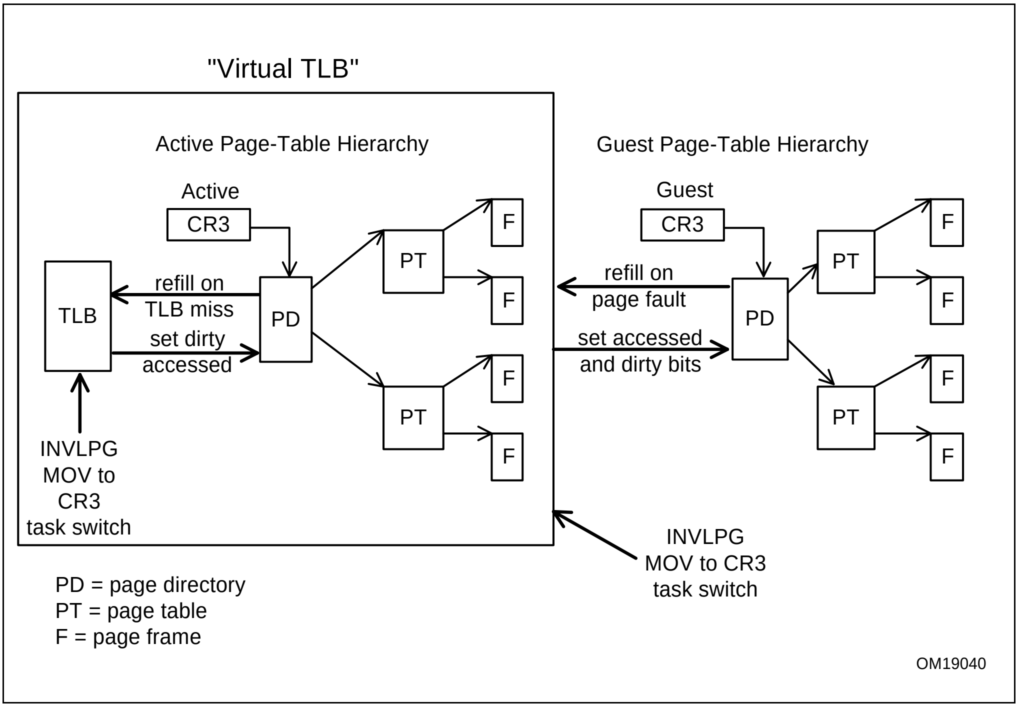

Figure 28-1. Virtual TLB Scheme

As noted above, the VMM maintains an active page-table hierarchy for each virtual machine that it supports. It also maintains, for each machine, values that the machine expects for control registers CR0, CR2, CR3, and CR4 (they control address translation). These values are called the guest control registers.

In general, the VMM selects the physical-address space that is allocated to guest software. The term guest address refers to an address installed by guest software in the guest CR3, in a guest PDE (as a page table base address or a page base address), or in a guest PTE (as a page base address). While guest software considers these to be specific physical addresses, the VMM may map them differently.

28.3.5.1 Initialization of Virtual TLB

To enable the Virtual TLB scheme, the VMCS must be set up to trigger VM exits on:

- All writes to CR3 (the CR3-target count should be 0) or the paging-mode bits in CR0 and CR4 (using the CR0 and CR4 guest/host masks)
- Page-fault (#PF) exceptions
- Execution of INVLPG

When guest software first enables paging, the VMM creates an aligned 4-KByte active page directory that is invalid (all entries marked not-present). This invalid directory is analogous to an empty TLB.

28.3.5.2 Response to Page Faults

Page faults can occur for a variety of reasons. In some cases, the page fault alerts the VMM to an inconsistency between the active and guest page-table hierarchy. In such cases, the VMM can update the former and re-execute the faulting instruction. In other cases, the hierarchies are already consistent and the fault should be handled by the guest operating system. The VMM can detect this and use an established mechanism for raising a page fault to guest software.

The VMM can handle a page fault by following these steps (The steps below assume the guest is operating in a paging mode without PAE. Analogous steps to handle address translation using PAE or four-level paging mechanisms can be derived by VMM developers according to the paging behavior defined in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*):

1. First consult the active PDE, which can be located using the upper 10 bits of the faulting address and the current value of CR3. The active PDE is the source of the fault if it is marked not present or if its R/W bit and U/S bits are inconsistent with the attempted guest access (the guest privilege level and the value of CR0:WP should also be taken into account).
2. If the active PDE is the source of the fault, consult the corresponding guest PDE using the same 10 bits from the faulting address and the physical address that corresponds to the guest address in the guest CR3. If the guest PDE would cause a page fault (for example: it is marked not present), then raise a page fault to the guest operating system.

   The following steps assume that the guest PDE would not have caused a page fault.
3. If the active PDE is the source of the fault and the guest PDE contains, as page-table base address (if PS = 0) or page base address (PS = 1), a guest address that the VMM has chosen not to support; then raise a machine check (or some other abort) to the guest operating system.

   The following steps assume that the guest address in the guest PDE is supported for the virtual machine.
4. If the active PDE is marked not-present, then set the active PDE to correspond to guest PDE as follows:
   a. If the active PDE contains a page-table base address (if PS = 0), then allocate an aligned 4-KByte active page table marked completely invalid and set the page-table base address in the active PDE to be the physical address of the newly allocated page table.

b. If the active PDE contains a page base address (if PS = 1), then set the page base address in the active PDE to be the physical page base address that corresponds to the guest address in the guest PDE.

c. Set the P, U/S, and PS bits in the active PDE to be identical to those in the guest PDE.

d. Set the PWT, PCD, and G bits according to the policy of the VMM.

e. Set A = 1 in the guest PDE.

f. If D = 1 in the guest PDE or PS = 0 (meaning that this PDE refers to a page table), then set the R/W bit in the active PDE as in the guest PDE.

g. If D = 0 in the guest PDE, PS = 1 (this is a 4-MByte page), and the attempted access is a write; then set R/W in the active PDE as in the guest PDE and set D = 1 in the guest PDE.

h. If D = 0 in the guest PDE, PS = 1, and the attempted access is not a write; then set R/W = 0 in the active PDE.

i. After modifying the active PDE, re-execute the faulting instruction.

The remaining steps assume that the active PDE is already marked present.

5. If the active PDE is the source of the fault, the active PDE refers to a 4-MByte page (PS = 1), the attempted access is a write; D = 0 in the guest PDE, and the active PDE has caused a fault solely because it has R/W = 0; then set R/W in the active PDE as in the guest PDE; set D = 1 in the guest PDE, and re-execute the faulting instruction.

6. If the active PDE is the source of the fault and none of the above cases apply, then raise a page fault of the guest operating system.

The remaining steps assume that the source of the original page fault is not the active PDE.

NOTE

It is possible that the active PDE might be causing a fault even though the guest PDE would not. However, this can happen only if the guest operating system increased access in the guest PDE and did not take action to ensure that older translations were flushed from the TLB. Such translations might have caused a page fault if the guest software were running on bare hardware.

7. If the active PDE refers to a 4-MByte page (PS = 1) but is not the source of the fault, then the fault resulted from an inconsistency between the active page-table hierarchy and the processor’s TLB. Since the transition to the VMM caused an address-space change and flushed the processor’s TLB, the VMM can simply re-execute the faulting instruction.

The remaining steps assume that PS = 0 in the active and guest PDEs.

8. Consult the active PTE, which can be located using the next 10 bits of the faulting address (bits 21–12) and the physical page-table base address in the active PDE. The active PTE is the source of the fault if it is marked not-present or if its R/W bit and U/S bits are inconsistent with the attempted guest access (the guest privilege level and the value of CR0:WP should also be taken into account).

9. If the active PTE is not the source of the fault, then the fault has resulted from an inconsistency between the active page-table hierarchy and the processor’s TLB. Since the transition to the VMM caused an address-space change and flushed the processor’s TLB, the VMM simply re-executes the faulting instruction.

   The remaining steps assume that the active PTE is the source of the fault.

10. Consult the corresponding guest PTE using the same 10 bits from the faulting address and the physical address that correspond to the guest page-table base address in the guest PDE. If the guest PTE would cause a page fault (it is marked not-present), the raise a page fault to the guest operating system.

    The following steps assume that the guest PTE would not have caused a page fault.

11. If the guest PTE contains, as page base address, a physical address that is not valid for the virtual machine being supported; then raise a machine check (or some other abort) to the guest operating system.

    The following steps assume that the address in the guest PTE is valid for the virtual machine.

12. If the active PTE is marked not-present, then set the active PTE to correspond to guest PTE:

    a. Set the page base address in the active PTE to be the physical address that corresponds to the guest page base address in the guest PTE.

    b. Set the P, U/S, and PS bits in the active PTE to be identical to those in the guest PTE.

    c. Set the PWT, PCD, and G bits according to the policy of the VMM.

    d. Set A = 1 in the guest PTE.

    e. If D = 1 in the guest PTE, then set the R/W bit in the active PTE as in the guest PTE.

    f. If D = 0 in the guest PTE and the attempted access is a write, then set R/W in the active PTE as in the guest PTE and set D = 1 in the guest PTE.

    g. If D = 0 in the guest PTE and the attempted access is not a write, then set R/W = 0 in the active PTE.

    h. After modifying the active PTE, re-execute the faulting instruction.

    The remaining steps assume that the active PTE is already marked present.

13. If the attempted access is a write, D = 0 (not dirty) in the guest PTE and the active PTE has caused a fault solely because it has R/W = 0 (read-only); then set

R/W in the active PTE as in the guest PTE, set D = 1 in the guest PTE and re-execute the faulting instruction.

14. If none of the above cases apply, then raise a page fault of the guest operating system.

28.3.5.3 Response to Uses of INVLPG

Operating-systems can use INVLPG to flush entries from the TLB. This instruction takes a linear address as an operand and software expects any cached translations for the address to be flushed. VMM should set the processor-based VMCS execution control invplg-exiting = 1, such that any attempts by a privileged guest to execute INVLPG will trap to the VMM (attempts to execute INVLPG by unprivileged guest are managed by the exception bitmap control in the VMCS). The VMM can then modify the active page-table hierarchy to emulate the desired effect of the INVLPG.

The following steps are performed. Note that these steps are performed only if the guest invocation of INVLPG would not fault and only if the guest software is running at privilege level 0:

1. Locate the relevant active PDE using the upper 10 bits of the operand address and the current value of CR3. If the PDE refers to a 4-MByte page (PS = 1), then set P = 0 in the PDE.
2. If the PDE is marked present and refers to a page table (PS = 0), locate the relevant active PTE using the next 10 bits of the operand address (bits 21–12) and the page-table base address in the PDE. Set P = 0 in the PTE. Examine all PTEs in the page table; if they are now all marked not-present, de-allocate the page table and set P = 0 in the PDE (this step may be optional).

28.3.5.4 Response to CR3 Writes

A guest operating system may attempt to write to CR3. Any write to CR3 implies a TLB flush and a possible page table change. The following steps are performed:

1. The VMM notes the new CR3 value (used later to walk guest page tables) and emulates the write.
2. The VMM allocates a new PD page, with all invalid entries.
3. The VMM sets actual processor CR3 register to point to the new PD page.

The VMM may, at this point, speculatively fill in VTLB mappings for performance reasons.

28.4 MICROCODE UPDATE FACILITY

The microcode code update facility may be invoked at various points during the operation of a platform. Typically, the BIOS invokes the facility on all processors during the BIOS boot process. This is sufficient to boot the BIOS and operating system. As a microcode update more current than the system BIOS may be available, system software should provide another mechanism for invoking the microcode update facility. The implications of the microcode update mechanism on the design of the VMM are described in this section.

NOTE

Microcode updates must not be performed during VMX non-root operation. Updates performed in VMX non-root operation may result in unpredictable system behavior.

28.4.1 Early Load of Microcode Updates

The microcode update facility may be invoked early in the VMM or guest OS boot process. Loading the microcode update early provides the opportunity to correct errata affecting the boot process but the technique generally requires a reboot of the software.

A microcode update may be loaded from the OS or VMM image loader. Typically, such image loaders do not run on every logical processor, so this method effects only one logical processor. Later in the VMM or OS boot process, after bringing all application processors on-line, the VMM or OS needs to invoke the microcode update facility for all application processors.

Depending on the order of the VMM and the guest OS boot, the microcode update facility may be invoked by the VMM or the guest OS. For example, if the guest OS boots first and then loads the VMM, the guest OS may invoke the microcode update facility on all the logical processors. If a VMM boots before its guests, then the VMM may invoke the microcode update facility during its boot process. In both cases, the VMM or OS should invoke the microcode update facilities soon after performing the multiprocessor startup.

In the early load scenario, microcode updates may be contained in the VMM or OS image or, the VMM or OS may manage a separate database or file of microcode updates. Maintaining a separate microcode update image database has the advantage of reducing the number of required VMM or OS releases as a result of microcode update releases.

28.4.2 Late Load of Microcode Updates

A microcode update may be loaded during normal system operation. This allows system software to activate the microcode update at anytime without requiring a system reboot. This scenario does not allow the microcode update to correct errata which affect the processor’s boot process but does allow high-availability systems to activate microcode updates without interrupting the availability of the system. In this late load scenario, either the VMM or a designated guest may load the microcode update. If the guest is loading the microcode update, the VMM must make sure that the entire guest memory buffer (which contains the microcode update image) will not cause a page fault when accessed.

If the VMM loads the microcode update, then the VMM must have access to the current set of microcode updates. These updates could be part of the VMM image or could be contained in a separate microcode update image database (for example: a database file on disk or in memory). Again, maintaining a separate microcode update image database has the advantage of reducing the number of required VMM or OS releases as a result of microcode update releases.

The VMM may wish to prevent a guest from loading a microcode update or may wish to support the microcode update requested by a guest using emulation (without actually loading the microcode update). To prevent microcode update loading, the VMM may return a microcode update signature value greater than the value of IA32_BISO_SIGN_ID MSR. A well behaved guest will not attempt to load an older microcode update. The VMM may also drop the guest attempts to write to IA32_BIOS_UPDT_TRIG MSR, preventing the guest from loading any microcode updates. Later, when the guest queries IA32_BIOS_SIGN_ID MSR, the VMM could emulate the microcode update signature that the guest expects.

In general, loading a microcode update later will limit guest software’s visibility of features that may be enhanced by a microcode update.

CHAPTER 29
HANDLING BOUNDARY CONDITIONS IN A VIRTUAL MACHINE MONITOR

29.1 OVERVIEW

This chapter describes what a VMM must consider when handling exceptions, interrupts, error conditions, and transitions between activity states.

29.2 INTERRUPT HANDLING IN VMX OPERATION

The following bullets summarize VMX support for handling interrupts:

- **Control of Processor Exceptions**. The VMM can get control on specific guest exceptions through the exception-bitmap in the guest controlling-VMCS. The exception bitmap is a 32-bit field that allows the VMM to specify processor behavior on specific exceptions (including traps, faults and aborts). Setting a specific bit in the exception bitmap implies VM exits will be generated when the corresponding exception occurs. Any exceptions that are programmed not to cause VM exits are delivered directly to the guest through the guest IDT. The exception bitmap also controls execution of relevant instructions such as BOUND, INTO and INT3. VM exits on page-faults are treated in such a way the page-fault error-code is qualified through the page fault error-code mask and match fields in the VMCS.
- **Control over Triple-faults**. If a fault occurs while attempting to call a double-fault handler in the guest and that fault is not configured to cause a VM exit in the exception bitmap, the resulting triple fault causes a VM exit.
- **Control of External-Interrupts**. VMX allows both host and guest control of external interrupts through the “external-interrupt exiting” VM execution control. With guest control (external-interrupt exiting set to 0), external-interrupts do not cause VM exits and the interrupt delivery is masked by the guest programmed RFLAGS.IF value.[1] With host control (external-interrupt exiting set to 1), external-interrupts causes VM exits and are not masked by RFLAGS.IF. The VMM can identify VM exits due to external interrupts by checking the exit-reason for an ‘external-interrupt’ (value = 1).

[1] This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.).

- **Control of Other Events**. There is a pin-based VM-execution control that controls system behavior (exit or no-exit) for NMI events. Most VMM usages will need handling of NMI external events in the VMM and hence will specify host control of these events.

  Some processors also support a pin-based VM-execution control called “virtual NMIs.” When this control is set, NMIs cause VM exits, but the processor tracks guest readiness for virtual NMIs. This control interacts with the “NMI-window exiting” VM-execution control (see below).

  INIT and SIPI events always cause VM exits.
- **Acknowledge-Interrupt-On-Exit**. The acknowledge-interrupt-on-exit bit in the VM-exit control field in the controlling-VMCS controls processor behavior for external interrupt acknowledgement. If the control bit is set, the processor acknowledges the interrupt controller to acquire the interrupt vector upon VM exit, and stores the vector in the VM-exit interruption-information field. If the control bit is clear, the external interrupt is not acknowledged during VM exit. Since RFLAGS.IF is automatically cleared on VM exits due to external interrupts, VMM re-enabling of interrupts (setting RFLAGS.IF = 1) initiates the external interrupt acknowledgement and vectoring of the external interrupt through the monitor/host IDT.
- **Event Masking Support**. VMX captures the masking conditions of specific events while in VMX non-root operation through the interruptibility-state field in the guest-state area of the VMCS.

  This feature allows proper virtualization of various interrupt blocking states, such as: (a) blocking of external interrupts for the instruction following STI; (b) blocking of interrupts for the instruction following a MOV-SS or POP-SS instruction; (c) SMI blocking of subsequent SMIs until the next execution of RSM; and (d) NMI/SMI blocking of NMIs until the next execution of IRET or RSM.

  INIT and SIPI events are treated specially. INIT assertions are always blocked in VMX root operation and while in SMM, and unblocked otherwise. SIPI events are always blocked in VMX root operation.

  The interruptibility state is loaded from the VMCS guest-state area on every VM entry and saved into the VMCS on every VM exit.
- **Event injection**. VMX operation allows injecting interruptions to a guest virtual machine through the use of VM-entry interrupt-information field in VMCS. Injectable interruptions include external interrupts, NMI, processor exceptions, software generated interrupts, and software traps. If the interrupt-information field indicates a valid interrupt, exception or trap event upon the next VM entry; the processor will use the information in the field to vector a virtual interruption through the guest IDT after all guest state and MSRs are loaded. Delivery through the guest IDT emulates vectoring in non-VMX operation by doing the normal privilege checks and pushing appropriate entries to the guest stack (entries may include RFLAGS, EIP and exception error code). A VMM with host control of NMI and external interrupts can use the event-injection facility to forward virtual interruptions to various guest virtual machines.

- **Interrupt-window Exiting**. The interrupt-window exiting control bit in the VM-execution controls (Section 20.6.2) causes VM exits when guest RFLAGS.IF is 1 and no other conditions block external interrupts. If the control is 1, a VM exit occurs at the beginning of any instruction at which RFLAGS.IF = 1 and on which the interruptibility state of the guest would allow delivery of an interrupt. For example: when the guest executes an STI instruction, RFLAGS = 1, and if at the completion of next instruction the interruptibility state masking due to STI is removed; a VM exit occurs if interrupt-window exiting control is 1. The interrupt-window exiting feature allows a VMM to queue a virtual interrupt to the guest when the guest is not in an interruptible state. The VMM can set the interrupt-window exiting control for the guest and depend on a VM exit to know when the guest becomes interruptible (and, therefore, when it can inject a virtual interrupt). The VMM can detect such VM exits by checking for the basic exit reason 'interrupt-window' (value = 7). Without interrupt-window exiting support, the VMM will need to poll and check the interruptibility state of the guest to deliver virtual interrupts.
- **NMI-window Exiting**. If the "virtual NMIs" VM-execution is set, the processor tracks virtual-NMI blocking. The NMI-window exiting control bit in VM-execution controls (Section 20.6.2) causes VM exits when there is no virtual-NMI blocking. For example, after execution of the IRET instruction, a VM exit occurs if NMI-window exiting control is 1. The NMI-window exiting feature allows a VMM to queue a virtual NMI to a guest when the guest is not ready to receive NMIs. The VMM can set the NMI-window exiting control for the guest and depend on a VM exit to know when the guest becomes ready for NMIs (and, therefore, when it can inject a virtual NMI). The VMM can detect such VM exits by checking for the basic exit reason 'NMI window' (value = 8). Without NMI-window exiting support, the VMM will need to poll and check the interruptibility state of the guest to deliver virtual NMIs.
- **VM-Exit Information**. The VM-exit information fields provide details on VM exits due to exceptions and interrupts. This information is provided through the exit-qualification, VM-exit-interruption-information, instruction-length and interruption-error-code fields. Also, for VM exits that occur in the course of vectoring through the guest-IDT, information about the event that was being vectored through the guest-IDT is provided in the IDT-vectoring-information and IDT-vectoring-error-code fields. These information fields allow the VMM to identify the exception cause and to handle it properly.

29.3 EXTERNAL INTERRUPT VIRTUALIZATION

VMX operation allows both host and guest control of external interrupts. While guest control of external interrupts might be suitable for partitioned usages (different CPU cores/threads and I/O devices partitioned to independent virtual machines), most VMMs built upon VMX are expected to utilize host control of external interrupts. The rest of this section describes a general host-controlled interrupt virtualization architecture for standard PC platforms through the use of VMX supported features.

With host control of external interrupts, the VMM (or the host OS in a hosted VMM model) manages the physical interrupt controllers in the platform and the interrupts generated through them. The VMM exposes software-emulated virtual interrupt controller devices (such as PIC and APIC) to each guest virtual machine instance.

29.3.1 Virtualization of Interrupt Vector Space

The Intel 64 and IA-32 architectures use 8-bit vectors of which 244 (20H - FFH) are available for external interrupts. Vectors are used to select the appropriate entry in the interrupt descriptor table (IDT). VMX operation allows each guest to control its own IDT. Host vectors refer to vectors delivered by the platform to the processor during the interrupt acknowledgement cycle. Guest vectors refer to vectors programmed by a guest to select an entry in its guest IDT. Depending on the I/O resource management models supported by the VMM design, the guest vector space may or may not overlap with the underlying host vector space.

- Interrupts from virtual devices: Guest vector numbers for virtual interrupts delivered to guests on behalf of emulated virtual devices have no direct relation to the host vector numbers of interrupts from physical devices on which they are emulated. A guest-vector assigned for a virtual device by the guest operating environment is saved by the VMM and utilized when injecting virtual interrupts on behalf of the virtual device.
- Interrupts from assigned physical devices: Hardware support for I/O device assignment allows physical I/O devices in the host platform to be assigned (direct-mapped) to VMs. Guest vectors for interrupts from direct-mapped physical devices take up equivalent space from the host vector space, and require the VMM to perform host-vector to guest-vector mapping for interrupts.

Figure 29-1 illustrates the functional relationship between host external interrupts and guest virtual external interrupts. Device A is owned by the host and generates external interrupts with host vector X. The host IDT is set up such that the interrupt service routine (ISR) for device driver A is hooked to host vector X as normal. VMM emulates (over device A) virtual device C in software which generates virtual interrupts to the VM with guest expected vector P. Device B is assigned to a VM and generates external interrupts with host vector Y. The host IDT is programmed to hook the VMM interrupt service routine (ISR) for assigned devices for vector Y, and the VMM handler injects virtual interrupt with guest vector Q to the VM. The guest operating system programs the guest to hook appropriate guest driver’s ISR to vectors P and Q.

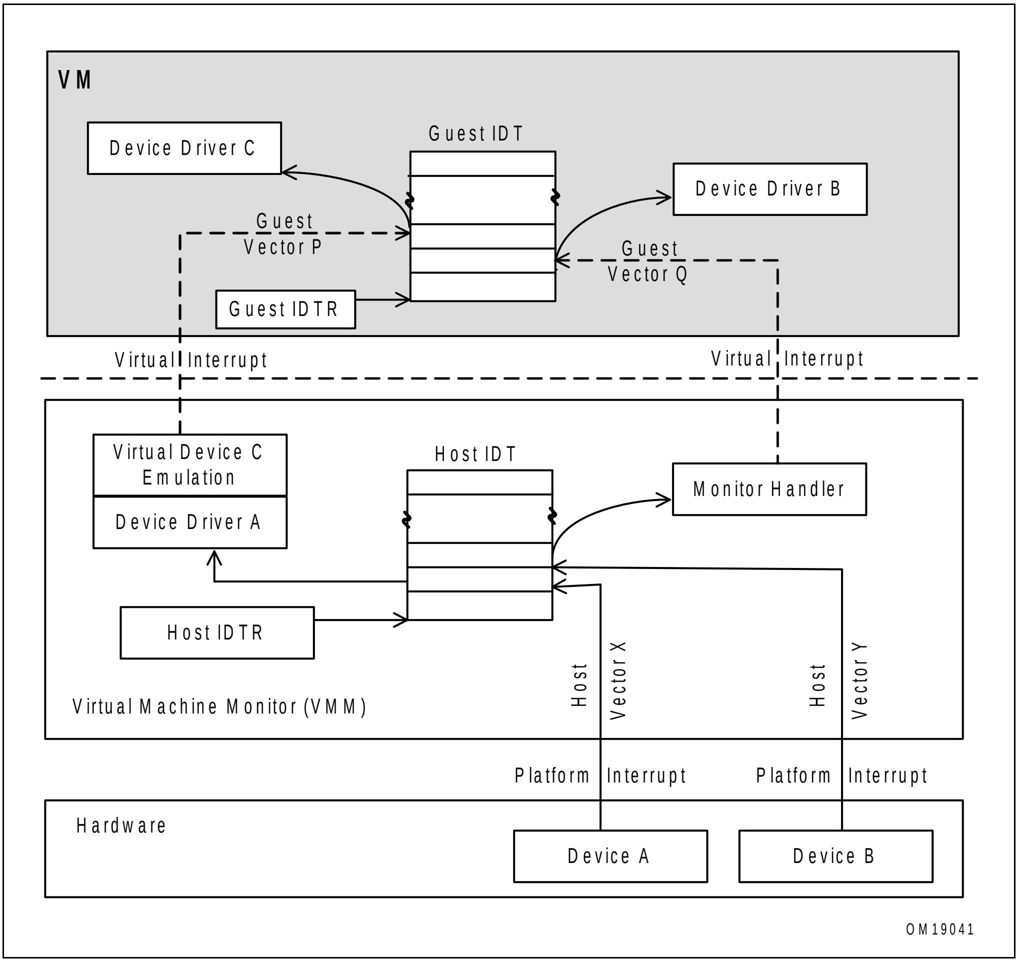

Figure 29-1. Host External Interrupts and Guest Virtual Interrupts

29.3.2 Control of Platform Interrupts

To meet the interrupt virtualization requirements, the VMM needs to take ownership of the physical interrupts and the various interrupt controllers in the platform. VMM control of physical interrupts may be enabled through the host-control settings of the “external-interrupt exiting” VM-execution control. To take ownership of the platform interrupt controllers, the VMM needs to expose the virtual interrupt controller devices to the virtual machines and restrict guest access to the platform interrupt controllers.

Intel 64 and IA-32 platforms can support three types of external interrupt control mechanisms: Programmable Interrupt Controllers (PIC), Advanced Programmable

Interrupt Controllers (APIC), and Message Signaled Interrupts (MSI). The following sections provide information on the virtualization of each of these mechanisms.

29.3.2.1 PIC Virtualization

Typical PIC-enabled platform implementations support dual 8259 interrupt controllers cascaded as master and slave controllers. They supporting up to 15 possible interrupt inputs. The 8259 controllers are programmed through initialization command words (ICWx) and operation command words (OCWx) accessed through specific I/O ports. The various interrupt line states are captured in the PIC through interrupt requests, interrupt service routines and interrupt mask registers.

Guest access to the PIC I/O ports can be restricted by activating I/O bitmaps in the guest controlling-VMCS (activate-I/O-bitmap bit in VM-execution control field set to 1) and pointing the I/O-bitmap physical addresses to valid bitmap regions. Bits corresponding to the PIC I/O ports can be cleared to cause a VM exit on guest access to these ports.

If the VMM is not supporting direct access to any I/O ports from a guest, it can set the unconditional-I/O-exiting in the VM-execution control field instead of activating I/O bitmaps. The exit-reason field in VM-exit information allows identification of VM exits due to I/O access and can provide an exit-qualification to identify details about the guest I/O operation that caused the VM exit.

The VMM PIC virtualization needs to emulate the platform PIC functionality including interrupt priority, mask, request and service states, and specific guest programmed modes of PIC operation.

29.3.2.2 xAPIC Virtualization

Most modern Intel 64 and IA-32 platforms include support for an APIC. While the standard PIC is intended for use on uniprocessor systems, APIC can be used in either uniprocessor or multi-processor systems.

APIC based interrupt control consists of two physical components: the interrupt acceptance unit (Local APIC) which is integrated with the processor, and the interrupt delivery unit (I/O APIC) which is part of the I/O subsystem. APIC virtualization involves protecting the platform's local and I/O APICs and emulating them for the guest.

29.3.2.3 Local APIC Virtualization

The local APIC is responsible for the local interrupt sources, interrupt acceptance, dispensing interrupts to the logical processor, and generating inter-processor interrupts. Software interacts with the local APIC by reading and writing its memory-mapped registers residing within a 4-KByte uncached memory region with base address stored in the IA32_APIC_BASE MSR. Since the local APIC registers are memory-mapped, the VMM can utilize memory virtualization techniques (such as page-table virtualization) to trap guest accesses to the page frame hosting the virtual local APIC registers.

Local APIC virtualization in the VMM needs to emulate the various local APIC operations and registers, such as: APIC identification/format registers, the local vector table (LVT), the interrupt command register (ICR), interrupt capture registers (TMR, IRR and ISR), task and processor priority registers (TPR, PPR), the EOI register and the APIC-timer register. Since local APICs are designed to operate with non-specific EOI, local APIC emulation also needs to emulate broadcast of EOI to the guest’s virtual I/O APICs for level triggered virtual interrupts.

A local APIC allows interrupt masking at two levels: (1) mask bit in the local vector table entry for local interrupts and (2) raising processor priority through the TPR registers for masking lower priority external interrupts. The VMM needs to comprehend these virtual local APIC mask settings as programmed by the guest in addition to the guest virtual processor interruptibility state (when injecting APIC routed external virtual interrupts to a guest VM).

VMX provides several features which help the VMM to virtualize the local APIC. These features allow many of guest TPR accesses (using CR8 only) to occur without VM exits to the VMM:

- The VMCS contains a 'Virtual-APIC page address' field. This 64-bit field is the physical address of the 4-KByte virtual APIC page (4-KByte aligned). The virtual-APIC page contains a TPR shadow, which is accessed by the MOV CR8 instruction. The TPR shadow comprises bits 7:4 in byte 80H of the virtual-APIC page.
- The TPR threshold: bits 3:0 of this 32-bit field determine the threshold below which the TPR shadow cannot fall. A VM exit will occur after an execution of MOV CR8 that reduces the TPR shadow below this value.
- The processor-based VM-execution controls field contains a 'Use TPR shadow' bit and a 'CR8-store exiting' bit. If 'Use TPR shadow' is set and 'CR8-store exiting' is cleared, then a MOV from CR8 reads from the TPR shadow. If the 'CR8-store exiting' VM-execution control is set, then MOV from CR8 causes a VM exit. 'Use TPR shadow' is ignored in this case.
- The processor-based VM-execution controls field contains a 'CR8-load exiting' bit. If 'Use TPR shadow' is set and 'CR8-load exiting' is clear, then MOV to CR8 writes to the 'TPR shadow'. A VM exit will occur after this write if the value written is below the TPR threshold. If 'CR8-load exiting' is set, then MOV to CR8 causes a VM exit. 'Use TPR shadow' is ignored in this case.

29.3.2.4 I/O APIC Virtualization

The I/O APIC registers are typically mapped to a 1 MByte region where each I/O APIC is allocated a 4K address window within this range. The VMM may utilize physical memory virtualization to trap guest accesses to the virtual I/O APIC memory-mapped registers. The I/O APIC virtualization needs to emulate the various I/O APIC operations and registers such as identification/version registers, indirect-I/O-access registers, EOI register, and the I/O redirection table. I/O APIC virtualization also need to emulate various redirection table entry settings such as delivery mode, destination mode, delivery status, polarity, masking, and trigger mode programmed by the guest and track remote-IRR state on guest EOI writes to various virtual local APICs.

29.3.2.5 Virtualization of Message Signaled Interrupts

The *PCI Local Bus Specification* (Rev. 2.2) introduces the concept of message signaled interrupts (MSI). MSI enable PCI devices to request service by writing a system-specified message to a system specified address. The transaction address specifies the message destination while the transaction data specifies the interrupt vector, trigger mode and delivery mode. System software is expected to configure the message data and address during MSI device configuration, allocating one or more no-shared messages to MSI capable devices. Chapter 9, “Advanced Programmable Interrupt Controller (APIC),” specifies the MSI message address and data register formats to be followed on Intel 64 and IA-32 platforms. While MSI is optional for conventional PCI devices, it is the preferred interrupt mechanism for PCI-Express devices.

Since the MSI address and data are configured through PCI configuration space, to control these physical interrupts the VMM needs to assume ownership of PCI configuration space. This allows the VMM to capture the guest configuration of message address and data for MSI-capable virtual and assigned guest devices. PCI configuration transactions on PC-compatible systems are generated by software through two different methods:

1. The standard CONFIG_ADDRESS/CONFIG_DATA register mechanism (CFCH/CF8H ports) as defined in the *PCI Local Bus Specification.*
2. The enhanced flat memory-mapped (MEMCFG) configuration mechanism as defined in the *PCI-Express Base Specification* (Rev. 1.0a.).

The CFCH/CF8H configuration access from guests can be trapped by the VMM through use of I/O-bitmap VM-execution controls. The memory-mapped PCI-Express MEMCFG guest configuration accesses can be trapped by VMM through physical memory virtualization.

29.3.3 Examples of Handling of External Interrupts

The following sections illustrate interrupt processing in a VMM (when used to support the external interrupt virtualization requirements).

29.3.3.1 Guest Setup

The VMM sets up the guest to cause a VM exit to the VMM on external interrupts. This is done by setting the “external-interrupt exiting” VM-execution control in the guest controlling-VMCS.

29.3.3.2 Processor Treatment of External Interrupt

Interrupts are automatically masked by hardware in the processor on VM exit by clearing RFLAGS.IF. The exit-reason field in VMCS is set to 1 to indicate an external interrupt as the exit reason.

If the VMM is utilizing the acknowledge-on-exit feature (by setting the acknowledge-interrupt-on-exit bit in guest VM-exit control field), the processor acknowledges the interrupt, retrieves the host vector, and saves the interrupt in the exit-interruption-information field (in the VM-exit information region of the VMCS) before transitioning control to the VMM.

29.3.3.3 Processing of External Interrupts by VMM

Upon VM exit, the VMM can determine the exit cause of an external interrupt by checking the exit-reason field (value = 1) in VMCS. If the acknowledge-interrupt-on-exit control (see Section 20.7.1) is enabled, the VMM can use the saved host vector (in the exit-interruption-information field) to switch to the appropriate interrupt handler. If acknowledge-interrupt-on-exit is not enabled, the VMM may re-enable interrupts (by setting RFLAGS.IF) to allow vectoring of external interrupts through the monitor/host IDT.

The following steps may need to be performed by the VMM to process an external interrupt:

- **Host Owned I/O Devices:** For host-owned I/O devices, the interrupting device is owned by the VMM (or hosting OS in a hosted VMM). In this model, the interrupt service routine in the VMM/host driver is invoked and, upon ISR completion, the appropriate write sequences (TPR updates, EOI etc.) to respective interrupt controllers are performed as normal. If the work completion indicated by the driver implies virtual device activity, the VMM runs the virtual device emulation. Depending on the device class, physical device activity could imply activity by multiple virtual devices mapped over the device. For each affected virtual device, the VMM injects a virtual external interrupt event to respective guest virtual machines. The guest driver interacts with the emulated virtual device to process the virtual interrupt. The interrupt controller emulation in the VMM supports various guest accesses to the VMM’s virtual interrupt controller.
- **Guest Assigned I/O Devices:** For assigned I/O devices, either the VMM uses a software proxy or it can directly map the physical device to the assigned VM. In both cases, servicing of the interrupt condition on the physical device is initiated by the driver running inside the guest VM. With host control of external interrupts, interrupts from assigned physical devices cause VM exits to the VMM and vectoring through the host IDT to the registered VMM interrupt handler. To unblock delivery of other low priority platform interrupts, the VMM interrupt handler must mask the interrupt source (for level triggered interrupts) and issue the appropriate EOI write sequences.

Once the physical interrupt source is masked and the platform EOI generated, the VMM can map the host vector to its corresponding guest vector to inject the virtual interrupt into the assigned VM. The guest software does EOI write sequences to its virtual interrupt controller after completing interrupt processing. For level triggered interrupts, these EOI writes to the virtual interrupt controller may be trapped by the VMM which may in turn unmask the previously masked interrupt source.

29.3.3.4 Generation of Virtual Interrupt Events by VMM

The following provides some of the general steps that need to be taken by VMM designs when generating virtual interrupts:

1. Check virtual processor interruptibility state. The virtual processor interruptibility state is reflected in the guest RFLAGS.IF flag and the processor interruptibility-state saved in the guest state area of the controlling-VMCS. If RFLAGS.IF is set and the interruptibility state indicates readiness to take external interrupts (STI-masking and MOV-SS/POP-SS-masking bits are clear), the guest virtual processor is ready to take external interrupts. If the VMM design supports non-active guest sleep states, the VMM needs to make sure the current guest sleep state allows injection of external interrupt events.
2. If the guest virtual processor state is currently not interruptible, a VMM may utilize the “interrupt-window exiting” VM-execution to notify the VM (through a VM exit) when the virtual processor state changes to interruptible state.
3. Check the virtual interrupt controller state. If the guest VM exposes a virtual local APIC, the current value of its processor priority register specifies if guest software allows dispensing an external virtual interrupt with a specific priority to the virtual processor. If the virtual interrupt is routed through the local vector table (LVT) entry of the local APIC, the mask bits in the corresponding LVT entry specifies if the interrupt is currently masked. Similarly, the virtual interrupt controller’s current mask (IO-APIC or PIC) and priority settings reflect guest state to accept specific external interrupts. The VMM needs to check both the virtual processor and interrupt controller states to verify its guest interruptibility state. If the guest is currently interruptible, the VMM can inject the virtual interrupt. If the current guest state does not allow injecting a virtual interrupt, the interrupt needs to be queued by the VMM until it can be delivered.
4. Prioritize the use of VM-entry event injection. A VMM may use VM-entry event injection to deliver various virtual events (such as external interrupts, exceptions, traps, and so forth). VMM designs may prioritize use of virtual-interrupt injection between these event types. Since each VM entry allows injection of one event, depending on the VMM event priority policies, the VMM may need to queue the external virtual interrupt if a higher priority event is to be delivered on the next VM entry. Since the VMM has masked this particular interrupt source (if it was level triggered) and done EOI to the platform interrupt controller, other platform interrupts can be serviced while this virtual interrupt event is queued for later delivery to the VM.

5. Update the virtual interrupt controller state. When the above checks have passed, before generating the virtual interrupt to the guest, the VMM updates the virtual interrupt controller state (Local-APIC, IO-APIC and/or PIC) to reflect assertion of the virtual interrupt. This involves updating the various interrupt capture registers, and priority registers as done by the respective hardware interrupt controllers. Updating the virtual interrupt controller state is required for proper interrupt event processing by guest software.
6. Inject the virtual interrupt on VM entry. To inject an external virtual interrupt to a guest VM, the VMM sets up the VM-entry interruption-information field in the guest controlling-VMCS before entry to guest using VMRESUME. Upon VM entry, the processor will use this vector to access the gate in guest's IDT and the value of RFLAGS and EIP in guest-state area of controlling-VMCS is pushed on the guest stack. If the guest RFLAGS.IF is clear, the STI-masking bit is set, or the MOV- SS/POP-SS-masking bit is set, the VM entry will fail and the processor will load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 22.7).

29.4 ERROR HANDLING BY VMM

Error conditions may occur during VM entries and VM exits and a few other situations. This section describes how VMM should handle these error conditions, including triple faults and machine check exceptions.

29.4.1 VM-Exit Failures

All VM exits load processor state from the host-state area of the VMCS that was the controlling VMCS before the VM exit. This state is checked for consistency while being loaded. Because the host-state is checked on VM entry, these checks will generally succeed. Failure is possible only if host software is incorrect or if VMCS data in the VMCS region in memory has been written by guest software (or by I/O DMA) since the last VM entry. VM exits may fail for the following reasons:

- There was a failure on storing guest MSRs.
- There was failure in loading a PDPTR.
- The controlling VMCS has been corrupted (through writes to the corresponding VMCS region) in such a way that the implementation cannot complete the VM exit.
- There was a failure on loading host MSRs.
- A machine check occurred.

If one of these problems occurs on a VM exit, a VMX abort results.

29.4.2 Machine Check Considerations

The following sequence determine how machine checks are handled during VMXON, VMXOFF, VM entries, and VM exits:

- VMXOFF and VMXON:

  If a machine check occurs during VMXOFF or VMXON and CR4.MCE = 1, a machine-check exception (#MC) is generated. If CR4.MCE = 0, the processor goes to shutdown state.

- VM entry:

  If a machine check occurs during VM entry, one of the following two treatments must occur:

  a. Normal delivery. If CR4.MCE = 1, delivery of a machine-check exception (#MC) through the host IDT occurs. If CR4.MCE = 0, the processor goes to shutdown state.

  b. Load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 22.7). The basic exit reason will be "VM-entry failure due to machine check."

  If the machine check occurs after any guest state has been loaded, option b above must be used. If the machine check occurs while checking host state and VMX controls (or while reporting a failure due to such checks), option a should be preferred; however, an implementation may use b, since software will not be able to tell whether any guest state has been loaded.

- VM exit:

  If a machine check occurs during VM exit, one of the following two treatments must occur:

  - Normal delivery. If CR4.MCE = 1, delivery of a machine-check exception (#MC) through the guest IDT. If CR4.MCE = 0, the processor goes to shutdown state.

  - Fail the VM exit. If the VM exit is to VMX root operation, a VMX abort will result; it will block events as done normally in VMX abort. The VMX abort indicator will show a machine check has induced the abort operation.

  If a machine check is induced by an action in VMX non-root operation before any determination is made that the inducing action may cause a VM exit, that machine check should be considered as happening during guest execution in VMX non-root operation. This is the case even if the part of the action that caused the machine check was VMX-specific (for example: the processor's consulting an I/O bitmap). A machine-check exception will occur. If bit 12H of the exception bitmap is cleared to 0, a machine-check exception could be delivered to the guest through gate 12H of its IDT; if the bit is set to 1, the machine-check exception will cause a VM exit.

NOTE

The state saved in the guest-state area on VM exits due to machine-check exceptions should be considered suspect. A VMM should consult the RIPV and EIPV bits in the IA32_MCG_STATUS MSR before resuming a guest that caused a VM exit due to a machine-check exception.

29.5 HANDLING ACTIVITY STATES BY VMM

A VMM might place a logic processor in the wait-for-SIPI activity state if supporting certain guest operating system using the multi-processor (MP) start-up algorithm. A guest with direct access to the physical local APIC and using the MP start-up algorithm sends an INIT-SIPI-SIPI IPI sequence to start the application processor. In order to trap the SIPIs, the VMM must start the logic processor which is the target of the SIPIs in wait-for-SIPI mode.

APPENDIX A
PERFORMANCE-MONITORING EVENTS

This appendix lists the performance-monitoring events that can be monitored with the Intel 64 or IA-32 processors. The ability to monitor performance events and the events that can be monitored in these processors are mostly model-specific, except for architectural performance events, described in Section A.1.

Non-architectural performance events (i.e. model-specific events) are listed for each generation of microarchitecture:

- Section A.3 - Processors based on Intel Core microarchitecture
- Section A.4 - Processors based on Intel Atom microarchitecture
- Section A.5 - Intel Core Solo and Intel Core Duo processors
- Section A.6 - Processors based on Intel NetBurst microarchitecture
- Section A.7 - Pentium M family processors
- Section A.8 - P6 family processors
- Section A.9 - Pentium processors

NOTE

These performance-monitoring events are intended to be used as guides for performance tuning. The counter values reported by the performance-monitoring events are approximate and believed to be useful as relative guides for tuning software. Known discrepancies are documented where applicable.

A.1 ARCHITECTURAL PERFORMANCE-MONITORING EVENTS

Architectural performance events are introduced in Intel Core Solo and Intel Core Duo processors. They are also supported on processors based on Intel Core microarchitecture. Table A-1 lists pre-defined architectural performance events that can be configured using general-purpose performance counters and associated event-select registers.

Table A-1. Architectural Performance Events

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
3CH	UnHalted Core Cycles	00H	Unhalted core cycles	
3CH	UnHalted Reference Cycles	01H	Unhalted reference cycles	Measures bus cycle[1]
C0H	Instruction Retired	00H	Instruction retired	
2EH	LLC Reference	4FH	LL cache references	
2EH	LLC Misses	41H	LL cache misses	
C4H	Branch Instruction Retired	00H	Branch instruction retired	
C5H	Branch Misses Retired	00H	Mispredicted Branch Instruction retired	

NOTES:

1. Implementation of this event in Intel Core 2 processor family, Intel Core Duo, and Intel Core Solo processors measures bus clocks.

A.2 PERFORMANCE MONITORING EVENTS FOR INTEL® XEON® PROCESSOR 5200, 5400 SERIES AND INTEL® CORE™2 EXTREME PROCESSORS QX 9000 SERIES

Processors based on the Enhanced Intel Core microarchitecture support the architectural and non-architectural performance-monitoring events listed in Table A-1 and Table A-4. In addition, they also support the following non-architectural performance-monitoring events listed in Table A-2.

Table A-2. Non-Architectural Performance Events for Processors based on Enhanced Intel Core Microarchitecture

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
C0H	08H	INST_RETIRED.VM_HOST	Instruction retired while in VMX root operations	
D2H	10H	RAT_STAALS.OTHER_SERIALIZATION_STALLS	This events counts the number of stalls due to other RAT resource serialization not counted by Umask value 0FH.	

A.3 PERFORMANCE MONITORING EVENTS FOR INTEL® XEON® PROCESSOR 3000, 3200, 5100, 5300 SERIES AND INTEL® CORE™2 DUO PROCESSORS

Processors based on the Intel Core microarchitecture support architectural and non-architectural performance-monitoring events.

Fixed-function performance counters are introduced first on processors based on Intel Core microarchitecture. Table A-3 lists pre-defined performance events that can be counted using fixed-function performance counters.

Table A-3. Fixed-Function Performance Counter and Pre-defined Performance Events

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
MSR_PERF_FIXED_CTR0	309H	Instr_Retired.Any	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continue counting during hardware interrupts, traps, and inside interrupt handlers
MSR_PERF_FIXED_CTR1	30AH	CPU_CLK_UNHALTED.CORE	This event counts the number of core cycles while the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. This event is a component in many key event ratios. The core frequency may change from time to time due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason this event may have a changing ratio with regards to time. When the core frequency is constant, this event can approximate elapsed time while the core was not in halt state.
MSR_PERF_FIXED_CTR2	30BH	CPU_CLK_UNHALTED.REF	This event counts the number of reference cycles when the core is not in a halt state. The core enters the halt state when it is running the HLT instruction or the MWAIT instruction.

Table A-3. Fixed-Function Performance Counter and Pre-defined Performance Events (Contd.)

Fixed-Function Performance Counter	Address	Event Mask Mnemonic	Description
			This event is not affected by core frequency changes (e.g., P states, TM2 transitions) but counts at the same frequency as the time stamp counter. This event can approximate elapsed time while the core was not in halt state. This event has a constant ratio with the CPU_CLK_UNHALTED.BUS event.

Table A-4 lists general-purpose non-architectural performance-monitoring events supported in processors based on Intel Core microarchitecture. For convenience, Table A-4 also includes architectural events and describes minor model-specific behavior where applicable. Software must use a general-purpose performance counter to count events listed in Table A-4.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture

Event Num	Umask Value	Event Name	Definition	Description and Comment
03H	02H	LOAD_BLOCK.STA	Loads blocked by a preceding store with unknown address	This event indicates that loads are blocked by preceding stores. A load is blocked when there is a preceding store to an address that is not yet calculated. The number of events is greater or equal to the number of load operations that were blocked. If the load and the store are always to different addresses, check why the memory disambiguation mechanism is not working. To avoid such blocks, increase the distance between the store and the following load so that the store address is known at the time the load is dispatched.
03H	04H	LOAD_BLOCK.STD	Loads blocked by a preceding store with unknown data	This event indicates that loads are blocked by preceding stores. A load is blocked when there is a preceding store to the same address and the stored data value is not yet known. The number of events is greater or equal to the number of load operations that were blocked. To avoid such blocks, increase the distance between the store and the dependant load, so that the store data is known at the time the load is dispatched.
03H	08H	LOAD_BLOCK. OVERLAP_STORE	Loads that partially overlap an earlier store, or 4-Kbyte aliased with a previous store	This event indicates that loads are blocked due to a variety of reasons. Some of the triggers for this event are when a load is blocked by a preceding store, in one of the following: ▪ Some of the loaded byte locations are written by the preceding store and some are not. ▪ The load is from bytes written by the preceding store, the store is aligned to its size and either: ▪ The load's data size is one or two bytes and it is not aligned to the store. ▪ The load's data size is of four or eight bytes and the load is misaligned.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
				▪ The load is from bytes written by the preceding store, the store is misaligned and the load is not aligned on the beginning of the store. ▪ The load is split over an eight byte boundary (excluding 16-byte loads). ▪ The load and store have the same offset relative to the beginning of different 4-KByte pages. This case is also called 4-KByte aliasing. ▪ In all these cases the load is blocked until after the blocking store retires and the stored data is committed to the cache hierarchy.
03H	10H	LOAD_BLOCK.UNTIL_RETIRE	Loads blocked until retirement	This event indicates that load operations were blocked until retirement. The number of events is greater or equal to the number of load operations that were blocked. This includes mainly uncacheable loads and split loads (loads that cross the cache line boundary) but may include other cases where loads are blocked until retirement.
03H	20H	LOAD_BLOCK.L1D	Loads blocked by the L1 data cache	This event indicates that loads are blocked due to one or more reasons. Some triggers for this event are: ▪ The number of L1 data cache misses exceeds the maximum number of outstanding misses supported by the processor. This includes misses generated as result of demand fetches, software prefetches or hardware prefetches. ▪ Cache line split loads. ▪ Partial reads, such as reads to uncacheable memory, I/O instructions and more. ▪ A locked load operation is in progress. The number of events is greater or equal to the number of load operations that were blocked.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
04H	01H	SB_DRAIN_CYCLES	Cycles while stores are blocked due to store buffer drain	This event counts every cycle during which the store buffer is draining. This includes: ▪ Serializing operations such as CPUID ▪ Synchronizing operations such as XCHG ▪ Interrupt acknowledgment ▪ Other conditions, such as cache flushing
04H	02H	STORE_BLOCK.ORDER	Cycles while store is waiting for a preceding store to be globally observed	This event counts the total duration, in number of cycles, which stores are waiting for a preceding stored cache line to be observed by other cores. This situation happens as a result of the strong store ordering behavior, as defined in “Memory Ordering,” Chapter 7, *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*. The stall may occur and be noticeable if there are many cases when a store either misses the L1 data cache or hits a cache line in the Shared state. If the store requires a bus transaction to read the cache line then the stall ends when snoop response for the bus transaction arrives.
04H	08H	STORE_BLOCK.SNOOP	A store is blocked due to a conflict with an external or internal snoop.	This event counts the number of cycles the store port was used for snooping the L1 data cache and a store was stalled by the snoop. The store is typically resubmitted one cycle later.
06H	00H	SEGMENT_REG_LOADS	Number of segment register loads	This event counts the number of segment register load operations. Instructions that load new values into segment registers cause a penalty.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
				This event indicates performance issues in 16-bit code. If this event occurs frequently, it may be useful to calculate the number of instructions retired per segment register load. If the resulting calculation is low (on average a small number of instructions are executed between segment register loads), then the code's segment register usage should be optimized. As a result of branch misprediction, this event is speculative and may include segment register loads that do not actually occur. However, most segment register loads are internally serialized and such speculative effects are minimized.
07H	00H	SSE_PRE_EXEC.NTA	Streaming SIMD Extensions (SSE) Prefetch NTA instructions executed	This event counts the number of times the SSE instruction prefetchNTA is executed. This instruction prefetches the data to the L1 data cache.
07H	01H	SSE_PRE_EXEC.L1	Streaming SIMD Extensions (SSE) PrefetchT0 instructions executed	This event counts the number of times the SSE instruction prefetchT0 is executed. This instruction prefetches the data to the L1 data cache and L2 cache.
07H	02H	SSE_PRE_EXEC.L2	Streaming SIMD Extensions (SSE) PrefetchT1 and PrefetchT2 instructions executed	This event counts the number of times the SSE instructions prefetchT1 and prefetchT2 are executed. These instructions prefetch the data to the L2 cache.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
07H	03H	SSE_PRE_EXEC.STORES	Streaming SIMD Extensions (SSE) Weakly-ordered store instructions executed	This event counts the number of times SSE non-temporal store instructions are executed.
08H	01H	DTLB_MISSES.ANY	Memory accesses that missed the DTLB	This event counts the number of Data Table Lookaside Buffer (DTLB) misses. The count includes misses detected as a result of speculative accesses. Typically a high count for this event indicates that the code accesses a large number of data pages.
08H	02H	DTLB_MISSES.MISS_LD	DTLB misses due to load operations	This event counts the number of Data Table Lookaside Buffer (DTLB) misses due to load operations. This count includes misses detected as a result of speculative accesses.
08H	04H	DTLB_MISSES.L0_MISS_LD	L0 DTLB misses due to load operations	This event counts the number of level 0 Data Table Lookaside Buffer (DTLB0) misses due to load operations. This count includes misses detected as a result of speculative accesses. Loads that miss that DTLB0 and hit the DTLB1 can incur two-cycle penalty.
08H	08H	DTLB_MISSES.MISS_ST	TLB misses due to store operations	This event counts the number of Data Table Lookaside Buffer (DTLB) misses due to store operations. This count includes misses detected as a result of speculative accesses. Address translation for store operations is performed in the DTLB1.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
09H	01H	MEMORY_ DISAMBIGUATION. RESET	Memory disambiguation reset cycles	This event counts the number of cycles during which memory disambiguation misprediction occurs. As a result the execution pipeline is cleaned and execution of the mispredicted load instruction and all succeeding instructions restarts.
				This event occurs when the data address accessed by a load instruction, collides infrequently with preceding stores, but usually there is no collision. It happens rarely, and may have a penalty of about 20 cycles.
09H	02H	MEMORY_DISAMBI GUATION.SUCCESS	Number of loads successfully disambiguated.	This event counts the number of load operations that were successfully disambiguated. Loads are preceded by a store with an unknown address, but they are not blocked.
0CH	01H	PAGE_WALKS .COUNT	Number of page-walks executed	This event counts the number of page-walks executed due to either a DTLB or ITLB miss. The page walk duration, PAGE_WALKS.CYCLES, divided by number of page walks is the average duration of a page walk. The average can hint whether most of the page-walks are satisfied by the caches or cause an L2 cache miss.
0CH	02H	PAGE_WALKS. CYCLES	Duration of page-walks in core cycles	This event counts the duration of page-walks in core cycles. The paging mode in use typically affects the duration of page walks. Page walk duration divided by number of page walks is the average duration of page-walks. The average can hint at whether most of the page-walks are satisfied by the caches or cause an L2 cache miss.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
10H	00H	FP_COMP_OPS_EXE	Floating point computational micro-ops executed	This event counts the number of floating point computational micro-ops executed. Use IA32_PMC0 only.
11H	00H	FP_ASSIST	Floating point assists	This event counts the number of floating point operations executed that required micro-code assist intervention. Assists are required in the following cases: ▪ Streaming SIMD Extensions (SSE) instructions:
				▪ Denormal input when the DAZ (Denormals Are Zeros) flag is off ▪ Underflow result when the FTZ (Flush To Zero) flag is off ▪ X87 instructions: ▪ NaN or denormal are loaded to a register or used as input from memory ▪ Division by 0 ▪ Underflow output Use IA32_PMC1 only.
12H	00H	MUL	Multiply operations executed	This event counts the number of multiply operations executed. This includes integer as well as floating point multiply operations. Use IA32_PMC1 only.
13H	00H	DIV	Divide operations executed	This event counts the number of divide operations executed. This includes integer divides, floating point divides and square-root operations executed. Use IA32_PMC1 only.
14H	00H	CYCLES_DIV_BUSY	Cycles the divider busy	This event counts the number of cycles the divider is busy executing divide or square root operations. The divide can be integer, X87 or Streaming SIMD Extensions (SSE). The square root operation can be either X87 or SSE. Use IA32_PMC0 only.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
18H	00H	IDLE_DURING _DIV	Cycles the divider is busy and all other execution units are idle.	This event counts the number of cycles the divider is busy (with a divide or a square root operation) and no other execution unit or load operation is in progress. Load operations are assumed to hit the L1 data cache. This event considers only micro-ops dispatched after the divider started operating. Use IA32_PMC0 only.
19H	00H	DELAYED_ BYPASS.FP	Delayed bypass to FP operation	This event counts the number of times floating point operations use data immediately after the data was generated by a non-floating point execution unit. Such cases result in one penalty cycle due to data bypass between the units. Use IA32_PMC1 only.
19H	01H	DELAYED_ BYPASS.SIMD	Delayed bypass to SIMD operation	This event counts the number of times SIMD operations use data immediately after the data was generated by a non-SIMD execution unit. Such cases result in one penalty cycle due to data bypass between the units. Use IA32_PMC1 only.
19H	02H	DELAYED_ BYPASS.LOAD	Delayed bypass to load operation	This event counts the number of delayed bypass penalty cycles that a load operation incurred. When load operations use data immediately after the data was generated by an integer execution unit, they may (pending on certain dynamic internal conditions) incur one penalty cycle due to delayed data bypass between the units. Use IA32_PMC1 only.
21H	See Table 18-8	L2_ADS.(Core)	Cycles L2 address bus is in use	This event counts the number of cycles the L2 address bus is being used for accesses to the L2 cache or bus queue. It can count occurrences for this core or both cores.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
23H	See Table 18-8	L2_DBUS_BUSY _RD.(Core)	Cycles the L2 transfers data to the core	This event counts the number of cycles during which the L2 data bus is busy transferring data from the L2 cache to the core. It counts for all L1 cache misses (data and instruction) that hit the L2 cache. This event can count occurrences for this core or both cores.
24H	Com-bined mask from Table 18-8 and Table 18-10	L2_LINES_IN. (Core, Prefetch)	L2 cache misses	This event counts the number of cache lines allocated in the L2 cache. Cache lines are allocated in the L2 cache as a result of requests from the L1 data and instruction caches and the L2 hardware prefetchers to cache lines that are missing in the L2 cache. This event can count occurrences for this core or both cores. It can also count demand requests and L2 hardware prefetch requests together or separately.
25H	See Table 18-8	L2_M_LINES_IN. (Core)	L2 cache line modifications	This event counts whenever a modified cache line is written back from the L1 data cache to the L2 cache. This event can count occurrences for this core or both cores.
26H	See Table 18-8 and Table 18-10	L2_LINES_OUT. (Core, Prefetch)	L2 cache lines evicted	This event counts the number of L2 cache lines evicted. This event can count occurrences for this core or both cores. It can also count evictions due to demand requests and L2 hardware prefetch requests together or separately.
27H	See Table 18-8 and Table 18-10	L2_M_LINES_OUT.(Core, Prefetch)	Modified lines evicted from the L2 cache	This event counts the number of L2 modified cache lines evicted. These lines are written back to memory unless they also exist in a modified-state in one of the L1 data caches. This event can count occurrences for this core or both cores. It can also count evictions due to demand requests and L2 hardware prefetch requests together or separately.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
28H	Combined mask from Table 18-8 and Table 18-11	L2_IFETCH.(Core, Cache Line State)	L2 cacheable instruction fetch requests	This event counts the number of instruction cache line requests from the IFU. It does not include fetch requests from uncacheable memory. It does not include ITLB miss accesses. This event can count occurrences for this core or both cores. It can also count accesses to cache lines at different MESI states.
29H	Combined mask from Table 18-8, Table 18-10, and Table 18-11	L2_LD.(Core, Prefetch, Cache Line State)	L2 cache reads	This event counts L2 cache read requests coming from the L1 data cache and L2 prefetchers. The event can count occurrences: ▪ for this core or both cores ▪ due to demand requests and L2 hardware prefetch requests together or separately ▪ of accesses to cache lines at different MESI states
2AH	See Table 18-8 and Table 18-11	L2_ST.(Core, Cache Line State)	L2 store requests	This event counts all store operations that miss the L1 data cache and request the data from the L2 cache. The event can count occurrences for this core or both cores. It can also count accesses to cache lines at different MESI states.
2BH	See Table 18-8 and Table 18-11	L2_LOCK.(Core, Cache Line State)	L2 locked accesses	This event counts all locked accesses to cache lines that miss the L1 data cache. The event can count occurrences for this core or both cores. It can also count accesses to cache lines at different MESI states.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
2EH	See Table 18-8, Table 18-10, and Table 18-11	L2_RQSTS.(Core, Prefetch, Cache Line State)	L2 cache requests	This event counts all completed L2 cache requests. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, instruction fetches, and all L2 hardware prefetch requests. This event can count occurrences: ▪ for this core or both cores. ▪ due to demand requests and L2 hardware prefetch requests together, or separately ▪ of accesses to cache lines at different MESI states
2EH	41H	L2_RQSTS.SELF.DEMAND.I_STATE	L2 cache demand requests from this core that missed the L2	This event counts all completed L2 cache demand requests from this core that miss the L2 cache. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, and instruction fetches. This is an architectural performance event.
2EH	4FH	L2_RQSTS.SELF.DEMAND.MESI	L2 cache demand requests from this core	This event counts all completed L2 cache demand requests from this core. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, and instruction fetches. This is an architectural performance event.
30H	See Table 18-8, Table 18-10, and Table 18-11	L2_REJECT_BUSQ.(Core, Prefetch, Cache Line State)	Rejected L2 cache requests	This event indicates that a pending L2 cache request that requires a bus transaction is delayed from moving to the bus queue. Some of the reasons for this event are: ▪ The bus queue is full. ▪ The bus queue already holds an entry for a cache line in the same set. The number of events is greater or equal to the number of requests that were rejected. ▪ for this core or both cores. ▪ due to demand requests and L2 hardware prefetch requests together, or separately. ▪ of accesses to cache lines at different MESI states.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
32H	See Table 18-8	L2_NO_REQ.(Core)	Cycles no L2 cache requests are pending	This event counts the number of cycles that no L2 cache requests were pending from a core. When using the BOTH_CORE modifier, the event counts only if none of the cores have a pending request. The event counts also when one core is halted and the other is not halted. The event can count occurrences for this core or both cores.
3AH	00H	EIST_TRANS	Number of Enhanced Intel SpeedStep Technology (EIST) transitions	This event counts the number of transitions that include a frequency change, either with or without voltage change. This includes Enhanced Intel SpeedStep Technology (EIST) and TM2 transitions. The event is incremented only while the counting core is in C0 state. Since transitions to higher-numbered CxE states and TM2 transitions include a frequency change or voltage transition, the event is incremented accordingly.
3BH	C0H	THERMAL_TRIP	Number of thermal trips	This event counts the number of thermal trips. A thermal trip occurs whenever the processor temperature exceeds the thermal trip threshold temperature. Following a thermal trip, the processor automatically reduces frequency and voltage. The processor checks the temperature every millisecond and returns to normal when the temperature falls below the thermal trip threshold temperature.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
3CH	00H	CPU_CLK_ UNHALTED. CORE_P	Core cycles when core is not halted	This event counts the number of core cycles while the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. This event is a component in many key event ratios. The core frequency may change due to transitions associated with Enhanced Intel SpeedStep Technology or TM2. For this reason, this event may have a changing ratio in regard to time. When the core frequency is constant, this event can give approximate elapsed time while the core not in halt state. This is an architectural performance event.
3CH	01H	CPU_CLK_ UNHALTED.BUS	Bus cycles when core is not halted	This event counts the number of bus cycles while the core is not in the halt state. This event can give a measurement of the elapsed time while the core was not in the halt state. The core enters the halt state when it is running the HLT instruction. The event also has a constant ratio with CPU_CLK_UNHALTED.REF event, which is the maximum bus to processor frequency ratio. Non-halted bus cycles are a component in many key event ratios.
3CH	02H	CPU_CLK_ UNHALTED.NO _OTHER	Bus cycles when core is active and the other is halted	This event counts the number of bus cycles during which the core remains non-halted and the other core on the processor is halted. This event can be used to determine the amount of parallelism exploited by an application or a system. Divide this event count by the bus frequency to determine the amount of time that only one core was in use.
40H	See Table 18-11	L1D_CACHE_LD. (Cache Line State)	L1 cacheable data reads	This event counts the number of data reads from cacheable memory. Locked reads are not counted.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
41H	See Table 18-11	L1D_CACHE_ST.(Cache Line State)	L1 cacheable data writes	This event counts the number of data writes to cacheable memory. Locked writes are not counted.
42H	See Table 18-11	L1D_CACHE_ LOCK.(Cache Line State)	L1 data cacheable locked reads	This event counts the number of locked data reads from cacheable memory.
42H	10H	L1D_CACHE_ LOCK_DURATION	Duration of L1 data cacheable locked operation	This event counts the number of cycles during which any cache line is locked by any locking instruction. Locking happens at retirement and therefore the event does not occur for instructions that are speculatively executed. Locking duration is shorter than locked instruction execution duration.
43H	01H	L1D_ALL_REF	All references to the L1 data cache	This event counts all references to the L1 data cache, including all loads and stores with any memory types. The event counts memory accesses only when they are actually performed. For example, a load blocked by unknown store address and later performed is only counted once. The event includes non-cacheable accesses, such as I/O accesses.
43H	02H	L1D_ALL_ CACHE_REF	L1 Data cacheable reads and writes	This event counts the number of data reads and writes from cacheable memory, including locked operations. This event is a sum of: ▪ L1D_CACHE_LD.MESI ▪ L1D_CACHE_ST.MESI ▪ L1D_CACHE_LOCK.MESI
45H	0FH	L1D_REPL	Cache lines allocated in the L1 data cache	This event counts the number of lines brought into the L1 data cache.
46H	00H	L1D_M_REPL	Modified cache lines allocated in the L1 data cache	This event counts the number of modified lines brought into the L1 data cache.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
47H	00H	L1D_M_EVICT	Modified cache lines evicted from the L1 data cache	This event counts the number of modified lines evicted from the L1 data cache, whether due to replacement or by snoop HITM intervention.
48H	00H	L1D_PEND_ MISS	Total number of outstanding L1 data cache misses at any cycle	This event counts the number of outstanding L1 data cache misses at any cycle. An L1 data cache miss is outstanding from the cycle on which the miss is determined until the first chunk of data is available. This event counts: ▪ all cacheable demand requests ▪ L1 data cache hardware prefetch requests ▪ requests to write through memory ▪ requests to write combine memory Uncacheable requests are not counted. The count of this event divided by the number of L1 data cache misses, L1D_REPL, is the average duration in core cycles of an L1 data cache miss.
49H	01H	L1D_SPLIT.LOADS	Cache line split loads from the L1 data cache	This event counts the number of load operations that span two cache lines. Such load operations are also called split loads. Split load operations are executed at retirement.
49H	02H	L1D_SPLIT. STORES	Cache line split stores to the L1 data cache	This event counts the number of store operations that span two cache lines.
4BH	00H	SSE_PRE_ MISS.NTA	Streaming SIMD Extensions (SSE) Prefetch NTA instructions missing all cache levels	This event counts the number of times the SSE instructions prefetchNTA were executed and missed all cache levels. Due to speculation an executed instruction might not retire. This instruction prefetches the data to the L1 data cache.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
4BH	01H	SSE_PRE_ MISS.L1	Streaming SIMD Extensions (SSE) PrefetchT0 instructions missing all cache levels	This event counts the number of times the SSE instructions prefetchT0 were executed and missed all cache levels. Due to speculation executed instruction might not retire. The prefetchT0 instruction prefetches data to the L2 cache and L1 data cache.
4BH	02H	SSE_PRE_ MISS.L2	Streaming SIMD Extensions (SSE) PrefetchT1 and PrefetchT2 instructions missing all cache levels	This event counts the number of times the SSE instructions prefetchT1 and prefetchT2 were executed and missed all cache levels. Due to speculation, an executed instruction might not retire. The prefetchT1 and PrefetchNT2 instructions prefetch data to the L2 cache.
4CH	00H	LOAD_HIT_PRE	Load operations conflicting with a software prefetch to the same address	This event counts load operations sent to the L1 data cache while a previous Streaming SIMD Extensions (SSE) prefetch instruction to the same cache line has started prefetching but has not yet finished.
4EH	10H	L1D_PREFETCH. REQUESTS	L1 data cache prefetch requests	This event counts the number of times the L1 data cache requested to prefetch a data cache line. Requests can be rejected when the L2 cache is busy and resubmitted later or lost. All requests are counted, including those that are rejected.
60H	See Table 18-8 and Table 18-9	BUS_REQUEST_ OUTSTANDING. (Core and Bus Agents)	Outstanding cacheable data read bus requests duration	This event counts the number of pending full cache line read transactions on the bus occurring in each cycle. A read transaction is pending from the cycle it is sent on the bus until the full cache line is received by the processor. The event counts only full-line cacheable read requests from either the L1 data cache or the L2 prefetchers. It does not count Read for Ownership transactions, instruction byte fetch transactions, or any other bus transaction.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
61H	See Table 18-9.	BUS_BNR_DRV. (Bus Agents)	Number of Bus Not Ready signals asserted	This event counts the number of Bus Not Ready (BNR) signals that the processor asserts on the bus to suspend additional bus requests by other bus agents.
				A bus agent asserts the BNR signal when the number of data and snoop transactions is close to the maximum that the bus can handle. To obtain the number of bus cycles during which the BNR signal is asserted, multiply the event count by two. While this signal is asserted, new transactions cannot be submitted on the bus. As a result, transaction latency may have higher impact on program performance.
62H	See Table 18-9	BUS_DRDY_ CLOCKS.(Bus Agents)	Bus cycles when data is sent on the bus	This event counts the number of bus cycles during which the DRDY (Data Ready) signal is asserted on the bus. The DRDY signal is asserted when data is sent on the bus. With the 'THIS_AGENT' mask this event counts the number of bus cycles during which this agent (the processor) writes data on the bus back to memory or to other bus agents. This includes all explicit and implicit data writebacks, as well as partial writes. With the 'ALL_AGENTS' mask, this event counts the number of bus cycles during which any bus agent sends data on the bus. This includes all data reads and writes on the bus.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
63H	See Table 18-8 and Table 18-9	BUS_LOCK_ CLOCKS.(Core and Bus Agents)	Bus cycles when a LOCK signal asserted	This event counts the number of bus cycles, during which the LOCK signal is asserted on the bus. A LOCK signal is asserted when there is a locked memory access, due to: ▪ uncacheable memory ▪ locked operation that spans two cache lines ▪ page-walk from an uncacheable page table Bus locks have a very high performance penalty and it is highly recommended to avoid such accesses.
64H	See Table 18-8	BUS_DATA_ RCV.(Core)	Bus cycles while processor receives data	This event counts the number of bus cycles during which the processor is busy receiving data.
65H	See Table 18-8 and Table 18-9	BUS_TRANS_BRD.(Core and Bus Agents)	Burst read bus transactions	This event counts the number of burst read transactions including: ▪ L1 data cache read misses (and L1 data cache hardware prefetches) ▪ L2 hardware prefetches by the DPL and L2 streamer ▪ IFU read misses of cacheable lines. It does not include RFO transactions.
66H	See Table 18-8 and Table 18-9.	BUS_TRANS_RFO.(Core and Bus Agents)	RFO bus transactions	This event counts the number of Read For Ownership (RFO) bus transactions, due to store operations that miss the L1 data cache and the L2 cache. It also counts RFO bus transactions due to locked operations.
67H	See Table 18-8 and Table 18-9.	BUS_TRANS_WB. (Core and Bus Agents)	Explicit writeback bus transactions	This event counts all explicit writeback bus transactions due to dirty line evictions. It does not count implicit writebacks due to invalidation by a snoop request.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
68H	See Table 18-8 and Table 18-9	BUS_TRANS_ IFETCH.(Core and Bus Agents)	Instruction-fetch bus transactions	This event counts all instruction fetch full cache line bus transactions.
69H	See Table 18-8 and Table 18-9	BUS_TRANS_ INVAL.(Core and Bus Agents)	Invalidate bus transactions	This event counts all invalidate transactions. Invalidate transactions are generated when: ▪ A store operation hits a shared line in the L2 cache. ▪ A full cache line write misses the L2 cache or hits a shared line in the L2 cache.
6AH	See Table 18-8 and Table 18-9	BUS_TRANS_ PWR.(Core and Bus Agents)	Partial write bus transaction	This event counts partial write bus transactions.
6BH	See Table 18-8 and Table 18-9	BUS_TRANS _P.(Core and Bus Agents)	Partial bus transactions	This event counts all (read and write) partial bus transactions.
6CH	See Table 18-8 and Table 18-9	BUS_TRANS_IO.(Core and Bus Agents)	IO bus transactions	This event counts the number of completed I/O bus transactions as a result of IN and OUT instructions. The count does not include memory mapped IO.
6DH	See Table 18-8 and Table 18-9	BUS_TRANS_ DEF.(Core and Bus Agents)	Deferred bus transactions	This event counts the number of deferred transactions.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
6EH	See Table 18-8 and Table 18-9	BUS_TRANS_ BURST.(Core and Bus Agents)	Burst (full cache-line) bus transactions	This event counts burst (full cache line) transactions including: ▪ Burst reads ▪ RFOs ▪ Explicit writebacks ▪ Write combine lines
6FH	See Table 18-8 and Table 18-9	BUS_TRANS_ MEM.(Core and Bus Agents)	Memory bus transactions	This event counts all memory bus transactions including: ▪ Burst transactions ▪ Partial reads and writes - invalidate transactions The BUS_TRANS_MEM count is the sum of BUS_TRANS_BURST, BUS_TRANS_P and BUS_TRANS_IVAL.
70H	See Table 18-8 and Table 18-9	BUS_TRANS_ ANY.(Core and Bus Agents)	All bus transactions	This event counts all bus transactions. This includes: ▪ Memory transactions ▪ IO transactions (non memory-mapped) ▪ Deferred transaction completion ▪ Other less frequent transactions, such as interrupts
77H	See Table 18-8 and Table 18-12	EXT_SNOOP. (Bus Agents, Snoop Response)	External snoops	This event counts the snoop responses to bus transactions. Responses can be counted separately by type and by bus agent. With the 'THIS_AGENT' mask, the event counts snoop responses from this processor to bus transactions sent by this processor. With the 'ALL_AGENTS' mask the event counts all snoop responses seen on the bus.
78H	See Table 18-8 and Table 18-13	CMP_SNOOP.(Core, Snoop Type)	L1 data cache snooped by other core	This event counts the number of times the L1 data cache is snooped for a cache line that is needed by the other core in the same processor. The cache line is either missing in the L1 instruction or data caches of the other core, or is available for reading only and the other core wishes to write the cache line.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
				The snoop operation may change the cache line state. If the other core issued a read request that hit this core in E state, typically the state changes to S state in this core. If the other core issued a read for ownership request (due a write miss or hit to S state) that hits this core's cache line in E or S state, this typically results in invalidation of the cache line in this core. If the snoop hits a line in M state, the state is changed at a later opportunity.
				These snoops are performed through the L1 data cache store port. Therefore, frequent snoops may conflict with extensive stores to the L1 data cache, which may increase store latency and impact performance.
7AH	See Table 18-9	BUS_HIT_DRV. (Bus Agents)	HIT signal asserted	This event counts the number of bus cycles during which the processor drives the HIT# pin to signal HIT snoop response.
7BH	See Table 18-9	BUS_HITM_DRV. (Bus Agents)	HITM signal asserted	This event counts the number of bus cycles during which the processor drives the HITM# pin to signal HITM snoop response.
7DH	See Table 18-8	BUSQ_EMPTY. (Core)	Bus queue empty	This event counts the number of cycles during which the core did not have any pending transactions in the bus queue. It also counts when the core is halted and the other core is not halted. This event can count occurrences for this core or both cores.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
7EH	See Table 18-8 and Table 18-9	SNOOP_STALL_ DRV.(Core and Bus Agents)	Bus stalled for snoops	This event counts the number of times that the bus snoop stall signal is asserted. To obtain the number of bus cycles during which snoops on the bus are prohibited, multiply the event count by two. During the snoop stall cycles, no new bus transactions requiring a snoop response can be initiated on the bus. A bus agent asserts a snoop stall signal if it cannot response to a snoop request within three bus cycles.
7FH	See Table 18-8	BUS_IO_WAIT. (Core)	IO requests waiting in the bus queue	This event counts the number of core cycles during which IO requests wait in the bus queue. With the SELF modifier this event counts IO requests per core. With the BOTH_CORE modifier, this event increments by one for any cycle for which there is a request from either core.
80H	00H	L1I_READS	Instruction fetches	This event counts all instruction fetches, including uncacheable fetches that bypass the Instruction Fetch Unit (IFU).
81H	00H	L1I_MISSES	Instruction Fetch Unit misses	This event counts all instruction fetches that miss the Instruction Fetch Unit (IFU) or produce memory requests. This includes uncacheable fetches. An instruction fetch miss is counted only once and not once for every cycle it is outstanding.
82H	02H	ITLB.SMALL_MISS	ITLB small page misses	This event counts the number of instruction fetches from small pages that miss the ITLB.
82H	10H	ITLB.LARGE_MISS	ITLB large page misses	This event counts the number of instruction fetches from large pages that miss the ITLB.
82H	40H	ITLB.FLUSH	ITLB flushes	This event counts the number of ITLB flushes. This usually happens upon CR3 or CR0 writes, which are executed by the operating system during process switches.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
82H	12H	ITLB.MISSES	ITLB misses	This event counts the number of instruction fetches from either small or large pages that miss the ITLB.
83H	02H	INST_QUEUE.FULL	Cycles during which the instruction queue is full	This event counts the number of cycles during which the instruction queue is full. In this situation, the core front-end stops fetching more instructions. This is an indication of very long stalls in the back-end pipeline stages.
86H	00H	CYCLES_L1I_ MEM_STALLED	Cycles during which instruction fetches stalled	This event counts the number of cycles for which an instruction fetch stalls, including stalls due to any of the following reasons: ▪ instruction Fetch Unit cache misses ▪ instruction TLB misses ▪ instruction TLB faults
87H	00H	ILD_STALL	Instruction Length Decoder stall cycles due to a length changing prefix	This event counts the number of cycles during which the instruction length decoder uses the slow length decoder. Usually, instruction length decoding is done in one cycle. When the slow decoder is used, instruction decoding requires 6 cycles. The slow decoder is used in the following cases: ▪ operand override prefix (66H) preceding an instruction with immediate data ▪ address override prefix (67H) preceding an instruction with a modr/m in real, big real, 16-bit protected or 32-bit protected modes To avoid instruction length decoding stalls, generate code using imm8 or imm32 values instead of imm16 values. If you must use an imm16 value, store the value in a register using “mov reg, imm32” and use the register format of the instruction.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
88H	00H	BR_INST_EXEC	Branch instructions executed	This event counts all executed branches (not necessarily retired). This includes only instructions and not micro-op branches. Frequent branching is not necessarily a major performance issue. However frequent branch mispredictions may be a problem.
89H	00H	BR_MISSP_EXEC	Mispredicted branch instructions executed	This event counts the number of mispredicted branch instructions that were executed.
8AH	00H	BR_BAC_ MISSP_EXEC	Branch instructions mispredicted at decoding	This event counts the number of branch instructions that were mispredicted at decoding.
8BH	00H	BR_CND_EXEC	Conditional branch instructions executed.	This event counts the number of conditional branch instructions executed, but not necessarily retired.
8CH	00H	BR_CND_ MISSP_EXEC	Mispredicted conditional branch instructions executed	This event counts the number of mispredicted conditional branch instructions that were executed.
8DH	00H	BR_IND_EXEC	Indirect branch instructions executed	This event counts the number of indirect branch instructions that were executed.
8EH	00H	BR_IND_MISSP _EXEC	Mispredicted indirect branch instructions executed	This event counts the number of mispredicted indirect branch instructions that were executed.
8FH	00H	BR_RET_EXEC	RET instructions executed	This event counts the number of RET instructions that were executed.
90H	00H	BR_RET_ MISSP_EXEC	Mispredicted RET instructions executed	This event counts the number of mispredicted RET instructions that were executed.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
91H	00H	BR_RET_BAC_ MISSP_EXEC	RET instructions executed mispredicted at decoding	This event counts the number of RET instructions that were executed and were mispredicted at decoding.
92H	00H	BR_CALL_EXEC	CALL instructions executed	This event counts the number of CALL instructions executed
93H	00H	BR_CALL_ MISSP_EXEC	Mispredicted CALL instructions executed	This event counts the number of mispredicted CALL instructions that were executed.
94H	00H	BR_IND_CALL_ EXEC	Indirect CALL instructions executed	This event counts the number of indirect CALL instructions that were executed.
97H	00H	BR_TKN_ BUBBLE_1	Branch predicted taken with bubble 1	The events BR_TKN_BUBBLE_1 and BR_TKN_BUBBLE_2 together count the number of times a taken branch prediction incurred a one-cycle penalty. The penalty incurs when: ▪ Too many taken branches are placed together. To avoid this, unroll loops and add a non-taken branch in the middle of the taken sequence. ▪ The branch target is unaligned. To avoid this, align the branch target.
98H	00H	BR_TKN_ BUBBLE_2	Branch predicted taken with bubble 2	The events BR_TKN_BUBBLE_1 and BR_TKN_BUBBLE_2 together count the number of times a taken branch prediction incurred a one-cycle penalty. The penalty incurs when: ▪ Too many taken branches are placed together. To avoid this, unroll loops and add a non-taken branch in the middle of the taken sequence. ▪ The branch target is unaligned. To avoid this, align the branch target.
A0H	00H	RS_UOPS_ DISPATCHED	Micro-ops dispatched for execution	This event counts the number of micro-ops dispatched for execution. Up to six micro-ops can be dispatched in each cycle.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
A1H	01H	RS_UOPS_ DISPATCHED.PORT 0	Cycles micro-ops dispatched for execution on port 0	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Issue Ports are described in *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Use IA32_PMC0 only.
A1H	02H	RS_UOPS_ DISPATCHED.PORT 1	Cycles micro-ops dispatched for execution on port 1	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.
A1H	04H	RS_UOPS_ DISPATCHED.PORT 2	Cycles micro-ops dispatched for execution on port 2	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.
A1H	08H	RS_UOPS_ DISPATCHED.PORT 3	Cycles micro-ops dispatched for execution on port 3	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.
A1H	10H	RS_UOPS_ DISPATCHED.PORT 4	Cycles micro-ops dispatched for execution on port 4	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.
A1H	20H	RS_UOPS_ DISPATCHED.PORT 5	Cycles micro-ops dispatched for execution on port 5	This event counts the number of cycles for which micro-ops dispatched for execution. Each cycle, at most one micro-op can be dispatched on the port. Use IA32_PMC0 only.
AAH	01H	MACRO_INSTS. DECODED	Instructions decoded	This event counts the number of instructions decoded (but not necessarily executed or retired).
AAH	08H	MACRO_INSTS. CISC_DECODED	CISC Instructions decoded	This event counts the number of complex instructions decoded. Complex instructions usually have more than four micro-ops. Only one complex instruction can be decoded at a time.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
ABH	01H	ESP.SYNCH	ESP register content synchronization	This event counts the number of times that the ESP register is explicitly used in the address expression of a load or store operation, after it is implicitly used, for example by a push or a pop instruction. ESP synch micro-op uses resources from the rename pipe-stage and up to retirement. The expected ratio of this event divided by the number of ESP implicit changes is 0,2. If the ratio is higher, consider rearranging your code to avoid ESP synchronization events.
ABH	02H	ESP.ADDITIONS	ESP register automatic additions	This event counts the number of ESP additions performed automatically by the decoder. A high count of this event is good, since each automatic addition performed by the decoder saves a micro-op from the execution units. To maximize the number of ESP additions performed automatically by the decoder, choose instructions that implicitly use the ESP, such as PUSH, POP, CALL, and RET instructions whenever possible.
B0H	00H	SIMD_UOPS_EXEC	SIMD micro-ops executed (excluding stores)	This event counts all the SIMD micro-ops executed. It does not count MOVQ and MOVD stores from register to memory.
B1H	00H	SIMD_SAT_UOP_EXEC	SIMD saturated arithmetic micro-ops executed	This event counts the number of SIMD saturated arithmetic micro-ops executed.
B3H	01H	SIMD_UOP_TYPE_EXEC.MUL	SIMD packed multiply micro-ops executed	This event counts the number of SIMD packed multiply micro-ops executed.
B3H	02H	SIMD_UOP_TYPE_EXEC.SHIFT	SIMD packed shift micro-ops executed	This event counts the number of SIMD packed shift micro-ops executed.
B3H	04H	SIMD_UOP_TYPE_EXEC.PACK	SIMD pack micro-ops executed	This event counts the number of SIMD pack micro-ops executed.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
B3H	08H	SIMD_UOP_TYPE_EXEC.UNPACK	SIMD unpack micro-ops executed	This event counts the number of SIMD unpack micro-ops executed.
B3H	10H	SIMD_UOP_TYPE_EXEC.LOGICAL	SIMD packed logical micro-ops executed	This event counts the number of SIMD packed logical micro-ops executed.
B3H	20H	SIMD_UOP_TYPE_EXEC.ARITHMETIC	SIMD packed arithmetic micro-ops executed	This event counts the number of SIMD packed arithmetic micro-ops executed.
C0H	00H	INST_RETIRED.ANY_P	Instructions retired	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continue counting during hardware interrupts, traps, and inside interrupt handlers. INST_RETIRED.ANY_P is an architectural performance event.
C0H	01H	INST_RETIRED.LOADS	Instructions retired, which contain a load	This event counts the number of instructions retired that contain a load operation.
C0H	02H	INST_RETIRED.STORES	Instructions retired, which contain a store	This event counts the number of instructions retired that contain a store operation.
C0H	04H	INST_RETIRED.OTHER	Instructions retired, with no load or store operation	This event counts the number of instructions retired that do not contain a load or a store operation.
C1H	01H	X87_OPS_RETIRED.FXCH	FXCH instructions retired	This event counts the number of FXCH instructions retired. Modern compilers generate more efficient code and are less likely to use this instruction. If you obtain a high count for this event consider recompiling the code.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
C1H	FEH	X87_OPS_ RETIRED.ANY	Retired floating-point computational operations (precise event)	This event counts the number of floating-point computational operations retired. It counts: ▪ floating point computational operations executed by the assist handler ▪ sub-operations of complex floating-point instructions like transcendental instructions
				This event does not count: ▪ floating-point computational operations that cause traps or assists. ▪ floating-point loads and stores. When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event.
C2H	01H	UOPS_RETIRED. LD_IND_BR	Fused load+op or load+indirect branch retired	This event counts the number of retired micro-ops that fused a load with another operation. This includes: ▪ Fusion of a load and an arithmetic operation, such as with the following instruction: ADD EAX, [EBX] where the content of the memory location specified by EBX register is loaded, added to EXA register, and the result is stored in EAX. ▪ Fusion of a load and a branch in an indirect branch operation, such as with the following instructions: ▪ JMP [RDI+200] ▪ RET ▪ Fusion decreases the number of micro-ops in the processor pipeline. A high value for this event count indicates that the code is using the processor resources effectively.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
C2H	02H	UOPS_RETIRED.STD_STA	Fused store address + data retired	This event counts the number of store address calculations that are fused with store data emission into one micro-op. Traditionally, each store operation required two micro-ops. This event counts fusion of retired micro-ops only. Fusion decreases the number of micro-ops in the processor pipeline. A high value for this event count indicates that the code is using the processor resources effectively.
C2H	04H	UOPS_RETIRED.MACRO_FUSION	Retired instruction pairs fused into one micro-op	This event counts the number of times CMP or TEST instructions were fused with a conditional branch instruction into one micro-op. It counts fusion by retired micro-ops only. Fusion decreases the number of micro-ops in the processor pipeline. A high value for this event count indicates that the code uses the processor resources more effectively.
C2H	07H	UOPS_RETIRED.FUSED	Fused micro-ops retired	This event counts the total number of retired fused micro-ops. The counts include the following fusion types: ▪ Fusion of load operation with an arithmetic operation or with an indirect branch (counted by event UOPS_RETIRED.LD_IND_BR) ▪ Fusion of store address and data (counted by event UOPS_RETIRED.STD_STA) ▪ Fusion of CMP or TEST instruction with a conditional branch instruction (counted by event UOPS_RETIRED.MACRO_FUSION) Fusion decreases the number of micro-ops in the processor pipeline. A high value for this event count indicates that the code is using the processor resources effectively.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
C2H	08H	UOPS_RETIRED. NON_FUSED	Non-fused micro-ops retired	This event counts the number of micro-ops retired that were not fused.
C2H	0FH	UOPS_RETIRED. ANY	Micro-ops retired	This event counts the number of micro-ops retired. The processor decodes complex macro instructions into a sequence of simpler micro-ops. Most instructions are composed of one or two micro-ops.
				Some instructions are decoded into longer sequences such as repeat instructions, floating point transcendental instructions, and assists. In some cases micro-op sequences are fused or whole instructions are fused into one micro-op. See other UOPS_RETIRED events for differentiating retired fused and non-fused micro-ops.
C3H	01H	MACHINE_ NUKES.SMC	Self-Modifying Code detected	This event counts the number of times that a program writes to a code section. Self-modifying code causes a sever penalty in all Intel 64 and IA-32 processors.
C3H	04H	MACHINE_NUKES. MEM_ORDER	Execution pipeline restart due to memory ordering conflict or memory disambiguation misprediction	This event counts the number of times the pipeline is restarted due to either multi-threaded memory ordering conflicts or memory disambiguation misprediction. A multi-threaded memory ordering conflict occurs when a store, which is executed in another core, hits a load that is executed out of order in this core but not yet retired. As a result, the load needs to be restarted to satisfy the memory ordering model. See Chapter 7, "Multiple-Processor Management" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. To count memory disambiguation mispredictions, use the event MEMORY_DISAMBIGUATION.RESET.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
C4H	00H	BR_INST_RETIRED.ANY	Retired branch instructions	This event counts the number of branch instructions retired. This is an architectural performance event.
C4H	01H	BR_INST_RETIRED.PRED_NOT_TAKEN	Retired branch instructions that were predicted not-taken	This event counts the number of branch instructions retired that were correctly predicted to be not-taken.
C4H	02H	BR_INST_RETIRED.MISPRED_NOT_TAKEN	Retired branch instructions that were mispredicted not-taken	This event counts the number of branch instructions retired that were mispredicted and not-taken.
C4H	04H	BR_INST_RETIRED.PRED_TAKEN	Retired branch instructions that were predicted taken	This event counts the number of branch instructions retired that were correctly predicted to be taken.
C4H	08H	BR_INST_RETIRED.MISPRED_TAKEN	Retired branch instructions that were mispredicted taken	This event counts the number of branch instructions retired that were mispredicted and taken.
C4H	0CH	BR_INST_RETIRED.TAKEN	Retired taken branch instructions	This event counts the number of branches retired that were taken.
C5H	00H	BR_INST_RETIRED.MISPRED	Retired mispredicted branch instructions. (precise event)	This event counts the number of retired branch instructions that were mispredicted by the processor. A branch misprediction occurs when the processor predicts that the branch would be taken, but it is not, or vice-versa. This is an architectural performance event.
C6H	01H	CYCLES_INT_MASKED	Cycles during which interrupts are disabled	This event counts the number of cycles during which interrupts are disabled.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
C6H	02H	CYCLES_INT_ PENDING_AND _MASKED	Cycles during which interrupts are pending and disabled	This event counts the number of cycles during which there are pending interrupts but interrupts are disabled.
C7H	01H	SIMD_INST_ RETIRED.PACKED_ SINGLE	Retired SSE packed-single instructions	This event counts the number of SSE packed-single instructions retired.
C7H	02H	SIMD_INST_ RETIRED.SCALAR_ SINGLE	Retired SSE scalar-single instructions	This event counts the number of SSE scalar-single instructions retired.
C7H	04H	SIMD_INST_ RETIRED.PACKED_ DOUBLE	Retired SSE2 packed-double instructions	This event counts the number of SSE2 packed-double instructions retired.
C7H	08H	SIMD_INST_ RETIRED.SCALAR_ DOUBLE	Retired SSE2 scalar-double instructions	This event counts the number of SSE2 scalar-double instructions retired.
C7H	10H	SIMD_INST_ RETIRED.VECTOR	Retired SSE2 vector integer instructions	This event counts the number of SSE2 vector integer instructions retired.
C7H	1FH	SIMD_INST_ RETIRED.ANY	Retired Streaming SIMD instructions (precise event)	This event counts the overall number of retired SIMD instructions that use XMM registers. To count each type of SIMD instruction separately, use the following events: ▪ SIMD_INST_RETIRED.PACKED_SINGLE ▪ SIMD_INST_RETIRED.SCALAR_SINGLE ▪ SIMD_INST_RETIRED.PACKED_DOUBLE ▪ SIMD_INST_RETIRED.SCALAR_DOUBLE ▪ and SIMD_INST_RETIRED.VECTOR When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event.
C8H	00H	HW_INT_RCV	Hardware interrupts received	This event counts the number of hardware interrupts received by the processor.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
C9H	00H	ITLB_MISS_ RETIRED	Retired instructions that missed the ITLB	This event counts the number of retired instructions that missed the ITLB when they were fetched.
CAH	01H	SIMD_COMP_ INST_RETIRED. PACKED_SINGLE	Retired computational SSE packed-single instructions	This event counts the number of computational SSE packed-single instructions retired. Computational instructions perform arithmetic computations (for example: add, multiply and divide). Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CAH	02H	SIMD_COMP_ INST_RETIRED. SCALAR_SINGLE	Retired computational SSE scalar-single instructions	This event counts the number of computational SSE scalar-single instructions retired. Computational instructions perform arithmetic computations (for example: add, multiply and divide). Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CAH	04H	SIMD_COMP_ INST_RETIRED. PACKED_DOUBLE	Retired computational SSE2 packed-double instructions	This event counts the number of computational SSE2 packed-double instructions retired. Computational instructions perform arithmetic computations (for example: add, multiply and divide). Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
CAH	08H	SIMD_COMP_INST_RETIRED.SCALAR_DOUBLE	Retired computational SSE2 scalar-double instructions	This event counts the number of computational SSE2 scalar-double instructions retired. Computational instructions perform arithmetic computations (for example: add, multiply and divide). Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CBH	01H	MEM_LOAD_RETIRED.L1D_MISS	Retired loads that miss the L1 data cache (precise event)	This event counts the number of retired load operations that missed the L1 data cache. This includes loads from cache lines that are currently being fetched, due to a previous L1 data cache miss to the same cache line. This event counts loads from cacheable memory only. The event does not count loads by software prefetches. When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event. Use IA32_PMC0 only.
CBH	02H	MEM_LOAD_RETIRED.L1D_LINE_MISS	L1 data cache line missed by retired loads (precise event)	This event counts the number of load operations that miss the L1 data cache and send a request to the L2 cache to fetch the missing cache line. That is the missing cache line fetching has not yet started. The event count is equal to the number of cache lines fetched from the L2 cache by retired loads. This event counts loads from cacheable memory only. The event does not count loads by software prefetches. The event might not be counted if the load is blocked (see LOAD_BLOCK events).

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
				When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event. Use IA32_PMC0 only.
CBH	04H	MEM_LOAD_ RETIRED.L2_MISS	Retired loads that miss the L2 cache (precise event)	This event counts the number of retired load operations that missed the L2 cache. This event counts loads from cacheable memory only. It does not count loads by software prefetches. When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event. Use IA32_PMC0 only.
CBH	08H	MEM_LOAD_ RETIRED.L2_LINE_ MISS	L2 cache line missed by retired loads (precise event)	This event counts the number of load operations that miss the L2 cache and result in a bus request to fetch the missing cache line. That is the missing cache line fetching has not yet started. This event count is equal to the number of cache lines fetched from memory by retired loads. This event counts loads from cacheable memory only. The event does not count loads by software prefetches. The event might not be counted if the load is blocked (see LOAD_BLOCK events). When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event. Use IA32_PMC0 only.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
CBH	10H	MEM_LOAD_ RETIRED.DTLB_ MISS	Retired loads that miss the DTLB (precise event)	This event counts the number of retired loads that missed the DTLB. The DTLB miss is not counted if the load operation causes a fault. This event counts loads from cacheable memory only. The event does not count loads by software prefetches. When this event is captured with the precise event mechanism, the collected samples contain the address of the instruction that was executed immediately after the instruction that caused the event. Use IA32_PMC0 only.
CCH	01H	FP_MMX_TRANS_ TO_MMX	Transitions from Floating Point to MMX Instructions	This event counts the first MMX instructions following a floating-point instruction. Use this event to estimate the penalties for the transitions between floating-point and MMX states.
CCH	02H	FP_MMX_TRANS_ TO_FP	Transitions from MMX Instructions to Floating Point Instructions	This event counts the first floating-point instructions following any MMX instruction. Use this event to estimate the penalties for the transitions between floating-point and MMX states.
CDH	00H	SIMD_ASSIST	SIMD assists invoked	This event counts the number of SIMD assists invoked. SIMD assists are invoked when an EMMS instruction is executed, changing the MMX state in the floating point stack.
CEH	00H	SIMD_INSTR_ RETIRED	SIMD Instructions retired	This event counts the number of retired SIMD instructions that use MMX registers.
CFH	00H	SIMD_SAT_INSTR_ RETIRED	Saturated arithmetic instructions retired	This event counts the number of saturated arithmetic SIMD instructions that retired.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
D2H	01H	RAT_STALLS. ROB_READ_PORT	ROB read port stalls cycles	This event counts the number of cycles when ROB read port stalls occurred, which did not allow new micro-ops to enter the out-of-order pipeline. Note that, at this stage in the pipeline, additional stalls may occur at the same cycle and prevent the stalled micro-ops from entering the pipe. In such a case, micro-ops retry entering the execution pipe in the next cycle and the ROB-read-port stall is counted again.
D2H	02H	RAT_STALLS. PARTIAL_CYCLES	Partial register stall cycles	This event counts the number of cycles instruction execution latency became longer than the defined latency because the instruction uses a register that was partially written by previous instructions.
D2H	04H	RAT_STALLS. FLAGS	Flag stall cycles	This event counts the number of cycles during which execution stalled due to several reasons, one of which is a partial flag register stall. A partial register stall may occur when two conditions are met: ▪ an instruction modifies some, but not all, of the flags in the flag register ▪ the next instruction, which depends on flags, depends on flags that were not modified by this instruction
D2H	08H	RAT_STALLS. FPSW	FPU status word stall	This event indicates that the FPU status word (FPSW) is written. To obtain the number of times the FPSW is written divide the event count by 2. The FPSW is written by instructions with long latency; a small count may indicate a high penalty.
D2H	0FH	RAT_STALLS. ANY	All RAT stall cycles	This event counts the number of stall cycles due to conditions described by: ▪ RAT_STALLS.ROB_READ_PORT ▪ RAT_STALLS.PARTIAL ▪ RAT_STALLS.FLAGS ▪ RAT_STALLS.FPSW.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
D4H	01H	SEG_RENAME_ STALLS.ES	Segment rename stalls - ES	This event counts the number of stalls due to the lack of renaming resources for the ES segment register. If a segment is renamed, but not retired and a second update to the same segment occurs, a stall occurs in the front-end of the pipeline until the renamed segment retires.
D4H	02H	SEG_RENAME_ STALLS.DS	Segment rename stalls - DS	This event counts the number of stalls due to the lack of renaming resources for the DS segment register. If a segment is renamed, but not retired and a second update to the same segment occurs, a stall occurs in the front-end of the pipeline until the renamed segment retires.
D4H	04H	SEG_RENAME_ STALLS.FS	Segment rename stalls - FS	This event counts the number of stalls due to the lack of renaming resources for the FS segment register. If a segment is renamed, but not retired and a second update to the same segment occurs, a stall occurs in the front-end of the pipeline until the renamed segment retires.
D4H	08H	SEG_RENAME_ STALLS.GS	Segment rename stalls - GS	This event counts the number of stalls due to the lack of renaming resources for the GS segment register. If a segment is renamed, but not retired and a second update to the same segment occurs, a stall occurs in the front-end of the pipeline until the renamed segment retires.
D4H	0FH	SEG_RENAME_ STALLS.ANY	Any (ES/DS/FS/GS) segment rename stall	This event counts the number of stalls due to the lack of renaming resources for the ES, DS, FS, and GS segment registers. If a segment is renamed but not retired and a second update to the same segment occurs, a stall occurs in the front-end of the pipeline until the renamed segment retires.
D5H	01H	SEG_REG_ RENAMES.ES	Segment renames - ES	This event counts the number of times the ES segment register is renamed.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
D5H	02H	SEG_REG_ RENAMES.DS	Segment renames - DS	This event counts the number of times the DS segment register is renamed.
D5H	04H	SEG_REG_ RENAMES.FS	Segment renames - FS	This event counts the number of times the FS segment register is renamed.
D5H	08H	SEG_REG_ RENAMES.GS	Segment renames - GS	This event counts the number of times the GS segment register is renamed.
D5H	0FH	SEG_REG_ RENAMES.ANY	Any (ES/DS/FS/GS) segment rename	This event counts the number of times any of the four segment registers (ES/DS/FS/GS) is renamed.
DCH	01H	RESOURCE_ STALLS.ROB_FULL	Cycles during which the ROB full	This event counts the number of cycles when the number of instructions in the pipeline waiting for retirement reaches the limit the processor can handle. A high count for this event indicates that there are long latency operations in the pipe (possibly load and store operations that miss the L2 cache, and other instructions that depend on these cannot execute until the former instructions complete execution). In this situation new instructions can not enter the pipe and start execution.
DCH	02H	RESOURCE_ STALLS.RS_FULL	Cycles during which the RS full	This event counts the number of cycles when the number of instructions in the pipeline waiting for execution reaches the limit the processor can handle. A high count of this event indicates that there are long latency operations in the pipe (possibly load and store operations that miss the L2 cache, and other instructions that depend on these cannot execute until the former instructions complete execution). In this situation new instructions can not enter the pipe and start execution.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
DCH	04	RESOURCE_ STALLS.LD_ST	Cycles during which the pipeline has exceeded load or store limit or waiting to commit all stores	This event counts the number of cycles while resource-related stalls occur due to: ▪ The number of load instructions in the pipeline reached the limit the processor can handle. The stall ends when a loading instruction retires. ▪ The number of store instructions in the pipeline reached the limit the processor can handle. The stall ends when a storing instruction commits its data to the cache or memory. ▪ There is an instruction in the pipe that can be executed only when all previous stores complete and their data is committed in the caches or memory. For example, the SFENCE and MFENCE instructions require this behavior.
DCH	08H	RESOURCE_ STALLS.FPCW	Cycles stalled due to FPU control word write	This event counts the number of cycles while execution was stalled due to writing the floating-point unit (FPU) control word.
DCH	10H	RESOURCE_ STALLS.BR_MISS_C LEAR	Cycles stalled due to branch misprediction	This event counts the number of cycles after a branch misprediction is detected at execution until the branch and all older micro-ops retire. During this time new micro-ops cannot enter the out-of-order pipeline.
DCH	1FH	RESOURCE_ STALLS.ANY	Resource related stalls	This event counts the number of cycles while resource-related stalls occurs for any conditions described by the following events: ▪ RESOURCE_STALLS.ROB_FULL ▪ RESOURCE_STALLS.RS_FULL ▪ RESOURCE_STALLS.LD_ST ▪ RESOURCE_STALLS.FPCW ▪ RESOURCE_STALLS.BR_MISS_CLEAR
E0H	00H	BR_INST_ DECODED	Branch instructions decoded	This event counts the number of branch instructions decoded.

Table A-4. Non-Architectural Performance Events in Processors Based on Intel Core Microarchitecture (Contd.)

Event Num	Umask Value	Event Name	Definition	Description and Comment
E4H	00H	BOGUS_BR	Bogus branches	This event counts the number of byte sequences that were mistakenly detected as taken branch instructions. This results in a BACLEAR event. This occurs mainly after task switches.
E6H	00H	BACLEARS	BACLEARS asserted	This event counts the number of times the front end is resteered, mainly when the BPU cannot provide a correct prediction and this is corrected by other branch handling mechanisms at the front and. This can occur if the code has many branches such that they cannot be consumed by the BPU. Each BACLEAR asserted costs approximately 7 cycles of instruction fetch. The effect on total execution time depends on the surrounding code.
F0	00H	PREF_RQSTS_UP	Upward prefetches issued from DPL	This event counts the number of upward prefetches issued from the Data Prefetch Logic (DPL) to the L2 cache. A prefetch request issued to the L2 cache cannot be cancelled and the requested cache line is fetched to the L2 cache.
F8	00H	PREF_RQSTS_DN	Downward prefetches issued from DPL.	This event counts the number of downward prefetches issued from the Data Prefetch Logic (DPL) to the L2 cache. A prefetch request issued to the L2 cache cannot be cancelled and the requested cache line is fetched to the L2 cache.

A.4 PERFORMANCE MONITORING EVENTS FOR INTEL® ATOM™ PROCESSORS

Processors based on the Intel Atom microarchitecture support the architectural and non-architectural performance-monitoring events listed in Table A-1 and Table A-4. In addition, they also support the following non-architectural performance-monitoring events listed in Table A-5.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
02H	81H	STORe_FORWARDS.GOOD	Good store forwards	This event counts the number of times store data was forwarded directly to a load.
06H	00H	SEGMENT_REG_LOADS.ANY	Number of segment register loads	This event counts the number of segment register load operations. Instructions that load new values into segment registers cause a penalty. This event indicates performance issues in 16-bit code. If this event occurs frequently, it may be useful to calculate the number of instructions retired per segment register load. If the resulting calculation is low (on average a small number of instructions are executed between segment register loads), then the code's segment register usage should be optimized. As a result of branch misprediction, this event is speculative and may include segment register loads that do not actually occur. However, most segment register loads are internally serialized and such speculative effects are minimized.
07H	01H	PREFETCH.PREFETCHT0	Streaming SIMD Extensions (SSE) PrefetchT0 instructions executed.	This event counts the number of times the SSE instruction prefetchT0 is executed. This instruction prefetches the data to the L1 data cache and L2 cache.
07H	06H	PREFETCH.SW_L2	Streaming SIMD Extensions (SSE) PrefetchT1 and PrefetchT2 instructions executed	This event counts the number of times the SSE instructions prefetchT1 and prefetchT2 are executed. These instructions prefetch the data to the L2 cache

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
07H	08H	PREFETCH.PREFETCHNTA	Streaming SIMD Extensions (SSE) Prefetch NTA instructions executed	This event counts the number of times the SSE instruction prefetchNTA is executed. This instruction prefetches the data to the L1 data cache.
08H	07H	DATA_TLB_MISSES.DTLB_MISS	Memory accesses that missed the DTLB	This event counts the number of Data Table Lookaside Buffer (DTLB) misses. The count includes misses detected as a result of speculative accesses. Typically a high count for this event indicates that the code accesses a large number of data pages.
08H	05H	DATA_TLB_MISSES.DTLB_MISS_LD	DTLB misses due to load operations	This event counts the number of Data Table Lookaside Buffer (DTLB) misses due to load operations. This count includes misses detected as a result of speculative accesses.
08H	09H	DATA_TLB_MISSES.UTLB_MISS_LD	UTLB misses due to load operations	This event counts the number of UTLB misses due to load operations. This count includes misses detected as a result of speculative accesses.
08H	06H	DATA_TLB_MISSES.DTLB_MISS_ST	DTLB misses due to store operations	This event counts the number of Data Table Lookaside Buffer (DTLB) misses due to store operations. This count includes misses detected as a result of speculative accesses.
0CH	03H	PAGE_WALKS.WALKS	Number of page-walks executed	This event counts the number of page-walks executed due to either a DTLB or ITLB miss. The page walk duration, PAGE_WALKS.CYCLES, divided by number of page walks is the average duration of a page walk. This can hint to whether most of the page-walks are satisfied by the caches or cause an L2 cache miss.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
0CH	03H	PAGE_WALKS.CYCLES	Duration of page-walks in core cycles	This event counts the duration of page-walks in core cycles. The paging mode in use typically affects the duration of page walks. Page walk duration divided by number of page walks is the average duration of page-walks. This can hint at whether most of the page-walks are satisfied by the caches or cause an L2 cache miss. (NOTE: Number of occurrences can be collected using edge detection)
10H	01H	X87_COMP_OPS_EXE.ANY.S	Floating point computational micro-ops executed	This event counts the number of x87 floating point computational micro-ops executed.
10H	81H	X87_COMP_OPS_EXE.ANY.AR	Floating point computational micro-ops retired	This event counts the number of x87 floating point computational micro-ops retired.
11H	01H	FP_ASSIST	Floating point assists	This event counts the number of floating point operations executed that required micro-code assist intervention. These assists are required in the following cases: X87 instructions: 1. NaN or denormal are loaded to a register or used as input from memory 2. Division by 0 3. Underflow output
11H	81H	FP_ASSIST.AR	Floating point assists	This event counts the number of floating point operations executed that required micro-code assist intervention. These assists are required in the following cases: X87 instructions: 1. NaN or denormal are loaded to a register or used as input from memory 2. Division by 0 3. Underflow output
12H	01H	MUL.S	Multiply operations executed	This event counts the number of multiply operations executed. This includes integer as well as floating point multiply operations.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
12H	81H	MUL.AR	Multiply operations retired	This event counts the number of multiply operations retired. This includes integer as well as floating point multiply operations.
13H	01H	DIV.S	Divide operations executed	This event counts the number of divide operations executed. This includes integer divides, floating point divides and square-root operations executed.
13H	81H	DIV.AR	Divide operations retired	This event counts the number of divide operations retired. This includes integer divides, floating point divides and square-root operations executed.
14H	01H	CYCLES_DIV_BUSY	Cycles the driver is busy	This event counts the number of cycles the divider is busy executing divide or square root operations. The divide can be integer, X87 or Streaming SIMD Extensions (SSE). The square root operation can be either X87 or SSE.
21H	See Table 18-8	L2_ADS	Cycles L2 address bus is in use	This event counts the number of cycles the L2 address bus is being used for accesses to the L2 cache or bus queue. This event can count occurrences for this core or both cores.
22H	See Table 18-8	L2_DBUS_BUSY	Cycles the L2 cache data bus is busy	This event counts core cycles during which the L2 cache data bus is busy transferring data from the L2 cache to the core. It counts for all L1 cache misses (data and instruction) that hit the L2 cache. The count will increment by two for a full cache-line request.
24H	See Table 18-8 and Table 18-10	L2_LINES_IN	L2 cache misses	This event counts the number of cache lines allocated in the L2 cache. Cache lines are allocated in the L2 cache as a result of requests from the L1 data and instruction caches and the L2 hardware prefetchers to cache lines that are missing in the L2 cache. This event can count occurrences for this core or both cores. This event can also count demand requests and L2 hardware prefetch requests together or separately.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
25H	See Table 18-8	L2_M_LINES_IN	L2 cache line modifications	This event counts whenever a modified cache line is written back from the L1 data cache to the L2 cache. This event can count occurrences for this core or both cores.
26H	See Table 18-8 and Table 18-10	L2_LINES_OUT	L2 cache lines evicted	This event counts the number of L2 cache lines evicted. This event can count occurrences for this core or both cores. This event can also count evictions due to demand requests and L2 hardware prefetch requests together or separately.
27H	See Table 18-8 and Table 18-10	L2_M_LINES_OUT	Modified lines evicted from the L2 cache	This event counts the number of L2 modified cache lines evicted. These lines are written back to memory unless they also exist in a shared-state in one of the L1 data caches. This event can count occurrences for this core or both cores. This event can also count evictions due to demand requests and L2 hardware prefetch requests together or separately.
28H	See Table 18-8 and Table 18-11	L2_IFETCH	L2 cacheable instruction fetch requests	This event counts the number of instruction cache line requests from the ICache. It does not include fetch requests from uncacheable memory. It does not include ITLB miss accesses. This event can count occurrences for this core or both cores. This event can also count accesses to cache lines at different MESI states.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
29H	See Table 18-8, Table 18-10 and Table 18-11	L2_LD	L2 cache reads	This event counts L2 cache read requests coming from the L1 data cache and L2 prefetchers. This event can count occurrences for this core or both cores. This event can count occurrences - for this core or both cores. - due to demand requests and L2 hardware prefetch requests together or separately. - of accesses to cache lines at different MESI states.
2AH	See Table 18-8 and Table 18-11	L2_ST	L2 store requests	This event counts all store operations that miss the L1 data cache and request the data from the L2 cache. This event can count occurrences for this core or both cores. This event can also count accesses to cache lines at different MESI states.
2BH	See Table 18-8 and Table 18-11	L2_LOCK	L2 locked accesses	This event counts all locked accesses to cache lines that miss the L1 data cache. This event can count occurrences for this core or both cores. This event can also count accesses to cache lines at different MESI states.
2EH	See Table 18-8, Table 18-10 and Table 18-11	L2_RQSTS	L2 cache requests	This event counts all completed L2 cache requests. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, instruction fetches, and all L2 hardware prefetch requests. This event can count occurrences - for this core or both cores. - due to demand requests and L2 hardware prefetch requests together, or separately. - of accesses to cache lines at different MESI states.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
2EH	41H	L2_RQSTS.SELF.DEMAND.I_STATE	L2 cache demand requests from this core that missed the L2	This event counts all completed L2 cache demand requests from this core that miss the L2 cache. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, and instruction fetches. This is an architectural performance event.
2EH	4FH	L2_RQSTS.SELF.DEMAND.MESI	L2 cache demand requests from this core	This event counts all completed L2 cache demand requests from this core. This includes L1 data cache reads, writes, and locked accesses, L1 data prefetch requests, and instruction fetches. This is an architectural performance event.
30H	See Table 18-8, Table 18-10 and Table 18-11	L2_REJECT_BUSQ	Rejected L2 cache requests	This event indicates that a pending L2 cache request that requires a bus transaction is delayed from moving to the bus queue. Some of the reasons for this event are: - The bus queue is full. - The bus queue already holds an entry for a cache line in the same set. The number of events is greater or equal to the number of requests that were rejected. - for this core or both cores. - due to demand requests and L2 hardware prefetch requests together, or separately. - of accesses to cache lines at different MESI states.
32H	See Table 18-8	L2_NO_REQ	Cycles no L2 cache requests are pending	This event counts the number of cycles that no L2 cache requests are pending.
3AH	00H	EIST_TRANS	Number of Enhanced Intel SpeedStep(R) Technology (EIST) transitions	This event counts the number of Enhanced Intel SpeedStep(R) Technology (EIST) transitions that include a frequency change, either with or without VID change. This event is incremented only while the counting core is in C0 state. Since the CxE states include an EIST transition, the event will be incremented accordingly.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
				EIST transitions are commonly initiated by OS, but can be initiated by HW internally. For example: CxE states are C-states (C1,C2,C3...) which not only place the CPU into a sleep state by turning off the clock and other components, but also lower the voltage (which reduces the leakage power consumption). The same is true for thermal throttling transition which uses EIST internally.
3BH	C0H	THERMAL_TRIP	Number of thermal trips	This event counts the number of thermal trips. A thermal trip occurs whenever the processor temperature exceeds the thermal trip threshold temperature. Following a thermal trip, the processor automatically reduces frequency and voltage. The processor checks the temperature every millisecond, and returns to normal when the temperature falls below the thermal trip threshold temperature.
3CH	00H	CPU_CLK_UNHALTED.CORE_P	Core cycles when core is not halted	This event counts the number of core cycles while the core is not in a halt state. The core enters the halt state when it is running the HLT instruction. This event is a component in many key event ratios. In mobile systems the core frequency may change from time to time. For this reason this event may have a changing ratio with regards to time. In systems with a constant core frequency, this event can give you a measurement of the elapsed time while the core was not in halt state by dividing the event count by the core frequency. -This is an architectural performance event. - The event CPU_CLK_UNHALTED.CORE_P is counted by a programmable counter. - The event CPU_CLK_UNHALTED.CORE is counted by a designated fixed counter, leaving the two programmable counters available for other events

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
3CH	01H	CPU_CLK_UNHALTED.BUS	Bus cycles when core is not halted	This event counts the number of bus cycles while the core is not in the halt state. This event can give you a measurement of the elapsed time while the core was not in the halt state, by dividing the event count by the bus frequency. The core enters the halt state when it is running the HLT instruction. The event also has a constant ratio with CPU_CLK_UNHALTED.REF event, which is the maximum bus to processor frequency ratio. Non-halted bus cycles are a component in many key event ratios.
3CH	02H	CPU_CLK_UNHALTED.NO_OTHER	Bus cycles when core is active and the other is halted	This event counts the number of bus cycles during which the core remains non-halted, and the other core on the processor is halted. This event can be used to determine the amount of parallelism exploited by an application or a system. Divide this event count by the bus frequency to determine the amount of time that only one core was in use.
40H	21H	L1D_CACHE.LD	L1 Cacheable Data Reads	This event counts the number of data reads from cacheable memory.
40H	22H	L1D_CACHE.ST	L1 Cacheable Data Writes	This event counts the number of data writes to cacheable memory
60H	See Table 18-8 and Table 18-9	BUS_REQUEST_OUTSTANDING	Outstanding cacheable data read bus requests duration	This event counts the number of pending full cache line read transactions on the bus occurring in each cycle. A read transaction is pending from the cycle it is sent on the bus until the full cache line is received by the processor. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
61H	See Table 18-9	BUS_BNR_DRV	Number of Bus Not Ready signals asserted	This event counts the number of Bus Not Ready (BNR) signals that the processor asserts on the bus to suspend additional bus requests by other bus agents. A bus agent asserts the BNR signal when the number of data and snoop transactions is close to the maximum that the bus can handle. While this signal is asserted, new transactions cannot be submitted on the bus. As a result, transaction latency may have higher impact on program performance. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled
62H	See Table 18-9	BUS_DRDY_CLOCKS	Bus cycles when data is sent on the bus	This event counts the number of bus cycles during which the DRDY (Data Ready) signal is asserted on the bus. The DRDY signal is asserted when data is sent on the bus. This event counts the number of bus cycles during which this agent (the processor) writes data on the bus back to memory or to other bus agents. This includes all explicit and implicit data writebacks, as well as partial writes. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.
63H	See Table 18-8 and Table 18-9	BUS_LOCK_CLOCKS	Bus cycles when a LOCK signal is asserted.	This event counts the number of bus cycles, during which the LOCK signal is asserted on the bus. A LOCK signal is asserted when there is a locked memory access, due to: - Uncacheable memory - Locked operation that spans two cache lines - Page-walk from an uncacheable page table. Bus locks have a very high performance penalty and it is highly recommended to avoid such accesses. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
64H	See Table 18-8	BUS_DATA_RCV	Bus cycles while processor receives data	This event counts the number of cycles during which the processor is busy receiving data. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled
65H	See Table 18-8 and Table 18-9	BUS_TRANS_BRD	Burst read bus transactions	This event counts the number of burst read transactions including: - L1 data cache read misses (and L1 data cache hardware prefetches) - L2 hardware prefetches by the DPL and L2 streamer - IFU read misses of cacheable lines. It does not include RFO transactions.
66H	See Table 18-8 and Table 18-9	BUS_TRANS_RFO	RFO bus transactions	This event counts the number of Read For Ownership (RFO) bus transactions, due to store operations that miss the L1 data cache and the L2 cache. This event also counts RFO bus transactions due to locked operations.
67H	See Table 18-8 and Table 18-9	BUS_TRANS_WB	Explicit writeback bus transactions	This event counts all explicit writeback bus transactions due to dirty line evictions. It does not count implicit writebacks due to invalidation by a snoop request.
68H	See Table 18-8 and Table 18-9	BUS_TRANS_IFETCH	Instruction-fetch bus transactions.	This event counts all instruction fetch full cache line bus transactions.
69H	See Table 18-8 and Table 18-9	BUS_TRANS_INVAL	Invalidate bus transactions	This event counts all invalidate transactions. Invalidate transactions are generated when: - A store operation hits a shared line in the L2 cache. - A full cache line write misses the L2 cache or hits a shared line in the L2 cache.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
6AH	See Table 18-8 and Table 18-9	BUS_TRANS_PWR	Partial write bus transaction.	This event counts partial write bus transactions.
6BH	See Table 18-8 and Table 18-9	BUS_TRANS_P	Partial bus transactions	This event counts all (read and write) partial bus transactions.
6CH	See Table 18-8 and Table 18-9	BUS_TRANS_IO	IO bus transactions	This event counts the number of completed I/O bus transactions as a result of IN and OUT instructions. The count does not include memory mapped IO.
6DH	See Table 18-8 and Table 18-9	BUS_TRANS_DEF	Deferred bus transactions	This event counts the number of deferred transactions.
6EH	See Table 18-8 and Table 18-9	BUS_TRANS_BURST	Burst (full cache-line) bus transactions.	This event counts burst (full cache line) transactions including: - Burst reads - RFOs - Explicit writebacks - Write combine lines
6FH	See Table 18-8 and Table 18-9	BUS_TRANS_MEM	Memory bus transactions	This event counts all memory bus transactions including: - burst transactions - partial reads and writes - invalidate transactions The BUS_TRANS_MEM count is the sum of BUS_TRANS_BURST, BUS_TRANS_P and BUS_TRANS_INVAL.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
70H	See Table 18-8 and Table 18-9	BUS_TRANS_ANY	All bus transactions	This event counts all bus transactions. This includes: - Memory transactions - IO transactions (non memory-mapped) - Deferred transaction completion - Other less frequent transactions, such as interrupts
77H	See Table 18-8 and Table 18-11	EXT_SNOOP	External snoops	This event counts the snoop responses to bus transactions. Responses can be counted separately by type and by bus agent. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled.
7AH	See Table 18-9	BUS_HIT_DRV	HIT signal asserted	This event counts the number of bus cycles during which the processor drives the HIT# pin to signal HIT snoop response. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled
7BH	See Table 18-9	BUS_HITM_DRV	HITM signal asserted	This event counts the number of bus cycles during which the processor drives the HITM# pin to signal HITM snoop response. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled
7DH	See Table 18-8	BUSQ_EMPTY	Bus queue is empty	This event counts the number of cycles during which the core did not have any pending transactions in the bus queue. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled
7EH	See Table 18-8 and Table 18-9	SNOOP_STALL_DRV	Bus stalled for snoops	This event counts the number of times that the bus snoop stall signal is asserted. During the snoop stall cycles no new bus transactions requiring a snoop response can be initiated on the bus. NOTE: This event is thread-independent and will not provide a count per logical processor when AnyThr is disabled

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
7FH	See Table 18-8	BUS_IO_WAIT	IO requests waiting in the bus queue	This event counts the number of core cycles during which IO requests wait in the bus queue. This event counts IO requests from the core.
80H	03H	ICACHE.ACCESSES	Instruction fetches	This event counts all instruction fetches, including uncacheable fetches.
80H	02H	ICACHE.MISSES	Icache miss	This event counts all instruction fetches that miss the Instruction cache or produce memory requests. This includes uncacheable fetches. An instruction fetch miss is counted only once and not once for every cycle it is outstanding
82H	04H	ITLB.FLUSH	ITLB flushes	This event counts the number of ITLB flushes.
82H	02H	ITLB.MISSES	ITLB misses	This event counts the number of instruction fetches that miss the ITLB.
AAH	02H	MACRO_INSTS.CISC_DECODED	CISC macro instructions decoded	This event counts the number of complex instructions decoded, but not necessarily executed or retired. Only one complex instruction can be decoded at a time
AAH	03H	MACRO_INSTS.ALL_DECODED	All Instructions decoded	This event counts the number of instructions decoded.
B0H	00H	SIMD_UOPS_EXEC.S	SIMD micro-ops executed (excluding stores)	This event counts all the SIMD micro-ops executed. This event does not count MOVQ and MOVD stores from register to memory.
B0H	80H	SIMD_UOPS_EXEC.AR	SIMD micro-ops retired (excluding stores)	This event counts the number of SIMD saturated arithmetic micro-ops executed.
B1H	00H	SIMD_SAT_UOP_EXEC.S	SIMD saturated arithmetic micro-ops executed	This event counts the number of SIMD saturated arithmetic micro-ops executed.
B1H	80H	SIMD_SAT_UOP_EXEC.AR	SIMD saturated arithmetic micro-ops retired	This event counts the number of SIMD saturated arithmetic micro-ops retired.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
B3H	01H	SIMD_UOP_TYPE_EXEC.MUL.S	SIMD packed multiply micro-ops executed	This event counts the number of SIMD packed multiply micro-ops executed.
B3H	81H	SIMD_UOP_TYPE_EXEC.MUL.AR	SIMD packed multiply micro-ops retired	This event counts the number of SIMD packed multiply micro-ops retired.
B3H	02H	SIMD_UOP_TYPE_EXEC.SHIFT.S	SIMD packed shift micro-ops executed	This event counts the number of SIMD packed shift micro-ops executed.
B3H	82H	SIMD_UOP_TYPE_EXEC.SHIFT.AR	SIMD packed shift micro-ops retired	This event counts the number of SIMD packed shift micro-ops retired.
B3H	04H	SIMD_UOP_TYPE_EXEC.PACK.S	SIMD pack micro-ops executed	This event counts the number of SIMD pack micro-ops executed.
B3H	84H	SIMD_UOP_TYPE_EXEC.PACK.AR	SIMD pack micro-ops retired	This event counts the number of SIMD pack micro-ops retired.
B3H	08H	SIMD_UOP_TYPE_EXEC.UNPACK.S	SIMD unpack micro-ops executed	This event counts the number of SIMD unpack micro-ops executed.
B3H	88H	SIMD_UOP_TYPE_EXEC.UNPACK.AR	SIMD unpack micro-ops retired	This event counts the number of SIMD unpack micro-ops retired.
B3H	10H	SIMD_UOP_TYPE_EXEC.LOGICAL.S	SIMD packed logical micro-ops executed	This event counts the number of SIMD packed logical micro-ops executed.
B3H	90H	SIMD_UOP_TYPE_EXEC.LOGICAL.AR	SIMD packed logical micro-ops retired	This event counts the number of SIMD packed logical micro-ops retired.
B3H	20H	SIMD_UOP_TYPE_EXEC.ARITHMETIC.S	SIMD packed arithmetic micro-ops executed	This event counts the number of SIMD packed arithmetic micro-ops executed.
B3H	A0H	SIMD_UOP_TYPE_EXEC.ARITHMETIC.AR	SIMD packed arithmetic micro-ops retired	This event counts the number of SIMD packed arithmetic micro-ops retired.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
C0H	00H	INST_RETIRED.ANY_P	Instructions retired (precise event).	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers.
N/A	00H	INST_RETIRED.ANY	Instructions retired	This event counts the number of instructions that retire execution. For instructions that consist of multiple micro-ops, this event counts the retirement of the last micro-op of the instruction. The counter continues counting during hardware interrupts, traps, and inside interrupt handlers.
C2H	10H	UOPS_RETIRED.ANY	Micro-ops retired	This event counts the number of micro-ops retired. The processor decodes complex macro instructions into a sequence of simpler micro-ops. Most instructions are composed of one or two micro-ops. Some instructions are decoded into longer sequences such as repeat instructions, floating point transcendental instructions, and assists. In some cases micro-op sequences are fused or whole instructions are fused into one micro-op. See other UOPS_RETIRED events for differentiating retired fused and non-fused micro-ops.
C3H	01H	MACHINE_CLEARS.SMC	Self-Modifying Code detected	This event counts the number of times that a program writes to a code section. Self-modifying code causes a severe penalty in all Intel® architecture processors.
C4H	00H	BR_INST_RETIRED.ANY	Retired branch instructions	This event counts the number of branch instructions retired. **This is an architectural performance event.**
C4H	01H	BR_INST_RETIRED.PRED_NOT_TAKEN	Retired branch instructions that were predicted not-taken	This event counts the number of branch instructions retired that were correctly predicted to be not-taken.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
C4H	02H	BR_INST_RETIRED.MISPRED_NOT_TAKEN	Retired branch instructions that were mispredicted not-taken	This event counts the number of branch instructions retired that were mispredicted and not-taken.
C4H	04H	BR_INST_RETIRED.PRED_TAKEN	Retired branch instructions that were predicted taken	This event counts the number of branch instructions retired that were correctly predicted to be taken.
C4H	08H	BR_INST_RETIRED.MISPRED_TAKEN	Retired branch instructions that were mispredicted taken	This event counts the number of branch instructions retired that were mispredicted and taken.
C4H	0AH	BR_INST_RETIRED.MISPRED	Retired mispredicted branch instructions (precise event)	This event counts the number of retired branch instructions that were mispredicted by the processor. A branch misprediction occurs when the processor predicts that the branch would be taken, but it is not, or vice-versa. Mispredicted branches degrade the performance because the processor starts executing instructions along a wrong path it predicts. When the misprediction is discovered, all the instructions executed in the wrong path must be discarded, and the processor must start again on the correct path. Using the Profile-Guided Optimization (PGO) features of the Intel® C++ compiler may help reduce branch mispredictions. See the compiler documentation for more information on this feature.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
				To determine the branch misprediction ratio, divide the BR_INST_RETIRED.MISPRED event count by the number of BR_INST_RETIRED.ANY event count. To determine the number of mispredicted branches per instruction, divide the number of mispredicted branches by the INST_RETIRED.ANY event count. To measure the impact of the branch mispredictions use the event RESOURCE_STALLS.BR_MISS_CLEAR. **Tips** - See the optimization guide for tips on reducing branch mispredictions. - PGO's purpose is to have straight line code for the most frequent execution paths, reducing branches taken and increasing the "basic block" size, possibly also reducing the code footprint or working-set.
C4H	0CH	BR_INST_RETIRED.TAKEN	Retired taken branch instructions	This event counts the number of branches retired that were taken.
C4H	0FH	BR_INST_RETIRED.ANY1	Retired branch instructions	This event counts the number of branch instructions retired that were mispredicted. This event is a duplicate of BR_INST_RETIRED.MISPRED.
C5H	00H	BR_INST_RETIRED.MISPRED	Retired mispredicted branch instructions (precise event).	This event counts the number of retired branch instructions that were mispredicted by the processor. A branch misprediction occurs when the processor predicts that the branch would be taken, but it is not, or vice-versa. Mispredicted branches degrade the performance because the processor starts executing instructions along a wrong path it predicts. When the misprediction is discovered, all the instructions executed in the wrong path must be discarded, and the processor must start again on the correct path.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
				Using the Profile-Guided Optimization (PGO) features of the Intel® C++ compiler may help reduce branch mispredictions. See the compiler documentation for more information on this feature. To determine the branch misprediction ratio, divide the BR_INST_RETIRED.MISPRED event count by the number of BR_INST_RETIRED.ANY event count. To determine the number of mispredicted branches per instruction, divide the number of mispredicted branches by the INST_RETIRED.ANY event count. To measure the impact of the branch mispredictions use the event RESOURCE_STALLS.BR_MISS_CLEAR. Tips - See the optimization guide for tips on reducing branch mispredictions. - PGO's purpose is to have straight line code for the most frequent execution paths, reducing branches taken and increasing the "basic block" size, possibly also reducing the code footprint or working-set.
C6H	01H	CYCLES_INT_MASKED.CYCLES_INT_MASKED	Cycles during which interrupts are disabled	This event counts the number of cycles during which interrupts are disabled.
C6H	02H	CYCLES_INT_MASKED.CYCLES_INT_PENDING_AND_MASKED	Cycles during which interrupts are pending and disabled	This event counts the number of cycles during which there are pending interrupts but interrupts are disabled.
C7H	01H	SIMD_INST_RETIRED.PACKED_SINGLE	Retired Streaming SIMD Extensions (SSE) packed-single instructions	This event counts the number of SSE packed-single instructions retired.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
C7H	02H	SIMD_INST_RETIRED.SCALAR_SINGLE	Retired Streaming SIMD Extensions (SSE) scalar-single instructions	This event counts the number of SSE scalar-single instructions retired.
C7H	04H	SIMD_INST_RETIRED.PACKED_DOUBLE	Retired Streaming SIMD Extensions 2 (SSE2) packed-double instructions	This event counts the number of SSE2 packed-double instructions retired.
C7H	08H	SIMD_INST_RETIRED.SCALAR_DOUBLE	Retired Streaming SIMD Extensions 2 (SSE2) scalar-double instructions.	This event counts the number of SSE2 scalar-double instructions retired.
C7H	10H	SIMD_INST_RETIRED.VECTOR	Retired Streaming SIMD Extensions 2 (SSE2) vector instructions.	This event counts the number of SSE2 vector instructions retired.
C7H	1FH	SIMD_INST_RETIRED.ANY	Retired Streaming SIMD instructions	This event counts the overall number of SIMD instructions retired. To count each type of SIMD instruction separately, use the following events: SIMD_INST_RETIRED.PACKED_SINGLE, SIMD_INST_RETIRED.SCALAR_SINGLE, SIMD_INST_RETIRED.PACKED_DOUBLE, SIMD_INST_RETIRED.SCALAR_DOUBLE, and SIMD_INST_RETIRED.VECTOR.
C8H	00H	HW_INT_RCV	Hardware interrupts received	This event counts the number of hardware interrupts received by the processor. This event will count twice for dual-pipe micro-ops.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
CAH	01H	SIMD_COMP_INST_RETIRED.PACKED_SINGLE	Retired computational Streaming SIMD Extensions (SSE) packed-single instructions.	This event counts the number of computational SSE packed-single instructions retired. Computational instructions perform arithmetic computations, like add, multiply and divide. Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CAH	02H	SIMD_COMP_INST_RETIRED.SCALAR_SINGLE	Retired computational Streaming SIMD Extensions (SSE) scalar-single instructions.	This event counts the number of computational SSE scalar-single instructions retired. Computational instructions perform arithmetic computations, like add, multiply and divide. Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event
CAH	04H	SIMD_COMP_INST_RETIRED.PACKED_DOUBLE	Retired computational Streaming SIMD Extensions 2 (SSE2) packed-double instructions.	This event counts the number of computational SSE2 packed-double instructions retired. Computational instructions perform arithmetic computations, like add, multiply and divide. Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CAH	08H	SIMD_COMP_INST_RETIRED.SCALAR_DOUBLE	Retired computational Streaming SIMD Extensions 2 (SSE2) scalar-double instructions	This event counts the number of computational SSE2 scalar-double instructions retired. Computational instructions perform arithmetic computations, like add, multiply and divide. Instructions that perform load and store operations or logical operations, like XOR, OR, and AND are not counted by this event.
CBH	01H	MEM_LOAD_RETIRED.L2_HIT	Retired loads that hit the L2 cache (precise event)	This event counts the number of retired load operations that missed the L1 data cache and hit the L2 cache.
CBH	02H	MEM_LOAD_RETIRED.L2_MISS	Retired loads that miss the L2 cache (precise event)	This event counts the number of retired load operations that missed the L2 cache.

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
CBH	04H	MEM_LOAD_RETIRED.DTLB_MISS	Retired loads that miss the DTLB (precise event)	This event counts the number of retired loads that missed the DTLB. The DTLB miss is not counted if the load operation causes a fault.
CDH	00H	SIMD_ASSIST	SIMD assists invoked	This event counts the number of SIMD assists invoked. SIMD assists are invoked when an EMMS instruction is executed after MMX™ technology code has changed the MMX state in the floating point stack. For example, these assists are required in the following cases: Streaming SIMD Extensions (SSE) instructions: 1. Denormal input when the DAZ (Denormals Are Zeros) flag is off 2. Underflow result when the FTZ (Flush To Zero) flag is off
CEH	00H	SIMD_INSTR_RETIRED	SIMD Instructions retired	This event counts the number of SIMD instructions that retired.
CFH	00H	SIMD_SAT_INSTR_RETIRED	Saturated arithmetic instructions retired	This event counts the number of saturated arithmetic SIMD instructions that retired.
E0H	01H	BR_INST_DECODED	Branch instructions decoded	This event counts the number of branch instructions decoded

Table A-5. Non-Architectural Performance Events for Intel Atom Processors

Event Num.	Umask Value	Event Name	Definition	Description and Comment
E4H	01H	BOGUS_BR	Bogus branches	This event counts the number of byte sequences that were mistakenly detected as taken branch instructions. This results in a BACLEAR event and the BTB is flushed. This occurs mainly after task switches
E6H	01H	BACLEARS.ANY	BACLEARS asserted	This event counts the number of times the front end is redirected for a branch prediction, mainly when an early branch prediction is corrected by other branch handling mechanisms in the front-end. This can occur if the code has many branches such that they cannot be consumed by the branch predictor. Each Baclear asserted costs approximately 7 cycles. The effect on total execution time depends on the surrounding code

A.5 PERFORMANCE MONITORING EVENTS FOR INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Table A-6 lists non-architectural performance events for Intel Core Duo processors. If a non-architectural event requires qualification in core specificity, it is indicated in the comment column. Table A-6 also applies to Intel Core Solo processors; bits in the unit mask corresponding to core-specificity are reserved and should be 00B.

Table A-6. Non-Architectural Performance Events in Intel Core Solo and Intel Core Duo Processors

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
03H	LD_Blocks	00H	Load operations delayed due to store buffer blocks. The preceding store may be blocked due to unknown address, unknown data, or conflict due to partial overlap between the load and store.	
04H	SD_Drains	00H	Cycles while draining store buffers	
05H	Misalign_Mem_Ref	00H	Misaligned data memory references (MOB splits of loads and stores).	
06H	Seg_Reg_Loads	00H	Segment register loads	
07H	SSE_PrefNta_Ret	00H	SSE software prefetch instruction PREFETCHNTA retired	
07H	SSE_PrefT1_Ret	01H	SSE software prefetch instruction PREFETCHT1 retired	
07H	SSE_PrefT2_Ret	02H	SSE software prefetch instruction PREFETCHT2 retired	
07H	SSE_NTStores_Ret	03H	SSE streaming store instruction retired	
10H	FP_Comps_Op_Exe	00H	FP computational Instruction executed. FADD, FSUB, FCOM, FMULs, MUL, IMUL, FDIVs, DIV, IDIV, FPREMs, FSQRT are included; but exclude FADD or FMUL used in the middle of a transcendental instruction.	
11H	FP_Assist	00H	FP exceptions experienced microcode assists	IA32_PMC1 only.

Table A-6. Non-Architectural Performance Events in Intel Core Solo and Intel Core Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
12H	Mul	00H	Multiply operations (a speculative count, including FP and integer multiplies).	IA32_PMC1 only.
13H	Div	00H	Divide operations (a speculative count, including FP and integer divisions).	IA32_PMC1 only.
14H	Cycles_Div_Busy	00H	Cycles the divider is busy	IA32_PMC0 only.
21H	L2_ADS	00H	L2 Address strobes	Requires core-specificity
22H	Dbus_Busy	00H	Core cycle during which data bus was busy (increments by 4)	Requires core-specificity
23H	Dbus_Busy_Rd	00H	Cycles data bus is busy transferring data to a core (increments by 4)	Requires core-specificity
24H	L2_Lines_In	00H	L2 cache lines allocated	Requires core-specificity and HW prefetch qualification
25H	L2_M_Lines_In	00H	L2 Modified-state cache lines allocated	Requires core-specificity
26H	L2_Lines_Out	00H	L2 cache lines evicted	Requires core-specificity and HW prefetch qualification
27H	L2_M_Lines_Out	00H	L2 Modified-state cache lines evicted	
28H	L2_IFetch	Requires MESI qualification	L2 instruction fetches from instruction fetch unit (includes speculative fetches)	Requires core-specificity
29H	L2_LD	Requires MESI qualification	L2 cache reads	Requires core-specificity
2AH	L2_ST	Requires MESI qualification	L2 cache writes (includes speculation)	Requires core-specificity

Table A-6. Non-Architectural Performance Events in Intel Core Solo and Intel Core Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
2EH	L2_Rqsts	Requires MESI qualification	L2 cache reference requests	Requires core-specificity, HW prefetch qualification
30H	L2_Reject_Cycles	Requires MESI qualification	Cycles L2 is busy and rejecting new requests.	
32H	L2_No_Request_Cycles	Requires MESI qualification	Cycles there is no request to access L2.	
3AH	EST_Trans_All	00H	Any Intel Enhanced SpeedStep(R) Technology transitions	
3AH	EST_Trans_All	10H	Intel Enhanced SpeedStep Technology frequency transitions	
3BH	Thermal_Trip	C0H	Duration in a thermal trip based on the current core clock	Use edge trigger to count occurrence
3CH	NonHlt_Ref_Cycles	01H	Non-halted bus cycles	
3CH	Serial_Execution_Cycles	02H	Non-halted bus cycles of this core executing code while the other core is halted	
40H	DCache_Cache_LD	Requires MESI qualification	L1 cacheable data read operations	
41H	DCache_Cache_ST	Requires MESI qualification	L1 cacheable data write operations	
42H	DCache_Cache_Lock	Requires MESI qualification	L1 cacheable lock read operations to invalid state	
43H	Data_Mem_Ref	01H	L1 data read and writes of cacheable and non-cacheable types	
44H	Data_Mem_Cache_Ref	02H	L1 data cacheable read and write operations	
45H	DCache_Repl	0FH	L1 data cache line replacements	
46H	DCache_M_Repl	00H	L1 data M-state cache line allocated	

Table A-6. Non-Architectural Performance Events in Intel Core Solo and Intel Core Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
47H	DCache_M_Evict	00H	L1 data M-state cache line evicted	
48H	DCache_Pend_Miss	00H	Weighted cycles of L1 miss outstanding	Use Cmask =1 to count duration.
49H	Dtlb_Miss	00H	Data references that missed TLB	
4BH	SSE_PrefNta_Miss	00H	PREFETCHNTA missed all caches	
4BH	SSE_PrefT1_Miss	01H	PREFETCHT1 missed all caches	
4BH	SSE_PrefT2_Miss	02H	PREFETCHT2 missed all caches	
4BH	SSE_NTStores_ Miss	03H	SSE streaming store instruction missed all caches	
4FH	L1_Pref_Req	00H	L1 prefetch requests due to DCU cache misses	May overcount if request re-submitted
60H	Bus_Req_ Outstanding	00; Requires core-specificity, and agent specificity	Weighted cycles of cacheable bus data read requests. This event counts full-line read request from DCU or HW prefetcher, but not RFO, write, instruction fetches, or others.	Use Cmask =1 to count duration. Use Umask bit 12 to include HWP or exclude HWP separately.
61H	Bus_BNR_Clocks	00H	External bus cycles while BNR asserted	
62H	Bus_DRDY_Clocks	00H	External bus cycles while DRDY asserted	Requires agent specificity
63H	Bus_Locks_Clocks	00H	External bus cycles while bus lock signal asserted	Requires core specificity
64H	Bus_Data_Rcv	40H	External bus cycles while bus lock signal asserted	
65H	Bus_Trans_Brd	See comment.	Burst read bus transactions (data or code)	Requires core specificity

Table A-6. Non-Architectural Performance Events in Intel Core Solo and Intel Core Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
66H	Bus_Trans_RFO	See comment.	Completed read for ownership (RFO) transactions	Requires agent specificity Requires core specificity Each transaction counts its address strobe Retried transaction may be counted more than once
68H	Bus_Trans_Ifetch	See comment.	Completed instruction fetch transactions	
69H	Bus_Trans_Inval	See comment.	Completed invalidate transactions	
6AH	Bus_Trans_Pwr	See comment.	Completed partial write transactions	
6BH	Bus_Trans_P	See comment.	Completed partial transactions (include partial read + partial write + line write)	
6CH	Bus_Trans_IO	See comment.	Completed I/O transactions (read and write)	
6DH	Bus_Trans_Def	20H	Completed defer transactions	Requires core specificity Retried transaction may be counted more than once
67H	Bus_Trans_WB	C0H	Completed writeback transactions from DCU (does not include L2 writebacks)	Requires agent specificity Each transaction counts its address strobe Retried transaction may be counted more than once
6EH	Bus_Trans_Burst	C0H	Completed burst transactions (full line transactions include reads, write, RFO, and writebacks)	
6FH	Bus_Trans_Mem	C0H	Completed memory transactions. This includes Bus_Trans_Burst + Bus_Trans_P+Bus_Trans_Inval.	
70H	Bus_Trans_Any	C0H	Any completed bus transactions	
77H	Bus_Snoops	00H	External bus cycles while bus lock signal asserted	Requires MESI qualification Requires agent specificity
78H	DCU_Snoop_To_Share	01H	DCU snoops to share-state L1 cache line due to L1 misses	Requires core specificity
7DH	Bus_Not_In_Use	00H	Number of cycles there is no transaction from the core	Requires core specificity

Table A-6. Non-Architectural Performance Events in Intel Core Solo and Intel Core Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
7EH	Bus_Snoop_Stall	00H	Number of bus cycles while bus snoop is stalled	
80H	ICache_Reads	00H	Number of instruction fetches from ICache, streaming buffers (both cacheable and uncacheable fetches)	
81H	ICache_Misses	00H	Number of instruction fetch misses from ICache, streaming buffers.	
85H	ITLB_Misses	00H	Number of iITLB misses	
86H	IFU_Mem_Stall	00H	Cycles IFU is stalled while waiting for data from memory	
87H	ILD_Stall	00H	Number of instruction length decoder stalls (Counts number of LCP stalls)	
88H	Br_Inst_Exec	00H	Branch instruction executed (includes speculation).	
89H	Br_Missp_Exec	00H	Branch instructions executed and mispredicted at execution (includes branches that do not have prediction or mispredicted)	
8AH	Br_BAC_Missp_Exec	00H	Branch instructions executed that were mispredicted at front end	
8BH	Br_Cnd_Exec	00H	Conditional branch instructions executed	
8CH	Br_Cnd_Missp_Exec	00H	Conditional branch instructions executed that were mispredicted	
8DH	Br_Ind_Exec	00H	Indirect branch instructions executed	
8EH	Br_Ind_Missp_Exec	00H	Indirect branch instructions executed that were mispredicted	
8FH	Br_Ret_Exec	00H	Return branch instructions executed	
90H	Br_Ret_Missp_Exec	00H	Return branch instructions executed that were mispredicted	
91H	Br_Ret_BAC_Missp_Exec	00H	Return branch instructions executed that were mispredicted at the front end	

Table A-6. Non-Architectural Performance Events in Intel Core Solo and Intel Core Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
92H	Br_Call_Exec	00H	Return call instructions executed	
93H	Br_Call_Missp_Exec	00H	Return call instructions executed that were mispredicted	
94H	Br_Ind_Call_Exec	00H	Indirect call branch instructions executed	
A2H	Resource_Stall	00H	Cycles while there is a resource related stall (renaming, buffer entries) as seen by allocator	
B0H	MMX_Instr_Exec	00H	Number of MMX instructions executed (does not include MOVQ and MOVD stores)	
B1H	SIMD_Int_Sat_Exec	00H	Number of SIMD Integer saturating instructions executed	
B3H	SIMD_Int_Pmul_Exec	01H	Number of SIMD Integer packed multiply instructions executed	
B3H	SIMD_Int_Psft_Exec	02H	Number of SIMD Integer packed shift instructions executed	
B3H	SIMD_Int_Pck_Exec	04H	Number of SIMD Integer pack operations instruction executed	
B3H	SIMD_Int_Upck_Exec	08H	Number of SIMD Integer unpack instructions executed	
B3H	SIMD_Int_Plog_Exec	10H	Number of SIMD Integer packed logical instructions executed	
B3H	SIMD_Int_Pari_Exec	20H	Number of SIMD Integer packed arithmetic instructions executed	
C0H	Instr_Ret	00H	Number of instruction retired (Macro fused instruction count as 2)	
C1H	FP_Comp_Instr_Ret	00H	Number of FP compute instructions retired (X87 instruction or instruction that contain X87 operations)	Use IA32_PMC0 only.
C2H	Uops_Ret	00H	Number of micro-ops retired (include fused uops)	
C3H	SMC_Detected	00H	Number of times self-modifying code condition detected	

Table A-6. Non-Architectural Performance Events in Intel Core Solo and Intel Core Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
C4H	Br_Instr_Ret	00H	Number of branch instructions retired	
C5H	Br_MisPred_Ret	00H	Number of mispredicted branch instructions retired	
C6H	Cycles_Int_Masked	00H	Cycles while interrupt is disabled	
C7H	Cycles_Int_Pedning_ Masked	00H	Cycles while interrupt is disabled and interrupts are pending	
C8H	HW_Int_Rx	00H	Number of hardware interrupts received	
C9H	Br_Taken_Ret	00H	Number of taken branch instruction retired	
CAH	Br_MisPred_Taken_ Ret	00H	Number of taken and mispredicted branch instructions retired	
CCH	MMX_FP_Trans	00H	Number of transitions from MMX to X87	
CCH	FP_MMX_Trans	01H	Number of transitions from X87 to MMX	
CDH	MMX_Assist	00H	Number of EMMS executed	
CEH	MMX_Instr_Ret	00H	Number of MMX instruction retired	
D0H	Instr_Decoded	00H	Number of instruction decoded	
D7H	ESP_Uops	00H	Number of ESP folding instruction decoded	
D8H	SIMD_FP_SP_Ret	00H	Number of SSE/SSE2 single precision instructions retired (packed and scalar)	
D8H	SIMD_FP_SP_S_ Ret	01H	Number of SSE/SSE2 scalar single precision instructions retired	
D8H	SIMD_FP_DP_P_ Ret	02H	Number of SSE/SSE2 packed double precision instructions retired	
D8H	SIMD_FP_DP_S_ Ret	03H	Number of SSE/SSE2 scalar double precision instructions retired	
D8H	SIMD_Int_128_Ret	04H	Number of SSE2 128 bit integer instructions retired	

Table A-6. Non-Architectural Performance Events in Intel Core Solo and Intel Core Duo Processors (Contd.)

Event Num.	Event Mask Mnemonic	Umask Value	Description	Comment
D9H	SIMD_FP_SP_P_Comp_Ret	00H	Number of SSE/SSE2 packed single precision compute instructions retired (does not include AND, OR, XOR)	
D9H	SIMD_FP_SP_S_Comp_Ret	01H	Number of SSE/SSE2 scalar single precision compute instructions retired (does not include AND, OR, XOR)	
D9H	SIMD_FP_DP_P_Comp_Ret	02H	Number of SSE/SSE2 packed double precision compute instructions retired (does not include AND, OR, XOR)	
D9H	SIMD_FP_DP_S_Comp_Ret	03H	Number of SSE/SSE2 scalar double precision compute instructions retired (does not include AND, OR, XOR)	
DAH	Fused_Uops_Ret	00H	All fused uops retired	
DAH	Fused_Ld_Uops_Ret	01H	Fused load uops retired	
DAH	Fused_St_Uops_Ret	02H	Fused store uops retired	
DBH	Unfusion	00H	Number of unfusion events in the ROB (due to exception)	
E0H	Br_Instr_Decoded	00H	Branch instructions decoded	
E2H	BTB_Misses	00H	Number of branches the BTB did not produce a prediction	
E4H	Br_Bogus	00H	Number of bogus branches	
E6H	BAClears	00H	Number of BAClears asserted	
F0H	Pref_Rqsts_Up	00H	Number of hardware prefetch requests issued in forward streams	
F8H	Pref_Rqsts_Dn	00H	Number of hardware prefetch requests issued in backward streams	

A.6 PENTIUM 4 AND INTEL XEON PROCESSOR PERFORMANCE-MONITORING EVENTS

Tables A-7, A-8 and list performance-monitoring events that can be counted or sampled on processors based on Intel NetBurst microarchitecture. Table A-7 lists the non-retirement events, and Table A-8 lists the at-retirement events. Tables A-10, A-11, and A-12 describes three sets of parameters that are available for three of the at-retirement counting events defined in Table A-8. Table A-13 shows which of the non-retirement and at retirement events are logical processor specific (TS) (see Section 18.17.4, “Performance Monitoring Events”) and which are non-logical processor specific (TI).

Some of the Pentium 4 and Intel Xeon processor performance-monitoring events may be available only to specific models. The performance-monitoring events listed in Tables A-7 and A-8 apply to processors with CPUID signature that matches family encoding 15, model encoding 0, 1, 2 3, 4, or 6. Table applies to processors with a CPUID signature that matches family encoding 15, model encoding 3, 4 or 6.

The functionality of performance-monitoring events in Pentium 4 and Intel Xeon processors is also available when IA-32e mode is enabled.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting

Event Name	Event Parameters	Parameter Value	Description
TC_deliver_mode			This event counts the duration (in clock cycles) of the operating modes of the trace cache and decode engine in the processor package. The mode is specified by one or more of the event mask bits.
	ESCR restrictions	MSR_TC_ESCR0 MSR_TC_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	01H	ESCR[31:25]

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask		ESCR[24:9]
		Bit	
		0: DD	Both logical processors are in deliver mode.
		1: DB	Logical processor 0 is in deliver mode and logical processor 1 is in build mode.
		2: DI	Logical processor 0 is in deliver mode and logical processor 1 is either halted, under a machine clear condition or transitioning to a long microcode flow.
		3: BD	Logical processor 0 is in build mode and logical processor 1 is in deliver mode.
		4: BB	Both logical processors are in build mode.
		5: BI	Logical processor 0 is in build mode and logical processor 1 is either halted, under a machine clear condition or transitioning to a long microcode flow.
		6: ID	Logical processor 0 is either halted, under a machine clear condition or transitioning to a long microcode flow. Logical processor 1 is in deliver mode.
		7: IB	Logical processor 0 is either halted, under a machine clear condition or transitioning to a long microcode flow. Logical processor 1 is in build mode.
	CCCR Select	01H	CCCR[15:13]

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		If only one logical processor is available from a physical processor package, the event mask should be interpreted as logical processor 1 is halted. Event mask bit 2 was previously known as "DELIVER", bit 5 was previously known as "BUILD".
BPU_fetch_ request			This event counts instruction fetch requests of specified request type by the Branch Prediction unit. Specify one or more mask bits to qualify the request type(s).
	ESCR restrictions	MSR_BPU_ESCR0 MSR_BPU_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	03H	ESCR[31:25]
	ESCR Event Mask	 Bit 0: TCMISS	ESCR[24:9] Trace cache lookup miss
	CCCR Select	00H	CCCR[15:13]
ITLB_reference			This event counts translations using the Instruction Translation Look-aside Buffer (ITLB).
	ESCR restrictions	MSR_ITLB_ESCR0 MSR_ITLB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	18H	ESCR[31:25]

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask	 Bit 0: HIT 1: MISS 2: HIT_UC	ESCR[24:9] ITLB hit ITLB miss Uncacheable ITLB hit
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		All page references regardless of the page size are looked up as actual 4-KByte pages. Use the page_walk_type event with the ITMISS mask for a more conservative count.
memory_cancel			This event counts the canceling of various type of request in the Data cache Address Control unit (DAC). Specify one or more mask bits to select the type of requests that are canceled.
	ESCR restrictions	MSR_DAC_ESCR0 MSR_DAC_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask	 Bit 2: ST_RB_FULL 3: 64K_CONF	ESCR[24:9] Replayed because no store request buffer is available Conflicts due to 64-KByte aliasing
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		All_CACHE_MISS includes uncacheable memory in count

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
memory_ complete			This event counts the completion of a load split, store split, uncacheable (UC) split, or UC load. Specify one or more mask bits to select the operations to be counted.
	ESCR restrictions	MSR_SAAT_ESCR0 MSR_SAAT_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	08H	ESCR[31:25]
	ESCR Event Mask	Bit 0: LSC 1: SSC	ESCR[24:9] Load split completed, excluding UC/WC loads Any split stores completed
	CCCR Select	02H	CCCR[15:13]
load_port_replay			This event counts replayed events at the load port. Specify one or more mask bits to select the cause of the replay.
	ESCR restrictions	MSR_SAAT_ESCR0 MSR_SAAT_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	04H	ESCR[31:25]
	ESCR Event Mask	Bit 1: SPLIT_LD	ESCR[24:9] Split load.
	CCCR Select	02H	CCCR[15:13]
	Event Specific Notes		Must use ESCR1 for at-retirement counting

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
store_port_replay			This event counts replayed events at the store port. Specify one or more mask bits to select the cause of the replay.
	ESCR restrictions	MSR_SAAT_ESCR0 MSR_SAAT_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	05H	ESCR[31:25]
	ESCR Event Mask	Bit 1: SPLIT_ST	ESCR[24:9] Split store
	CCCR Select	02H	CCCR[15:13]
	Event Specific Notes		Must use ESCR1 for at-retirement counting
MOB_load_replay			This event triggers if the memory order buffer (MOB) caused a load operation to be replayed. Specify one or more mask bits to select the cause of the replay.
	ESCR restrictions	MSR_MOB_ESCR0 MSR_MOB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	03H	ESCR[31:25]
	ESCR Event Mask	Bit 1: NO_STA 3: NO_STD	ESCR[24:9] Replayed because of unknown store address Replayed because of unknown store data

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
		4: PARTIAL_DATA	Replayed because of partially overlapped data access between the load and store operations
		5: UNALGN_ADDR	Replayed because the lower 4 bits of the linear address do not match between the load and store operations
	CCCR Select	02H	CCCR[15:13]
page_walk_type			This event counts various types of page walks that the page miss handler (PMH) performs.
	ESCR restrictions	MSR_PMH_ ESCR0 MSR_PMH_ ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	01H	ESCR[31:25]
	ESCR Event Mask		ESCR[24:9]
		Bit	
		0: DTMISS	Page walk for a data TLB miss (either load or store)
		1: ITMISS	Page walk for an instruction TLB miss
	CCCR Select	04H	CCCR[15:13]
BSQ_cache _reference			This event counts cache references (2nd level cache or 3rd level cache) as seen by the bus unit. Specify one or more mask bit to select an access according to the access type (read type includes both load and RFO, write type includes writebacks and evictions) and the access result (hit, misses).

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR restrictions	MSR_BSU_ESCR0 MSR_BSU_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	0CH	ESCR[31:25]
			ESCR[24:9]
		Bit	
		0: RD_2ndL_HITS	Read 2nd level cache hit Shared (includes load and RFO)
		1: RD_2ndL_HITE	Read 2nd level cache hit Exclusive (includes load and RFO)
		2: RD_2ndL_HITM	Read 2nd level cache hit Modified (includes load and RFO)
		3: RD_3rdL_HITS	Read 3rd level cache hit Shared (includes load and RFO)
		4: RD_3rdL_HITE	Read 3rd level cache hit Exclusive (includes load and RFO)
		5: RD_3rdL_HITM	Read 3rd level cache hit Modified (includes load and RFO)
	ESCR Event Mask	8: RD_2ndL_MISS	Read 2nd level cache miss (includes load and RFO)
		9: RD_3rdL_MISS	Read 3rd level cache miss (includes load and RFO)
		10: WR_2ndL_MISS	A Writeback lookup from DAC misses the 2nd level cache (unlikely to happen)
	CCCR Select	07H	CCCR[15:13]
	Event Specific Notes		1: The implementation of this event in current Pentium 4 and Xeon processors treats either a load operation or a request for ownership (RFO) request as a “read” type operation.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
			2: Currently this event causes both over and undercounting by as much as a factor of two due to an erratum. 3: It is possible for a transaction that is started as a prefetch to change the transaction's internal status, making it no longer a prefetch. or change the access result status (hit, miss) as seen by this event.
IOQ_allocation			This event counts the various types of transactions on the bus. A count is generated each time a transaction is allocated into the IOQ that matches the specified mask bits. An allocated entry can be a sector (64 bytes) or a chunks of 8 bytes.
			Requests are counted once per retry. The event mask bits constitute 4 bit fields. A transaction type is specified by interpreting the values of each bit field. Specify one or more event mask bits in a bit field to select the value of the bit field. Each field (bits 0-4 are one field) are independent of and can be ORed with the others. The request type field is further combined with bit 5 and 6 to form a binary expression. Bits 7 and 8 form a bit field to specify the memory type of the target address.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
			Bits 13 and 14 form a bit field to specify the source agent of the request. Bit 15 affects read operation only. The event is triggered by evaluating the logical expression: (((Request type) OR Bit 5 OR Bit 6) OR (Memory type)) AND (Source agent).
	ESCR restrictions	MSR_FSB_ESCR0, MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1; ESCR1: 2, 3	
	ESCR Event Select	03H	ESCR[31:25]
	ESCR Event Mask		ESCR[24:9]
		Bits	
		0-4 (single field)	Bus request type (use 00001 for invalid or default)
		5: ALL_READ	Count read entries
		6: ALL_WRITE	Count write entries
		7: MEM_UC	Count UC memory access entries
		8: MEM_WC	Count WC memory access entries
		9: MEM_WT	Count write-through (WT) memory access entries.
		10: MEM_WP	Count write-protected (WP) memory access entries
		11: MEM_WB	Count WB memory access entries.
		13: OWN	Count all store requests driven by processor, as opposed to other processor or DMA.
		14: OTHER	Count all requests driven by other processors or DMA.
		15: PREFETCH	Include HW and SW prefetch requests in the count.
	CCCR Select	06H	CCCR[15:13]

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		1: If PREFETCH bit is cleared, sectors fetched using prefetch are excluded in the counts. If PREFETCH bit is set, all sectors or chunks read are counted. 2: Specify the edge trigger in CCCR to avoid double counting.
			3: The mapping of interpreted bit field values to transaction types may differ with different processor model implementations of the Pentium 4 processor family. Applications that program performance monitoring events should use CPUID to determine processor models when using this event. The logic equations that trigger the event are model-specific (see 4a and 4b below). 4a:For Pentium 4 and Xeon Processors starting with CPUID Model field encoding equal to 2 or greater, this event is triggered by evaluating the logical expression ((Request type) and (Bit 5 or Bit 6) and (Memory type) and (Source agent)).

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
			4b:For Pentium 4 and Xeon Processors with CPUID Model field encoding less than 2, this event is triggered by evaluating the logical expression [((Request type) or Bit 5 or Bit 6) or (Memory type)] and (Source agent). Note that event mask bits for memory type are ignored if either ALL_READ or ALL_WRITE is specified.
			5: This event is known to ignore CPL in early implementations of Pentium 4 and Xeon Processors. Both user requests and OS requests are included in the count. This behavior is fixed starting with Pentium 4 and Xeon Processors with CPUID signature 0xF27 (Family 15, Model 2, Stepping 7). 6: For write-through (WT) and write-protected (WP) memory types, this event counts reads as the number of 64-byte sectors. Writes are counted by individual chunks. 7: For uncacheable (UC) memory types, this events counts the number of 8-byte chunks allocated. 8: For Pentium 4 and Xeon Processors with CPUID Signature less than 0xf27, only MSR_FSB_ESCR0 is available.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
IOQ_active_ entries			This event counts the number of entries (clipped at 15) in the IOQ that are active. An allocated entry can be a sector (64 bytes) or a chunks of 8 bytes. The event must be programmed in conjunction with IOQ_allocation. Specify one or more event mask bits to select the transactions that is counted.
	ESCR restrictions	MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR1: 2, 3	
	ESCR Event Select	01AH	ESCR[30:25]
	ESCR Event Mask		ESCR[24:9]
		Bits	
		0-4 (single field)	Bus request type (use 00001 for invalid or default).
		5: ALL_READ	Count read entries.
		6: ALL_WRITE	Count write entries.
		7: MEM_UC	Count UC memory access entries.
		8: MEM_WC	Count WC memory access entries.
		9: MEM_WT	Count write-through (WT) memory access entries.
		10: MEM_WP	Count write-protected (WP) memory access entries.
		11: MEM_WB	Count WB memory access entries.
		13: OWN	Count all store requests driven by processor, as opposed to other processor or DMA.
		14: OTHER	Count all requests driven by other processors or DMA.
		15: PREFETCH	Include HW and SW prefetch requests in the count.
	CCCR Select	06H	CCCR[15:13]

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		1: Specified desired mask bits in ESCR0 and ESCR1. 2: See the ioq_allocation event for descriptions of the mask bits. 3: Edge triggering should not be used when counting cycles.
			4: The mapping of interpreted bit field values to transaction types may differ across different processor model implementations of the Pentium 4 processor family. Applications that programs performance monitoring events should use the CPUID instruction to detect processor models when using this event. The logical expression that triggers this event as describe below: 5a:For Pentium 4 and Xeon Processors starting with CPUID MODEL field encoding equal to 2 or greater, this event is triggered by evaluating the logical expression ((Request type) and (Bit 5 or Bit 6) and (Memory type) and (Source agent)).

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
			5b:For Pentium 4 and Xeon Processors starting with CPUID MODEL field encoding less than 2, this event is triggered by evaluating the logical expression [((Request type) or Bit 5 or Bit 6) or (Memory type)] and (Source agent). Event mask bits for memory type are ignored if either ALL_READ or ALL_WRITE is specified. 5c: This event is known to ignore CPL in the current implementations of Pentium 4 and Xeon Processors Both user requests and OS requests are included in the count.
			6: An allocated entry can be a full line (64 bytes) or in individual chunks of 8 bytes.
FSB_data_ activity			This event increments once for each DRDY or DBSY event that occurs on the front side bus. The event allows selection of a specific DRDY or DBSY event.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	17H	ESCR[31:25]
	ESCR Event Mask		ESCR[24:9]
		Bit 0: DRDY_DRV	Count when this processor drives data onto the bus - includes writes and implicit writebacks.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
			Asserted two processor clock cycles for partial writes and 4 processor clocks (usually in consecutive bus clocks) for full line writes.
		1: DRDY_OWN	Count when this processor reads data from the bus - includes loads and some PIC transactions. Asserted two processor clock cycles for partial reads and 4 processor clocks (usually in consecutive bus clocks) for full line reads. Count DRDY events that we drive. Count DRDY events sampled that we own.
		2: DRDY_OTHER	Count when data is on the bus but not being sampled by the processor. It may or may not be being driven by this processor. Asserted two processor clock cycles for partial transactions and 4 processor clocks (usually in consecutive bus clocks) for full line transactions.
		3: DBSY_DRV	Count when this processor reserves the bus for use in the next bus cycle in order to drive data. Asserted for two processor clock cycles for full line writes and not at all for partial line writes. May be asserted multiple times (in consecutive bus clocks) if we stall the bus waiting for a cache lock to complete.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
		4: DBSY_OWN	Count when some agent reserves the bus for use in the next bus cycle to drive data that this processor will sample. Asserted for two processor clock cycles for full line writes and not at all for partial line writes. May be asserted multiple times (all one bus clock apart) if we stall the bus for some reason.
		5:DBSY_OTHER	Count when some agent reserves the bus for use in the next bus cycle to drive data that this processor will NOT sample. It may or may not be being driven by this processor. Asserted two processor clock cycles for partial transactions and 4 processor clocks (usually in consecutive bus clocks) for full line transactions.
	CCCR Select	06H	CCCR[15:13]
	Event Specific Notes		Specify edge trigger in the CCCR MSR to avoid double counting. DRDY_OWN and DRDY_OTHER are mutually exclusive; similarly for DBSY_OWN and DBSY_OTHER.
BSQ_allocation			This event counts allocations in the Bus Sequence Unit (BSQ) according to the specified mask bit encoding. The event mask bits consist of four sub-groups: ▪ request type, ▪ request length ▪ memory type ▪ and sub-group consisting mostly of independent bits (bits 5, 6, 7, 8, 9, and 10) Specify an encoding for each sub-group.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR restrictions	MSR_BSU_ESCR0	
	Counter numbers per ESCR	ESCR0: 0, 1	
	ESCR Event Select	05H	ESCR[31:25]
	ESCR Event Mask	Bit	ESCR[24:9]
		0: REQ_TYPE0 1: REQ_TYPE1	Request type encoding (bit 0 and 1) are: 0 – Read (excludes read invalidate) 1 – Read invalidate 2 – Write (other than writebacks) 3 – Writeback (evicted from cache). (public)
		2: REQ_LEN0 3: REQ_LEN1	Request length encoding (bit 2, 3) are: 0 – 0 chunks 1 – 1 chunks 3 – 8 chunks
		5: REQ_IO_TYPE	Request type is input or output.
		6: REQ_LOCK_ TYPE	Request type is bus lock.
		7: REQ_CACHE_ TYPE	Request type is cacheable.
		8: REQ_SPLIT_ TYPE	Request type is a bus 8-byte chunk split across 8-byte boundary.
		9: REQ_DEM_TYPE	Request type is a demand if set. Request type is HW.SW prefetch if 0.
		10: REQ_ORD_ TYPE	Request is an ordered type.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
		11: MEM_TYPE0 12: MEM_TYPE1 13: MEM_TYPE2	Memory type encodings (bit 11-13) are: 0 – UC 1 – WC 4 – WT 5 – WP 6 – WB
	CCCR Select	07H	CCCR[15:13]
	Event Specific Notes		1: Specify edge trigger in CCCR to avoid double counting. 2: A writebacks to 3rd level cache from 2nd level cache counts as a separate entry, this is in additional to the entry allocated for a request to the bus. 3: A read request to WB memory type results in a request to the 64-byte sector, containing the target address, followed by a prefetch request to an adjacent sector.
			4: For Pentium 4 and Xeon processors with CPUID model encoding value equals to 0 and 1, an allocated BSQ entry includes both the demand sector and prefetched 2nd sector. 5: An allocated BSQ entry for a data chunk is any request less than 64 bytes. 6a: This event may undercount for requests of split type transactions if the data address straddled across modulo-64 byte boundary.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
			6b:This event may undercount for requests of read request of 16-byte operands from WC or UC address. 6c: This event may undercount WC partial requests originated from store operands that are dwords.
bsq_active_ entries			This event represents the number of BSQ entries (clipped at 15) currently active (valid) which meet the subevent mask criteria during allocation in the BSQ. Active request entries are allocated on the BSQ until de-allocated. De-allocation of an entry does not necessarily imply the request is filled. This event must be programmed in conjunction with BSQ_allocation. Specify one or more event mask bits to select the transactions that is counted.
	ESCR restrictions	ESCR1	
	Counter numbers per ESCR	ESCR1: 2, 3	
	ESCR Event Select	06H	ESCR[30:25]
	ESCR Event Mask		ESCR[24:9]
	CCCR Select	07H	CCCR[15:13]
	Event Specific Notes		1: Specified desired mask bits in ESCR0 and ESCR1. 2: See the BSQ_allocation event for descriptions of the mask bits. 3: Edge triggering should not be used when counting cycles.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
			4: This event can be used to estimate the latency of a transaction from allocation to de-allocation in the BSQ. The latency observed by BSQ_allocation includes the latency of FSB, plus additional overhead.
			5: Additional overhead may include the time it takes to issue two requests (the sector by demand and the adjacent sector via prefetch). Since adjacent sector prefetches have lower priority that demand fetches, on a heavily used system there is a high probability that the adjacent sector prefetch will have to wait until the next bus arbitration. 6: For Pentium 4 and Xeon processors with CPUID model encoding value less than 3, this event is updated every clock. 7: For Pentium 4 and Xeon processors with CPUID model encoding value equals to 3 or 4, this event is updated every other clock.
SSE_input_assist			This event counts the number of times an assist is requested to handle problems with input operands for SSE/SSE2/SSE3 operations; most notably denormal source operands when the DAZ bit is not set. Set bit 15 of the event mask to use this event.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	34H	ESCR[31:25]
	ESCR Event Mask	15: ALL	ESCR[24:9] Count assists for SSE/SSE2/SSE3 μops.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		1: Not all requests for assists are actually taken. This event is known to overcount in that it counts requests for assists from instructions on the non-retired path that do not incur a performance penalty. An assist is actually taken only for non-bogus μops. Any appreciable counts for this event are an indication that the DAZ or FTZ bit should be set and/or the source code should be changed to eliminate the condition.
			2: Two common situations for an SSE/SSE2/SSE3 operation needing an assist are: (1) when a denormal constant is used as an input and the Denormals-Are-Zero (DAZ) mode is not set, (2) when the input operand uses the underflowed result of a previous SSE/SSE2/SSE3 operation and neither the DAZ nor Flush-To-Zero (FTZ) modes are set.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
			3: Enabling the DAZ mode prevents SSE/SSE2/SSE3 operations from needing assists in the first situation. Enabling the FTZ mode prevents SSE/SSE2/SSE3 operations from needing assists in the second situation.
packed_SP_uop			This event increments for each packed single-precision µop, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	08H	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all µops operating on packed single-precision operands.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		1: If an instruction contains more than one packed SP µops, each packed SP µop that is specified by the event mask will be counted. 2: This metric counts instances of packed memory µops in a repeat move string.
packed_DP_uop			This event increments for each packed double-precision µop, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	0CH	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all µops operating on packed double-precision operands.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		If an instruction contains more than one packed DP µops, each packed DP µop that is specified by the event mask will be counted.
scalar_SP_uop			This event increments for each scalar single-precision µop, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	0AH	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all µops operating on scalar single-precision operands.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		If an instruction contains more than one scalar SP µops, each scalar SP µop that is specified by the event mask will be counted.
scalar_DP_uop			This event increments for each scalar double-precision µop, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	0EH	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all μops operating on scalar double-precision operands.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		If an instruction contains more than one scalar DP μops, each scalar DP μop that is specified by the event mask is counted.
64bit_MMX_uop			This event increments for each MMX instruction, which operate on 64-bit SIMD operands.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all μops operating on 64-bit SIMD integer operands in memory or MMX registers.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		If an instruction contains more than one 64-bit MMX μops, each 64-bit MMX μop that is specified by the event mask will be counted.
128bit_MMX_uop			This event increments for each integer SIMD SSE2 instruction, which operate on 128-bit SIMD operands.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	1AH	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all μops operating on 128-bit SIMD integer operands in memory or XMM registers.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		If an instruction contains more than one 128-bit MMX μops, each 128-bit MMX μop that is specified by the event mask will be counted.
x87_FP_uop			This event increments for each x87 floating-point μop, specified through the event mask for detection.
	ESCR restrictions	MSR_FIRM_ESCR0 MSR_FIRM_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	04H	ESCR[31:25]
	ESCR Event Mask	Bit 15: ALL	ESCR[24:9] Count all x87 FP μops.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		1: If an instruction contains more than one x87 FP μops, each x87 FP μop that is specified by the event mask will be counted. 2: This event does not count x87 FP μop for load, store, move between registers.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
TC_misc			This event counts miscellaneous events detected by the TC. The counter will count twice for each occurrence.
	ESCR restrictions	MSR_TC_ESCR0 MSR_TC_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	06H	ESCR[31:25]
	CCCR Select	01H	CCCR[15:13]
	ESCR Event Mask	 Bit 4: FLUSH	ESCR[24:9] Number of flushes
global_power _events			This event accumulates the time during which a processor is not stopped.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	013H	ESCR[31:25]
	ESCR Event Mask	Bit 0: Running	ESCR[24:9] The processor is active (includes the handling of HLT STPCLK and throttling.
	CCCR Select	06H	CCCR[15:13]
tc_ms_xfer			This event counts the number of times that uop delivery changed from TC to MS ROM.
	ESCR restrictions	MSR_MS_ESCR0 MSR_MS_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	05H	ESCR[31:25]

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask	Bit 0: CISC	ESCR[24:9] A TC to MS transfer occurred.
	CCCR Select	0H	CCCR[15:13]
uop_queue_ writes			This event counts the number of valid uops written to the uop queue. Specify one or more mask bits to select the source type of writes.
	ESCR restrictions	MSR_MS_ESCR0 MSR_MS_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	09H	ESCR[31:25]
	ESCR Event Mask	Bit 0: FROM_TC_ BUILD 1: FROM_TC_ DELIVER 2: FROM_ROM	ESCR[24:9] The uops being written are from TC build mode. The uops being written are from TC deliver mode. The uops being written are from microcode ROM.
	CCCR Select	0H	CCCR[15:13]
retired_mispred _branch_type			This event counts retiring mispredicted branches by type.
	ESCR restrictions	MSR_TBPU_ESCR0 MSR_TBPU_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	05H	ESCR[30:25]
	ESCR Event Mask	Bit 1: CONDITIONAL 2: CALL	ESCR[24:9] Conditional jumps. Indirect call branches.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
		3: RETURN 4: INDIRECT	Return branches. Returns, indirect calls, or indirect jumps.
	CCCR Select	02H	CCCR[15:13]
	Event Specific Notes		This event may overcount conditional branches if: ▪ Mispredictions cause the trace cache and delivery engine to build new traces. ▪ When the processor's pipeline is being cleared.
retired_branch _type			This event counts retiring branches by type. Specify one or more mask bits to qualify the branch by its type
	ESCR restrictions	MSR_TBPU_ESCR0 MSR_TBPU_ESCR1	
	Counter numbers per ESCR	ESCR0: 4, 5 ESCR1: 6, 7	
	ESCR Event Select	04H	ESCR[30:25]
	ESCR Event Mask	Bit 1: CONDITIONAL 2: CALL 3: RETURN 4: INDIRECT	ESCR[24:9] Conditional jumps. Direct or indirect calls. Return branches. Returns, indirect calls, or indirect jumps.
	CCCR Select	02H	CCCR[15:13]
	Event Specific Notes		This event may overcount conditional branches if : ▪ Mispredictions cause the trace cache and delivery engine to build new traces. ▪ When the processor's pipeline is being cleared.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
resource_stall			This event monitors the occurrence or latency of stalls in the Allocator.
	ESCR restrictions	MSR_ALF_ESCR0 MSR_ALF_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	01H	ESCR[30:25]
	Event Masks	Bit 5: SBFULL	ESCR[24:9] A Stall due to lack of store buffers.
	CCCR Select	01H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.
WC_Buffer			This event counts Write Combining Buffer operations that are selected by the event mask.
	ESCR restrictions	MSR_DAC_ESCR0 MSR_DAC_ESCR1	
	Counter numbers per ESCR	ESCR0: 8, 9 ESCR1: 10, 11	
	ESCR Event Select	05H	ESCR[30:25]
	Event Masks	Bit 0: WCB_EVICTS	ESCR[24:9] WC Buffer evictions of all causes.
		1: WCB_FULL_ EVICT	WC Buffer eviction: no WC buffer is available.
	CCCR Select	05H	CCCR[15:13]

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		This event is useful for detecting the subset of 64K aliasing cases that are more costly (i.e. 64K aliasing cases involving stores) as long as there are no significant contributions due to write combining buffer full or hit-modified conditions.
b2b_cycles			This event can be configured to count the number back-to-back bus cycles using sub-event mask bits 1 through 6.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	016H	ESCR[30:25]
	Event Masks	Bit	ESCR[24:9]
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.
bnr			This event can be configured to count bus not ready conditions using sub-event mask bits 0 through 2.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	08H	ESCR[30:25]
	Event Masks	Bit	ESCR[24:9]
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.

Table A-7. Performance Monitoring Events Supported by Intel NetBurst Microarchitecture for Non-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
snoop			This event can be configured to count snoop hit modified bus traffic using sub-event mask bits 2, 6 and 7.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	06H	ESCR[30:25]
	Event Masks	Bit	ESCR[24:9]
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.
Response			This event can be configured to count different types of responses using sub-event mask bits 1,2, 8, and 9.
	ESCR restrictions	MSR_FSB_ESCR0 MSR_FSB_ESCR1	
	Counter numbers per ESCR	ESCR0: 0, 1 ESCR1: 2, 3	
	ESCR Event Select	04H	ESCR[30:25]
	Event Masks	Bit	ESCR[24:9]
	CCCR Select	03H	CCCR[15:13]
	Event Specific Notes		This event may not be supported in all models of the processor family.

Table A-8. Performance Monitoring Events For Intel NetBurst Microarchitecture for At-Retirement Counting

Event Name	Event Parameters	Parameter Value	Description
front_end_event			This event counts the retirement of tagged μops, which are specified through the front-end tagging mechanism. The event mask specifies bogus or non-bogus μops.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	08H	ESCR[31:25]
	ESCR Event Mask		ESCR[24:9]
		Bit	
		0: NBOGUS	The marked μops are not bogus.
		1: BOGUS	The marked μops are bogus.
	CCCR Select	05H	CCCR[15:13]
	Can Support PEBS	Yes	
	Require Additional MSRs for tagging	Selected ESCRs and/or MSR_TC_PRECISE_EVENT	See list of metrics supported by Front_end tagging in Table A-3
execution_event			This event counts the retirement of tagged μops, which are specified through the execution tagging mechanism. The event mask allows from one to four types of μops to be specified as either bogus or non-bogus μops to be tagged.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	0CH	ESCR[31:25]

Table A-8. Performance Monitoring Events For Intel NetBurst Microarchitecture for At-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask		ESCR[24:9]
		Bit	
		0: NBOGUS0	The marked μops are not bogus.
		1: NBOGUS1	The marked μops are not bogus.
		2: NBOGUS2	The marked μops are not bogus.
		3: NBOGUS3	The marked μops are not bogus.
		4: BOGUS0	The marked μops are bogus.
		5: BOGUS1	The marked μops are bogus.
		6: BOGUS2	The marked μops are bogus.
		7: BOGUS3	The marked μops are bogus.
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		Each of the 4 slots to specify the bogus/non-bogus μops must be coordinated with the 4 TagValue bits in the ESCR (for example, NBOGUS0 must accompany a '1' in the lowest bit of the TagValue field in ESCR, NBOGUS1 must accompany a '1' in the next but lowest bit of the TagValue field).
	Can Support PEBS	Yes	
	Require Additional MSRs for tagging	An ESCR for an upstream event	See list of metrics supported by execution tagging in Table A-4.
replay_event			This event counts the retirement of tagged μops, which are specified through the replay tagging mechanism. The event mask specifies bogus or non-bogus μops.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	09H	ESCR[31:25]

Table A-8. Performance Monitoring Events For Intel NetBurst Microarchitecture for At-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Mask	 Bit 0: NBOGUS 1: BOGUS	ESCR[24:9] The marked μops are not bogus. The marked μops are bogus.
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		Supports counting tagged μops with additional MSRs.
	Can Support PEBS	Yes	
	Require Additional MSRs for tagging	IA32_PEBS_ ENABLE MSR_PEBS_ MATRIX_VERT Selected ESCR	See list of metrics supported by replay tagging in Table A-5.
instr_retired			This event counts instructions that are retired during a clock cycle. Mask bits specify bogus or non-bogus (and whether they are tagged using the front-end tagging mechanism).
	ESCR restrictions	MSR_CRU_ESCR0 MSR_CRU_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask	 Bit 0: NBOGUSNTAG 1: NBOGUSTAG 2: BOGUSNTAG 3: BOGUSTAG	ESCR[24:9] Non-bogus instructions that are not tagged. Non-bogus instructions that are tagged. Bogus instructions that are not tagged. Bogus instructions that are tagged.
	CCCR Select	04H	CCCR[15:13]

Table A-8. Performance Monitoring Events For Intel NetBurst Microarchitecture for At-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	Event Specific Notes		1: The event count may vary depending on the microarchitectural states of the processor when the event detection is enabled. 2: The event may count more than once for some instructions with complex uop flows and were interrupted before retirement.
	Can Support PEBS	No	
uops_retired			This event counts μops that are retired during a clock cycle. Mask bits specify bogus or non-bogus.
	ESCR restrictions	MSR_CRU_ESCR0 MSR_CRU_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	01H	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUS 1: BOGUS	ESCR[24:9] The marked μops are not bogus. The marked μops are bogus.
	CCCR Select	04H	CCCR[15:13]
	Event Specific Notes		P6: EMON_UOPS_RETIRED
	Can Support PEBS	No	
uop_type			This event is used in conjunction with the front-end at-retirement mechanism to tag load and store μops.
	ESCR restrictions	MSR_RAT_ESCR0 MSR_RAT_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	

Table A-8. Performance Monitoring Events For Intel NetBurst Microarchitecture for At-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask	Bit 1: TAGLOADS 2: TAGSTORES	ESCR[24:9] The µop is a load operation. The µop is a store operation.
	CCCR Select	02H	CCCR[15:13]
	Event Specific Notes		Setting the TAGLOADS and TAGSTORES mask bits does not cause a counter to increment. They are only used to tag uops.
	Can Support PEBS	No	
branch_retired			This event counts the retirement of a branch. Specify one or more mask bits to select any combination of taken, not-taken, predicted and mispredicted.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	See Table 18-18 for the addresses of the ESCR MSRs
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	The counter numbers associated with each ESCR are provided. The performance counters and corresponding CCCRs can be obtained from Table 18-18.
	ESCR Event Select	06H	ESCR[31:25]
	ESCR Event Mask	Bit 0: MMNP 1: MMNM 2: MMTP 3: MMTM	ESCR[24:9] Branch not-taken predicted Branch not-taken mispredicted Branch taken predicted Branch taken mispredicted
	CCCR Select	05H	CCCR[15:13]
	Event Specific Notes		P6: EMON_BR_INST_RETIRED
	Can Support PEBS	No	

Table A-8. Performance Monitoring Events For Intel NetBurst Microarchitecture for At-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
mispred_branch_retired			This event represents the retirement of mispredicted branch instructions.
	ESCR restrictions	MSR_CRU_ESCR0 MSR_CRU_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	03H	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUS	ESCR[24:9] The retired instruction is not bogus.
	CCCR Select	04H	CCCR[15:13]
	Can Support PEBS	No	
x87_assist			This event counts the retirement of x87 instructions that required special handling. Specifies one or more event mask bits to select the type of assistance.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	03H	ESCR[31:25]
	ESCR Event Mask	Bit 0: FPSU 1: FPSO 2: POAO 3: POAU 4: PREA	ESCR[24:9] Handle FP stack underflow Handle FP stack overflow Handle x87 output overflow Handle x87 output underflow Handle x87 input assist
	CCCR Select	05H	CCCR[15:13]
	Can Support PEBS	No	

Table A-8. Performance Monitoring Events For Intel NetBurst Microarchitecture for At-Retirement Counting (Contd.)

Event Name	Event Parameters	Parameter Value	Description
machine_clear			This event increments according to the mask bit specified while the entire pipeline of the machine is cleared. Specify one of the mask bit to select the cause.
	ESCR restrictions	MSR_CRU_ESCR2 MSR_CRU_ESCR3	
	Counter numbers per ESCR	ESCR2: 12, 13, 16 ESCR3: 14, 15, 17	
	ESCR Event Select	02H	ESCR[31:25]
	ESCR Event Mask		ESCR[24:9]
		Bit	
		0: CLEAR	Counts for a portion of the many cycles while the machine is cleared for any cause. Use Edge triggering for this bit only to get a count of occurrence versus a duration.
		2: MOCLEAR	Increments each time the machine is cleared due to memory ordering issues.
		6: SMCLEAR	Increments each time the machine is cleared due to self-modifying code issues.
	CCCR Select	05H	CCCR[15:13]
	Can Support PEBS	No	

Table A-9. Intel NetBurst Microarchitecture Model-Specific Performance Monitoring Events (For Model Encoding 3, 4 or 6)

Event Name	Event Parameters	Parameter Value	Description
instr_completed			This event counts instructions that have completed and retired during a clock cycle. Mask bits specify whether the instruction is bogus or non-bogus and whether they are:
	ESCR restrictions	MSR_CRU_ESCR0 MSR_CRU_ESCR1	
	Counter numbers per ESCR	ESCR0: 12, 13, 16 ESCR1: 14, 15, 17	
	ESCR Event Select	07H	ESCR[31:25]
	ESCR Event Mask	Bit 0: NBOGUS 1: BOGUS	ESCR[24:9] Non-bogus instructions Bogus instructions
	CCCR Select	04H	CCCR[15:13]
	Event Specific Notes		This metric differs from instr_retired, since it counts instructions completed, rather than the number of times that instructions started.
	Can Support PEBS	No	

Table A-10. List of Metrics Available for Front_end Tagging (For Front_end Event Only)

Front-end metric[1]	MSR_ TC_PRECISE_EVENT MSR Bit field	Additional MSR	Event mask value for Front_end_event
memory_loads	None	Set TAGLOADS bit in ESCR corresponding to event Uop_Type.	NBOGUS
memory_stores	None	Set TAGSTORES bit in the ESCR corresponding to event Uop_Type.	NBOGUS

NOTES:

1. There may be some undercounting of front end events when there is an overflow or underflow of the floating point stack.

Table A-11. List of Metrics Available for Execution Tagging (For Execution Event Only)

Execution metric	Upstream ESCR	TagValue in Upstream ESCR	Event mask value for execution_event
packed_SP_retired	Set ALL bit in event mask, TagUop bit in ESCR of packed_SP_uop.	1	NBOGUS0
packed_DP_retired	Set ALL bit in event mask, TagUop bit in ESCR of packed_DP_uop.	1	NBOGUS0
scalar_SP_retired	Set ALL bit in event mask, TagUop bit in ESCR of scalar_SP_uop.	1	NBOGUS0
scalar_DP_retired	Set ALL bit in event mask, TagUop bit in ESCR of scalar_DP_uop.	1	NBOGUS0
128_bit_MMX_retired	Set ALL bit in event mask, TagUop bit in ESCR of 128_bit_MMX_uop.	1	NBOGUS0

Table A-11. List of Metrics Available for Execution Tagging (For Execution Event Only) (Contd.)

Execution metric	Upstream ESCR	TagValue in Upstream ESCR	Event mask value for execution_event
64_bit_MMX_retired	Set ALL bit in event mask, TagUop bit in ESCR of 64_bit_MMX_uop.	1	NBOGUS0
X87_FP_retired	Set ALL bit in event mask, TagUop bit in ESCR of x87_FP_uop.	1	NBOGUS0
X87_SIMD_memory_moves_retired	Set ALLP0, ALLP2 bits in event mask, TagUop bit in ESCR of X87_SIMD_ moves_uop.	1	NBOGUS0

Table A-12. List of Metrics Available for Replay Tagging (For Replay Event Only)

Replay metric[1]	IA32_PEBS_ ENABLE Field to Set	MSR_PEBS_ MATRIX_VERT Bit Field to Set	Additional MSR/ Event	Event Mask Value for Replay_event
1stL_cache_load _miss_retired	Bit 0, Bit 24, Bit 25	Bit 0	None	NBOGUS
2ndL_cache_load _miss_retired[2]	Bit 1, Bit 24, Bit 25	Bit 0	None	NBOGUS
DTLB_load_miss _retired	Bit 2, Bit 24, Bit 25	Bit 0	None	NBOGUS
DTLB_store_miss _retired	Bit 2, Bit 24, Bit 25	Bit 1	None	NBOGUS
DTLB_all_miss _retired	Bit 2, Bit 24, Bit 25	Bit 0, Bit 1	None	NBOGUS
Tagged_mispred_ branch	Bit 15, Bit 16, Bit 24, Bit 25	Bit 4	None	NBOGUS
MOB_load _replay_retired[3]	Bit 9, Bit 24, Bit 25	Bit 0	Select MOB_load_replay event and set PARTIAL_DATA and UNALGN_ADDR bit.	NBOGUS

Table A-12. List of Metrics Available for Replay Tagging (For Replay Event Only) (Contd.)

Replay metric[1]	IA32_PEBS_ ENABLE Field to Set	MSR_PEBS_ MATRIX_VERT Bit Field to Set	Additional MSR/ Event	Event Mask Value for Replay_event
split_load_retired	Bit 10, Bit 24, Bit 25	Bit 0	Select load_port_replay event with the MSR_SAAT_ESCR1 MSR and set the SPLIT_LD mask bit.	NBOGUS
split_store_retired	Bit 10, Bit 24, Bit 25	Bit 1	Select store_port_replay event with the MSR_SAAT_ESCR0 MSR and set the SPLIT_ST mask bit.	NBOGUS

NOTES:

1. Certain kinds of μops cannot be tagged. These include I/O operations, UC and locked accesses, returns, and far transfers.
2. 2nd-level misses retired does not count all 2nd-level misses. It only includes those references that are found to be misses by the fast detection logic and not those that are later found to be misses.
3. While there are several causes for a MOB replay, the event counted with this event mask setting is the case where the data from a load that would otherwise be forwarded is not an aligned subset of the data from a preceding store.

Table A-13. Event Mask Qualification for Logical Processors

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
Non-Retirement	BPU_fetch_request	Bit 0: TCMISS	TS
Non-Retirement	BSQ_allocation	Bit 0: REQ_TYPE0	TS
		1: REQ_TYPE1	TS
		2: REQ_LEN0	TS
		3: REQ_LEN1	TS
		5: REQ_IO_TYPE	TS
		6: REQ_LOCK_TYPE	TS
		7: REQ_CACHE_TYPE	TS
		8: REQ_SPLIT_TYPE	TS
		9: REQ_DEM_TYPE	TS
		10: REQ_ORD_TYPE	TS
		11: MEM_TYPE0	TS
		12: MEM_TYPE1	TS
		13: MEM_TYPE2	TS
Non-Retirement	BSQ_cache_reference	Bit 0: RD_2ndL_HITS	TS
		1: RD_2ndL_HITE	TS
		2: RD_2ndL_HITM	TS
		3: RD_3rdL_HITS	TS
		4: RD_3rdL_HITE	TS
		5: RD_3rdL_HITM	TS
		6: WR_2ndL_HIT	TS
		7: WR_3rdL_HIT	TS
		8: RD_2ndL_MISS	TS
		9: RD_3rdL_MISS	TS
		10: WR_2ndL_MISS	TS
		11: WR_3rdL_MISS	TS

Table A-13. Event Mask Qualification for Logical Processors (Contd.)

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
Non-Retirement	memory_cancel	Bit 2: ST_RB_FULL 3: 64K_CONF	TS TS
Non-Retirement	SSE_input_assist	Bit 15: ALL	TI
Non-Retirement	64bit_MMX_uop	Bit 15: ALL	TI
Non-Retirement	packed_DP_uop	Bit 15: ALL	TI
Non-Retirement	packed_SP_uop	Bit 15: ALL	TI
Non-Retirement	scalar_DP_uop	Bit 15: ALL	TI
Non-Retirement	scalar_SP_uop	Bit 15: ALL	TI
Non-Retirement	128bit_MMX_uop	Bit 15: ALL	TI
Non-Retirement	x87_FP_uop	Bit 15: ALL	TI
Non-Retirement	x87_SIMD_moves_uop	Bit 3: ALLP0 4: ALLP2	TI TI
Non-Retirement	FSB_data_activity	Bit 0: DRDY_DRV 1: DRDY_OWN 2: DRDY_OTHER 3: DBSY_DRV 4: DBSY_OWN 5: DBSY_OTHER	TI TI TI TI TI TI
Non-Retirement	IOQ_allocation	Bit 0: ReqA0 1: ReqA1 2: ReqA2 3: ReqA3 4: ReqA4 5: ALL_READ 6: ALL_WRITE 7: MEM_UC 8: MEM_WC	TS TS TS TS TS TS TS TS TS

Table A-13. Event Mask Qualification for Logical Processors (Contd.)

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
		9: MEM_WT	TS
		10: MEM_WP	TS
		11: MEM_WB	TS
		13: OWN	TS
		14: OTHER	TS
		15: PREFETCH	TS
Non-Retirement	IOQ_active_entries	Bit	TS
		0: ReqA0	
		1:ReqA1	TS
		2: ReqA2	TS
		3: ReqA3	TS
		4: ReqA4	TS
		5: ALL_READ	TS
		6: ALL_WRITE	TS
		7: MEM_UC	TS
		8: MEM_WC	TS
		9: MEM_WT	TS
		10: MEM_WP	TS
		11: MEM_WB	TS
		13: OWN	TS
		14: OTHER	TS
		15: PREFETCH	TS
Non-Retirement	global_power_events	Bit 0: RUNNING	TS
Non-Retirement	ITLB_reference	Bit	
		0: HIT	TS
		1: MISS	TS
		2: HIT_UC	TS

Table A-13. Event Mask Qualification for Logical Processors (Contd.)

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
Non-Retirement	MOB_load_replay	Bit 1: NO_STA 3: NO_STD 4: PARTIAL_DATA 5: UNALGN_ADDR	 TS TS TS TS
Non-Retirement	page_walk_type	Bit 0: DTMISS 1: ITMISS	 TI TI
Non-Retirement	uop_type	Bit 1: TAGLOADS	 TS
		2: TAGSTORES	TS
Non-Retirement	load_port_replay	Bit 1: SPLIT_LD	TS
Non-Retirement	store_port_replay	Bit 1: SPLIT_ST	TS
Non-Retirement	memory_complete	Bit 0: LSC 1: SSC 2: USC 3: ULC	 TS TS TS TS
Non-Retirement	retired_mispred_branch_type	Bit 0: UNCONDITIONAL 1: CONDITIONAL 2: CALL 3: RETURN 4: INDIRECT	 TS TS TS TS TS
Non-Retirement	retired_branch_type	Bit 0: UNCONDITIONAL 1: CONDITIONAL 2: CALL 3: RETURN 4: INDIRECT	 TS TS TS TS TS

Table A-13. Event Mask Qualification for Logical Processors (Contd.)

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
Non-Retirement	tc_ms_xfer	Bit	
		0: CISC	TS
Non-Retirement	tc_misc	Bit	
		4: FLUSH	TS
Non-Retirement	TC_deliver_mode	Bit	
		0: DD	TI
		1: DB	TI
		2: DI	TI
		3: BD	TI
		4: BB	TI
		5: BI	TI
		6: ID	TI
		7: IB	TI
Non-Retirement	uop_queue_writes	Bit	
		0: FROM_TC_BUILD	TS
		1: FROM_TC_DELIVER	TS
		2: FROM_ROM	TS
Non-Retirement	resource_stall	Bit 5: SBFULL	TS
Non-Retirement	WC_Buffer	Bit	TI
		0: WCB_EVICTS	TI
		1: WCB_FULL_EVICT	TI
		2: WCB_HITM_EVICT	TI
At Retirement	instr_retired	Bit	
		0: NBOGUSNTAG	TS
		1: NBOGUSTAG	TS
		2: BOGUSNTAG	TS
		3: BOGUSTAG	TS

Table A-13. Event Mask Qualification for Logical Processors (Contd.)

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
At Retirement	machine_clear	Bit	
		0: CLEAR	TS
		2: MOCLEAR	TS
		6: SMCCLEAR	TS
At Retirement	front_end_event	Bit	
		0: NBOGUS	TS
		1: BOGUS	TS
At Retirement	replay_event	Bit	
		0: NBOGUS	TS
		1: BOGUS	TS
At Retirement	execution_event	Bit	
		0: NONBOGUS0	TS
		1: NONBOGUS1	TS
		2: NONBOGUS2	TS
		3: NONBOGUS3	TS
		4: BOGUS0	TS
		5: BOGUS1	TS
		6: BOGUS2	TS
		7: BOGUS3	TS
At Retirement	x87_assist	Bit	
		0: FPSU	TS
		1: FPSO	TS
		2: POAO	TS
		3: POAU	TS
		4: PREA	TS
At Retirement	branch_retired	Bit	
		0: MMNP	TS
		1: MMNM	TS
		2: MMTP	TS
		3: MMTM	TS
At Retirement	mispred_branch_retired	Bit 0: NBOGUS	TS

Table A-13. Event Mask Qualification for Logical Processors (Contd.)

Event Type	Event Name	Event Masks, ESCR[24:9]	TS or TI
At Retirement	uops_retired	Bit 0: NBOGUS 1: BOGUS	 TS TS
At Retirement	instr_completed	Bit 0: NBOGUS 1: BOGUS	 TS TS

A.7 PERFORMANCE MONITORING EVENTS FOR INTEL® PENTIUM® M PROCESSORS

The Pentium M processor's performance-monitoring events are based on monitoring events for the P6 family of processors. All of these performance events are model specific for the Pentium M processor and are not available in this form in other processors. Table A-14 lists the Performance-Monitoring events that were added in the Pentium M processor.

Table A-14. Performance Monitoring Events on Intel® Pentium® M Processors

Name	Hex Values	Descriptions
Power Management		

EMON_EST_TRANS	58H	Number of Enhanced Intel SpeedStep technology transitions:
		Mask = 00H - All transitions
		Mask = 02H - Only Frequency transitions

EMON_THERMAL_TRIP	59H	Duration/Occurrences in thermal trip; to count number of thermal trips: bit 22 in PerfEvtSel0/1 needs to be set to enable edge detect.
BPU		
BR_INST_EXEC	88H	Branch instructions that were executed (not necessarily retired).
BR_MISSP_EXEC	89H	Branch instructions executed that were mispredicted at execution.

Table A-14. Performance Monitoring Events on Intel® Pentium® M Processors (Contd.)

Name	Hex Values	Descriptions
BR_BAC_MISSP_EXEC	8AH	Branch instructions executed that were mispredicted at front end (BAC).
BR_CND_EXEC	8BH	Conditional branch instructions that were executed.
BR_CND_MISSP_EXEC	8CH	Conditional branch instructions executed that were mispredicted.
BR_IND_EXEC	8DH	Indirect branch instructions executed.
BR_IND_MISSP_EXEC	8EH	Indirect branch instructions executed that were mispredicted.
BR_RET_EXEC	8FH	Return branch instructions executed.
BR_RET_MISSP_EXEC	90H	Return branch instructions executed that were mispredicted at execution.
BR_RET_BAC_MISSP_EXEC	91H	Return branch instructions executed that were mispredicted at front end (BAC).
BR_CALL_EXEC	92H	CALL instruction executed.
BR_CALL_MISSP_EXEC	93H	CALL instruction executed and miss predicted.
BR_IND_CALL_EXEC	94H	Indirect CALL instructions executed.
Decoder		
EMON_SIMD_INSTR_RETIRED	CEH	Number of retired MMX instructions.
EMON_SYNCH_UOPS	D3H	Sync micro-ops
EMON_ESP_UOPS	D7H	Total number of micro-ops

EMON_FUSED_UOPS_RET	DAH	Number of retired fused micro-ops:
		Mask = 0 - Fused micro-ops
		Mask = 1 - Only load+Op micro-ops
		Mask = 2 - Only std+sta micro-ops

EMON_UNFUSION	DBH	Number of unfusion events in the ROB, happened on a FP exception to a fused µop.

Table A-14. Performance Monitoring Events on Intel® Pentium® M Processors (Contd.)

Name	Hex Values	Descriptions
Prefetcher		
EMON_PREF_RQSTS_UP	F0H	Number of upward prefetches issued
EMON_PREF_RQSTS_DN	F8H	Number of downward prefetches issued

A number of P6 family processor performance monitoring events are modified for the Pentium M processor. Table A-15 lists the performance monitoring events that were changed in the Pentium M processor, and differ from performance monitoring events for the P6 family of processors.

Table A-15. Performance Monitoring Events Modified on Intel® Pentium® M Processors

Name	Hex Values	Descriptions
CPU_CLK_UNHALTED	79H	Number of cycles during which the processor is not halted, and not in a thermal trip.

EMON_SSE_SSE2_INST_ RETIRED	D8H	Streaming SIMD Extensions Instructions Retired:
		Mask = 0 - SSE packed single and scalar single
		Mask = 1 - SSE scalar-single
		Mask = 2 - SSE2 packed-double
		Mask = 3 - SSE2 scalar-double

EMON_SSE_SSE2_COMP_INST_ RETIRED	D9H	Computational SSE Instructions Retired:
		Mask = 0 – SSE packed single
		Mask = 1 – SSE Scalar-single
		Mask = 2 – SSE2 packed-double
		Mask = 3 – SSE2 scalar-double

Table A-15. Performance Monitoring Events Modified on Intel® Pentium® M Processors (Contd.)

Name	Hex Values	Descriptions	

L2_LD	29H	L2 data loads	Mask[0] = 1 – count I state lines
			Mask[1] = 1 – count S state lines
			Mask[2] = 1 – count E state lines
			Mask[3] = 1 – count M state lines
			Mask[5:4]:
			00H – Excluding hardware-prefetched lines
			01H - Hardware-prefetched lines only
			02H/03H – All (HW-prefetched lines and non HW --Prefetched lines)

L2_LINES_IN	24H	L2 lines allocated	
L2_LINES_OUT	26H	L2 lines evicted	
L2_M_LINES_OUT	27H	Lw M-state lines evicted	

A.8 P6 FAMILY PROCESSOR PERFORMANCE-MONITORING EVENTS

Table A-16 lists the events that can be counted with the performance-monitoring counters and read with the RDPMC instruction for the P6 family processors. The unit column gives the microarchitecture or bus unit that produces the event; the event number column gives the hexadecimal number identifying the event; the mnemonic event name column gives the name of the event; the unit mask column gives the unit mask required (if any); the description column describes the event; and the comments column gives additional information about the event.

All of these performance events are model specific for the P6 family processors and are not available in this form in the Pentium 4 processors or the Pentium processors. Some events (such as those added in later generations of the P6 family processors) are only available in specific processors in the P6 family. All performance event encodings not listed in Table A-16 are reserved and their use will result in undefined counter results.

See the end of the table for notes related to certain entries in the table.

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
Data Cache Unit (DCU)	43H	DATA_MEM_REFS	00H	All loads from any memory type. All stores to any memory type. Each part of a split is counted separately. The internal logic counts not only memory loads and stores, but also internal retries. 80-bit floating-point accesses are double counted, since they are decomposed into a 16-bit exponent load and a 64-bit mantissa load. Memory accesses are only counted when they are actually performed (such as a load that gets squashed because a previous cache miss is outstanding to the same address, and which finally gets performed, is only counted once). Does not include I/O accesses, or other nonmemory accesses.	
	45H	DCU_LINES_IN	00H	Total lines allocated in DCU	
	46H	DCU_M_LINES_IN	00H	Number of M state lines allocated in DCU	
	47H	DCU_M_LINES_OUT	00H	Number of M state lines evicted from DCU This includes evictions via snoop HITM, intervention or replacement.	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
	48H	DCU_MISS_ OUTSTANDING	00H	Weighted number of cycles while a DCU miss is outstanding, incremented by the number of outstanding cache misses at any particular time. Cacheable read requests only are considered. Uncacheable requests are excluded. Read-for-ownerships are counted, as well as line fills, invalidates, and stores.	An access that also misses the L2 is short-changed by 2 cycles (i.e., if counts N cycles, should be N+2 cycles). Subsequent loads to the same cache line will not result in any additional counts. Count value not precise, but still useful.
Instruction Fetch Unit (IFU)	80H	IFU_IFETCH	00H	Number of instruction fetches, both cacheable and noncacheable, including UC fetches	
	81H	IFU_IFETCH_ MISS	00H	Number of instruction fetch misses All instruction fetches that do not hit the IFU (i.e., that produce memory requests). This includes UC accesses.	
	85H	ITLB_MISS	00H	Number of ITLB misses.	
	86H	IFU_MEM_STALL	00H	Number of cycles instruction fetch is stalled, for any reason. Includes IFU cache misses, ITLB misses, ITLB faults, and other minor stalls.	
	87H	ILD_STALL	00H	Number of cycles that the instruction length decoder is stalled.	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
L2 Cache[1]	28H	L2_IFETCH	MESI 0FH	Number of L2 instruction fetches. This event indicates that a normal instruction fetch was received by the L2. The count includes only L2 cacheable instruction fetches; it does not include UC instruction fetches. It does not include ITLB miss accesses.	
	29H	L2_LD	MESI 0FH	Number of L2 data loads. This event indicates that a normal, unlocked, load memory access was received by the L2. It includes only L2 cacheable memory accesses; it does not include I/O accesses, other nonmemory accesses, or memory accesses such as UC/WT memory accesses. It does include L2 cacheable TLB miss memory accesses.	
	2AH	L2_ST	MESI 0FH	Number of L2 data stores. This event indicates that a normal, unlocked, store memory access was received by the L2.	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
				it indicates that the DCU sent a read-for-ownership request to the L2. It also includes Invalid to Modified requests sent by the DCU to the L2. It includes only L2 cacheable memory accesses; it does not include I/O accesses, other nonmemory accesses, or memory accesses such as UC/WT memory accesses. It includes TLB miss memory accesses.	
	24H	L2_LINES_IN	00H	Number of lines allocated in the L2.	
	26H	L2_LINES_OUT	00H	Number of lines removed from the L2 for any reason.	
	25H	L2_M_LINES_INM	00H	Number of modified lines allocated in the L2.	
	27H	L2_M_LINES_ OUTM	00H	Number of modified lines removed from the L2 for any reason.	
	2EH	L2_RQSTS	MESI 0FH	Total number of L2 requests.	
	21H	L2_ADS	00H	Number of L2 address strobes.	
	22H	L2_DBUS_BUSY	00H	Number of cycles during which the L2 cache data bus was busy.	
	23H	L2_DBUS_BUSY_ RD	00H	Number of cycles during which the data bus was busy transferring read data from L2 to the processor.	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
External Bus Logic (EBL)[2]	62H	BUS_DRDY_ CLOCKS	00H (Self) 20H (Any)	Number of clocks during which DRDY# is asserted. Utilization of the external system data bus during data transfers.	Unit Mask = 00H counts bus clocks when the processor is driving DRDY#. Unit Mask = 20H counts in processor clocks when any agent is driving DRDY#.
	63H	BUS_LOCK_ CLOCKS	00H (Self) 20H (Any)	Number of clocks during which LOCK# is asserted on the external system bus.[3]	Always counts in processor clocks.
	60H	BUS_REQ_ OUTSTANDING	00H (Self)	Number of bus requests outstanding. This counter is incremented by the number of cacheable read bus requests outstanding in any given cycle.	Counts only DCU full-line cacheable reads, not RFOs, writes, instruction fetches, or anything else. Counts “waiting for bus to complete” (last data chunk received).
	65H	BUS_TRAN_BRD	00H (Self) 20H (Any)	Number of burst read transactions.	
	66H	BUS_TRAN_RFO	00H (Self) 20H (Any)	Number of completed read for ownership transactions.	
	67H	BUS_TRANS_WB	00H (Self) 20H (Any)	Number of completed write back transactions.	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
	68H	BUS_TRAN_ IFETCH	00H (Self) 20H (Any)	Number of completed instruction fetch transactions.	
	69H	BUS_TRAN_INVA L	00H (Self) 20H (Any)	Number of completed invalidate transactions.	
	6AH	BUS_TRAN_PWR	00H (Self) 20H (Any)	Number of completed partial write transactions.	
	6BH	BUS_TRANS_P	00H (Self) 20H (Any)	Number of completed partial transactions.	
	6CH	BUS_TRANS_IO	00H (Self) 20H (Any)	Number of completed I/O transactions.	
	6DH	BUS_TRAN_DEF	00H (Self) 20H (Any)	Number of completed deferred transactions.	
	6EH	BUS_TRAN_ BURST	00H (Self) 20H (Any)	Number of completed burst transactions.	
	70H	BUS_TRAN_ANY	00H (Self) 20H (Any)	Number of all completed bus transactions. Address bus utilization can be calculated knowing the minimum address bus occupancy. Includes special cycles, etc.	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
	6FH	BUS_TRAN_MEM	00H (Self) 20H (Any)	Number of completed memory transactions.	
	64H	BUS_DATA_RCV	00H (Self)	Number of bus clock cycles during which this processor is receiving data.	
	61H	BUS_BNR_DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the BNR# pin.	
	7AH	BUS_HIT_DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HIT# pin.	Includes cycles due to snoop stalls. The event counts correctly, but BPM*i* (breakpoint monitor) pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers): ▪ If the core-clock-to- bus-clock ratio is 2:1 or 3:1, and a PC bit is set, the BPM*i* pins will be asserted for a single clock when the counters overflow.

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
					▪ If the PC bit is clear, the processor toggles the BPM*i* pins when the counter overflows. ▪ If the clock ratio is not 2:1 or 3:1, the BPM*i* pins will not function for these performance-monitoring counter events.
	7BH	BUS_HITM_DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HITM# pin.	Includes cycles due to snoop stalls. The event counts correctly, but BPM*i* (breakpoint monitor) pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers): ▪ If the core-clock-to- bus-clock ratio is 2:1 or 3:1, and a PC bit is set, the BPM*i* pins will be asserted for a single clock when the counters overflow.

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
					▪ If the PC bit is clear, the processor toggles the BPM*i*pins when the counter overflows. ▪ If the clock ratio is not 2:1 or 3:1, the BPM*i* pins will not function for these performance-monitoring counter events.
	7EH	BUS_SNOOP_ STALL	00H (Self)	Number of clock cycles during which the bus is snoop stalled.	
Floating-Point Unit	C1H	FLOPS	00H	Number of computational floating-point operations retired. Excludes floating-point computational operations that cause traps or assists. Includes floating-point computational operations executed by the assist handler. Includes internal sub-operations for complex floating-point instructions like transcendentals. Excludes floating-point loads and stores.	Counter 0 only.

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
	10H	FP_COMP_OPS_ EXE	00H	Number of computational floating-point operations executed. The number of FADD, FSUB, FCOM, FMULs, integer MULs and IMULs, FDIVs, FPREMs, FSQRTS, integer DIVs, and IDIVs. This number does not include the number of cycles, but the number of operations. This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.	Counter 0 only.
	11H	FP_ASSIST	00H	Number of floating-point exception cases handled by microcode.	Counter 1 only. This event includes counts due to speculative execution.
	12H	MUL	00H	Number of multiplies. This count includes integer as well as FP multiplies and is speculative.	Counter 1 only.
	13H	DIV	00H	Number of divides. This count includes integer as well as FP divides and is speculative.	Counter 1 only.

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
	14H	CYCLES_DIV_BUSY	00H	Number of cycles during which the divider is busy, and cannot accept new divides. This includes integer and FP divides, FPREM, FPSQRT, etc. and is speculative.	Counter 0 only.
Memory Ordering	03H	LD_BLOCKS	00H	Number of load operations delayed due to store buffer blocks. Includes counts caused by preceding stores whose addresses are unknown, preceding stores whose addresses are known but whose data is unknown, and preceding stores that conflicts with the load but which incompletely overlap the load.	
	04H	SB_DRAINS	00H	Number of store buffer drain cycles. Incremented every cycle the store buffer is draining. Draining is caused by serializing operations like CPUID, synchronizing operations like XCHG, interrupt acknowledgment, as well as other conditions (such as cache flushing).	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
	05H	MISALIGN_MEM_REF	00H	Number of misaligned data memory references. Incremented by 1 every cycle, during which either the processor's load or store pipeline dispatches a misaligned µop. Counting is performed if it is the first or second half, or if it is blocked, squashed, or missed. In this context, misaligned means crossing a 64-bit boundary.	MISALIGN_MEM_REF is only an approximation to the true number of misaligned memory references. The value returned is roughly proportional to the number of misaligned memory accesses (the size of the problem).
	07H	EMON_KNI_PREF_DISPATCHED		Number of Streaming SIMD extensions prefetch/weakly-ordered instructions dispatched (speculative prefetches are included in counting):	Counters 0 and 1. Pentium III processor only.
			00H	0: prefetch NTA	
			01H	1: prefetch T1	
			02H	2: prefetch T2	
			03H	3: weakly ordered stores	
	4BH	EMON_KNI_PREF_MISS		Number of prefetch/weakly-ordered instructions that miss all caches:	Counters 0 and 1. Pentium III processor only.
			00H	0: prefetch NTA	
			01H	1: prefetch T1	
			02H	2: prefetch T2	
			03H	3: weakly ordered stores	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
Instruction Decoding and Retirement	C0H	INST_RETIRED	00H	Number of instructions retired.	A hardware interrupt received during/after the last iteration of the REP STOS flow causes the counter to undercount by 1 instruction. An SMI received while executing a HLT instruction will cause the performance counter to not count the RSM instruction and undercount by 1.
	C2H	UOPS_RETIRED	00H	Number of µops retired.	
	D0H	INST_DECODED	00H	Number of instructions decoded.	
	D8H	EMON_KNI_INST_RETIRED	 00H 01H	Number of Streaming SIMD extensions retired: 0: packed & scalar 1: scalar	Counters 0 and 1. Pentium III processor only.
	D9H	EMON_KNI_COMP_INST_RET	 00H 01H	Number of Streaming SIMD extensions computation instructions retired: 0: packed and scalar 1: scalar	Counters 0 and 1. Pentium III processor only.

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
Interrupts	C8H	HW_INT_RX	00H	Number of hardware interrupts received.	
	C6H	CYCLES_INT_ MASKED	00H	Number of processor cycles for which interrupts are disabled.	
	C7H	CYCLES_INT_ PENDING_ AND_MASKED	00H	Number of processor cycles for which interrupts are disabled and interrupts are pending.	
Branches	C4H	BR_INST_ RETIRED	00H	Number of branch instructions retired.	
	C5H	BR_MISS_PRED_ RETIRED	00H	Number of mispredicted branches retired.	
	C9H	BR_TAKEN_ RETIRED	00H	Number of taken branches retired.	
	CAH	BR_MISS_PRED_ TAKEN_RET	00H	Number of taken mispredictions branches retired.	
	E0H	BR_INST_ DECODED	00H	Number of branch instructions decoded.	
	E2H	BTB_MISSES	00H	Number of branches for which the BTB did not produce a prediction.	
	E4H	BR_BOGUS	00H	Number of bogus branches.	
	E6H	BACLEARS	00H	Number of times BACLEAR is asserted. This is the number of times that a static branch prediction was made, in which the branch decoder decided to make a branch prediction because the BTB did not.	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
Stalls	A2H	RESOURCE_ STALLS	00H	Incremented by 1 during every cycle for which there is a resource related stall. Includes register renaming buffer entries, memory buffer entries. Does not include stalls due to bus queue full, too many cache misses, etc. In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, such as if retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	
	D2H	PARTIAL_RAT_ STALLS	00H	Number of cycles or events for partial stalls. This includes flag partial stalls.	
Segment Register Loads	06H	SEGMENT_REG_ LOADS	00H	Number of segment register loads.	
Clocks	79H	CPU_CLK_ UNHALTED	00H	Number of cycles during which the processor is not halted.	

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
MMX Unit	B0H	MMX_INSTR_EXEC	00H	Number of MMX Instructions Executed.	Available in Intel Celeron, Pentium II and Pentium II Xeon processors only. Does not account for MOVQ and MOVD stores from register to memory.
	B1H	MMX_SAT_INSTR_EXEC	00H	Number of MMX Saturating Instructions Executed.	Available in Pentium II and Pentium III processors only.
	B2H	MMX_UOPS_EXEC	0FH	Number of MMX µops Executed.	Available in Pentium II and Pentium III processors only.
	B3H	MMX_INSTR_TYPE_EXEC	01H 02H 04H 08H 10H 20H	MMX packed multiply instructions executed. MMX packed shift instructions executed. MMX pack operation instructions executed. MMX unpack operation instructions executed. MMX packed logical instructions executed. MMX packed arithmetic instructions executed.	Available in Pentium II and Pentium III processors only.
	CCH	FP_MMX_TRANS	00H 01H	Transitions from MMX instruction to floating-point instructions. Transitions from floating-point instructions to MMX instructions.	Available in Pentium II and Pentium III processors only.
	CDH	MMX_ASSIST	00H	Number of MMX Assists (that is, the number of EMMS instructions executed).	Available in Pentium II and Pentium III processors only.
	CEH	MMX_INSTR_RET	00H	Number of MMX Instructions Retired.	Available in Pentium II processors only.

Table A-16. Events That Can Be Counted with the P6 Family Performance-Monitoring Counters (Contd.)

Unit	Event Num.	Mnemonic Event Name	Unit Mask	Description	Comments
Segment Register Renaming	D4H	SEG_RENAME_ STALLS		Number of Segment Register Renaming Stalls:	Available in Pentium II and Pentium III processors only.
			02H	Segment register ES	
			04H	Segment register DS	
			08H	Segment register FS	
			0FH	Segment register FS	
				Segment registers ES + DS + FS + GS	
	D5H	SEG_REG_ RENAMES		Number of Segment Register Renames:	Available in Pentium II and Pentium III processors only.
			01H	Segment register ES	
			02H	Segment register DS	
			04H	Segment register FS	
			08H	Segment register FS	
			0FH	Segment registers ES + DS + FS + GS	
	D6H	RET_SEG_ RENAMES	00H	Number of segment register rename events retired.	Available in Pentium II and Pentium III processors only.

NOTES:

1. Several L2 cache events, where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. The lower 4 bits of the Unit Mask field are used in conjunction with L2 events to indicate the cache state or cache states involved.

   The P6 family processors identify cache states using the “MESI” protocol and consequently each bit in the Unit Mask field represents one of the four states: UMSK[3] = M (8H) state, UMSK[2] = E (4H) state, UMSK[1] = S (2H) state, and UMSK[0] = I (1H) state. UMSK[3:0] = MESI” (FH) should be used to collect data for all states; UMSK = 0H, for the applicable events, will result in nothing being counted.

2. All of the external bus logic (EBL) events, except where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers.

   Bit 5 of the UMSK field is used in conjunction with the EBL events to indicate whether the processor should count transactions that are self- generated (UMSK[5] = 0) or transactions that result from any processor on the bus (UMSK[5] = 1).

3. L2 cache locks, so it is possible to have a zero count.

A.9 PENTIUM PROCESSOR PERFORMANCE-MONITORING EVENTS

Table A-17 lists the events that can be counted with the performance-monitoring counters for the Pentium processor. The Event Number column gives the hexadecimal code that identifies the event and that is entered in the ES0 or ES1 (event select) fields of the CESR MSR. The Mnemonic Event Name column gives the name of the event, and the Description and Comments columns give detailed descriptions of the events. Most events can be counted with either counter 0 or counter 1; however, some events can only be counted with only counter 0 or only counter 1 (as noted).

NOTE

The events in the table that are shaded are implemented only in the Pentium processor with MMX technology.

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters

Event Num.	Mnemonic Event Name	Description	Comments
00H	DATA_READ	Number of memory data reads (internal data cache hit and miss combined)	Split cycle reads are counted individually. Data Memory Reads that are part of TLB miss processing are not included. These events may occur at a maximum of two per clock. I/O is not included.
01H	DATA_WRITE	Number of memory data writes (internal data cache hit and miss combined); I/O not included	Split cycle writes are counted individually. These events may occur at a maximum of two per clock. I/O is not included.
0H2	DATA_TLB_MISS	Number of misses to the data cache translation look-aside buffer	

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
03H	DATA_READ_MISS	Number of memory read accesses that miss the internal data cache whether or not the access is cacheable or noncacheable	Additional reads to the same cache line after the first BRDY# of the burst line fill is returned but before the final (fourth) BRDY# has been returned, will not cause the counter to be incremented additional times. Data accesses that are part of TLB miss processing are not included. Accesses directed to I/O space are not included.
04H	DATA WRITE MISS	Number of memory write accesses that miss the internal data cache whether or not the access is cacheable or noncacheable	Data accesses that are part of TLB miss processing are not included. Accesses directed to I/O space are not included.
05H	WRITE_HIT_TO_ M-_OR_E- STATE_LINES	Number of write hits to exclusive or modified lines in the data cache	These are the writes that may be held up if EWBE# is inactive. These events may occur a maximum of two per clock.
06H	DATA_CACHE_ LINES_ WRITTEN_BACK	Number of dirty lines (all) that are written back, regardless of the cause	Replacements and internal and external snoops can all cause writeback and are counted.
07H	EXTERNAL_ SNOOPS	Number of accepted external snoops whether they hit in the code cache or data cache or neither	Assertions of EADS# outside of the sampling interval are not counted, and no internal snoops are counted.
08H	EXTERNAL_DATA_ CACHE_SNOOP_ HITS	Number of external snoops to the data cache	Snoop hits to a valid line in either the data cache, the data line fill buffer, or one of the write back buffers are all counted as hits.
09H	MEMORY ACCESSES IN BOTH PIPES	Number of data memory reads or writes that are paired in both pipes of the pipeline	These accesses are not necessarily run in parallel due to cache misses, bank conflicts, etc.
0AH	BANK CONFLICTS	Number of actual bank conflicts	

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
0BH	MISALIGNED DATA MEMORY OR I/O REFERENCES	Number of memory or I/O reads or writes that are misaligned	A 2- or 4-byte access is misaligned when it crosses a 4-byte boundary; an 8-byte access is misaligned when it crosses an 8-byte boundary. Ten byte accesses are treated as two separate accesses of 8 and 2 bytes each.
0CH	CODE READ	Number of instruction reads; whether the read is cacheable or noncacheable	Individual 8-byte noncacheable instruction reads are counted.
0DH	CODE TLB MISS	Number of instruction reads that miss the code TLB whether the read is cacheable or noncacheable	Individual 8-byte noncacheable instruction reads are counted.
0EH	CODE CACHE MISS	Number of instruction reads that miss the internal code cache; whether the read is cacheable or noncacheable	Individual 8-byte noncacheable instruction reads are counted.
0FH	ANY SEGMENT REGISTER LOADED	Number of writes into any segment register in real or protected mode including the LDTR, GDTR, IDTR, and TR	Segment loads are caused by explicit segment register load instructions, far control transfers, and task switches. Far control transfers and task switches causing a privilege level change will signal this event twice. Interrupts and exceptions may initiate a far control transfer.
10H	Reserved		
11H	Reserved		

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
12H	Branches	Number of taken and not taken branches, including: conditional branches, jumps, calls, returns, software interrupts, and interrupt returns	Also counted as taken branches are serializing instructions, VERR and VERW instructions, some segment descriptor loads, hardware interrupts (including FLUSH#), and programmatic exceptions that invoke a trap or fault handler. The pipe is not necessarily flushed. The number of branches actually executed is measured, not the number of predicted branches.
13H	BTB_HITS	Number of BTB hits that occur	Hits are counted only for those instructions that are actually executed.
14H	TAKEN_BRANCH_ OR_BTB_HIT	Number of taken branches or BTB hits that occur	This event type is a logical OR of taken branches and BTB hits. It represents an event that may cause a hit in the BTB. Specifically, it is either a candidate for a space in the BTB or it is already in the BTB.
15H	PIPELINE FLUSHES	Number of pipeline flushes that occur Pipeline flushes are caused by BTB misses on taken branches, mispredictions, exceptions, interrupts, and some segment descriptor loads.	The counter will not be incremented for serializing instructions (serializing instructions cause the prefetch queue to be flushed but will not trigger the Pipeline Flushed event counter) and software interrupts (software interrupts do not flush the pipeline).

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
16H	INSTRUCTIONS_ EXECUTED	Number of instructions executed (up to two per clock)	Invocations of a fault handler are considered instructions. All hardware and software interrupts and exceptions will also cause the count to be incremented. Repeat prefixed string instructions will only increment this counter once despite the fact that the repeat loop executes the same instruction multiple times until the loop criteria is satisfied. This applies to all the Repeat string instruction prefixes (i.e., REP, REPE, REPZ, REPNE, and REPNZ). This counter will also only increment once per each HLT instruction executed regardless of how many cycles the processor remains in the HALT state.
17H	INSTRUCTIONS_ EXECUTED_ V PIPE	Number of instructions executed in the V_pipe The event indicates the number of instructions that were paired.	This event is the same as the 16H event except it only counts the number of instructions actually executed in the V-pipe.
18H	BUS_CYCLE_ DURATION	Number of clocks while a bus cycle is in progress This event measures bus use.	The count includes HLDA, AHOLD, and BOFF# clocks.
19H	WRITE_BUFFER_ FULL_STALL_ DURATION	Number of clocks while the pipeline is stalled due to full write buffers	Full write buffers stall data memory read misses, data memory write misses, and data memory write hits to S-state lines. Stalls on I/O accesses are not included.
1AH	WAITING_FOR_ DATA_MEMORY_ READ_STALL_ DURATION	Number of clocks while the pipeline is stalled while waiting for data memory reads	Data TLB Miss processing is also included in the count. The pipeline stalls while a data memory read is in progress including attempts to read that are not bypassed while a line is being filled.

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
1BH	STALL ON WRITE TO AN E- OR M-STATE LINE	Number of stalls on writes to E- or M-state lines	
1CH	LOCKED BUS CYCLE	Number of locked bus cycles that occur as the result of the LOCK prefix or LOCK instruction, page-table updates, and descriptor table updates	Only the read portion of the locked read-modify-write is counted. Split locked cycles (SCYC active) count as two separate accesses. Cycles restarted due to BOFF# are not re-counted.
1DH	I/O READ OR WRITE CYCLE	Number of bus cycles directed to I/O space	Misaligned I/O accesses will generate two bus cycles. Bus cycles restarted due to BOFF# are not re-counted.
1EH	NONCACHEABLE_ MEMORY_READS	Number of noncacheable instruction or data memory read bus cycles. The count includes read cycles caused by TLB misses, but does not include read cycles to I/O space.	Cycles restarted due to BOFF# are not re-counted.
1FH	PIPELINE_AGI_ STALLS	Number of address generation interlock (AGI) stalls An AGI occurring in both the U- and V- pipelines in the same clock signals this event twice.	An AGI occurs when the instruction in the execute stage of either of U- or V-pipelines is writing to either the index or base address register of an instruction in the D2 (address generation) stage of either the U- or V- pipelines.
20H	Reserved		
21H	Reserved		

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
22H	FLOPS	Number of floating-point operations that occur	Number of floating-point adds, subtracts, multiplies, divides, remainders, and square roots are counted. The transcendental instructions consist of multiple adds and multiplies and will signal this event multiple times. Instructions generating the divide-by-zero, negative square root, special operand, or stack exceptions will not be counted. Instructions generating all other floating-point exceptions will be counted. The integer multiply instructions and other instructions which use the x87 FPU will be counted.
23H	BREAKPOINT MATCH ON DR0 REGISTER	Number of matches on register DR0 breakpoint	The counters is incremented regardless if the breakpoints are enabled or not. However, if breakpoints are not enabled, code breakpoint matches will not be checked for instructions executed in the V-pipe and will not cause this counter to be incremented. (They are checked on instruction executed in the U-pipe only when breakpoints are not enabled.) These events correspond to the signals driven on the BP[3:0] pins. Refer to Chapter 18, “Debugging and Performance Monitoring” for more information.
24H	BREAKPOINT MATCH ON DR1 REGISTER	Number of matches on register DR1 breakpoint	See comment for 23H event.
25H	BREAKPOINT MATCH ON DR2 REGISTER	Number of matches on register DR2 breakpoint	See comment for 23H event.

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
26H	BREAKPOINT MATCH ON DR3 REGISTER	Number of matches on register DR3 breakpoint	See comment for 23H event.
27H	HARDWARE INTERRUPTS	Number of taken INTR and NMI interrupts	
28H	DATA_READ_OR_ WRITE	Number of memory data reads and/or writes (internal data cache hit and miss combined)	Split cycle reads and writes are counted individually. Data Memory Reads that are part of TLB miss processing are not included. These events may occur at a maximum of two per clock. I/O is not included.
29H	DATA_READ_MISS OR_WRITE MISS	Number of memory read and/or write accesses that miss the internal data cache, whether or not the access is cacheable or noncacheable	Additional reads to the same cache line after the first BRDY# of the burst line fill is returned but before the final (fourth) BRDY# has been returned, will not cause the counter to be incremented additional times. Data accesses that are part of TLB miss processing are not included. Accesses directed to I/O space are not included.
2AH	BUS_OWNERSHIP_ LATENCY (Counter 0)	The time from LRM bus ownership request to bus ownership granted (that is, the time from the earlier of a PBREQ (0), PHITM# or HITM# assertion to a PBGNT assertion)	The ratio of the 2AH events counted on counter 0 and counter 1 is the average stall time due to bus ownership conflict.
2AH	BUS OWNERSHIP TRANSFERS (Counter 1)	The number of buss ownership transfers (that is, the number of PBREQ (0) assertions	The ratio of the 2AH events counted on counter 0 and counter 1 is the average stall time due to bus ownership conflict.
2BH	MMX_ INSTRUCTIONS_ EXECUTED_ U-PIPE (Counter 0)	Number of MMX instructions executed in the U-pipe	

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
2BH	MMX_ INSTRUCTIONS_ EXECUTED_ V-PIPE (Counter 1)	Number of MMX instructions executed in the V-pipe	
2CH	CACHE_M-STATE_LINE_ SHARING (Counter 0)	Number of times a processor identified a hit to a modified line due to a memory access in the other processor (PHITM (O))	If the average memory latencies of the system are known, this event enables the user to count the Write Backs on PHITM(O) penalty and the Latency on Hit Modified(I) penalty.
2CH	CACHE_LINE_ SHARING (Counter 1)	Number of shared data lines in the L1 cache (PHIT (O))	
2DH	EMMS_ INSTRUCTIONS_ EXECUTED (Counter 0)	Number of EMMS instructions executed	
2DH	TRANSITIONS_ BETWEEN_MMX_ AND_FP_ INSTRUCTIONS (Counter 1)	Number of transitions between MMX and floating-point instructions or vice versa An even count indicates the processor is in MMX state. an odd count indicates it is in FP state.	This event counts the first floating-point instruction following an MMX instruction or first MMX instruction following a floating-point instruction. The count may be used to estimate the penalty in transitions between floating-point state and MMX state.
2EH	BUS_UTILIZATION_ DUE_TO_ PROCESSOR_ ACTIVITY (Counter 0)	Number of clocks the bus is busy due to the processor's own activity (the bus activity that is caused by the processor)	
2EH	WRITES_TO_ NONCACHEABLE_ MEMORY (Counter 1)	Number of write accesses to noncacheable memory	The count includes write cycles caused by TLB misses and I/O write cycles. Cycles restarted due to BOFF# are not re-counted.

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
2FH	SATURATING_ MMX_ INSTRUCTIONS_ EXECUTED (Counter 0)	Number of saturating MMX instructions executed, independently of whether they actually saturated.	
2FH	SATURATIONS_ PERFORMED (Counter 1)	Number of MMX instructions that used saturating arithmetic when at least one of its results actually saturated	If an MMX instruction operating on 4 doublewords saturated in three out of the four results, the counter will be incremented by one only.
30H	NUMBER_OF_ CYCLES_NOT_IN_ HALT_STATE (Counter 0)	Number of cycles the processor is not idle due to HLT instruction	This event will enable the user to calculate "net CPI". Note that during the time that the processor is executing the HLT instruction, the Time-Stamp Counter is not disabled. Since this event is controlled by the Counter Controls CC0, CC1 it can be used to calculate the CPI at CPL=3, which the TSC cannot provide.
30H	DATA_CACHE_ TLB_MISS_ STALL_DURATION (Counter 1)	Number of clocks the pipeline is stalled due to a data cache translation look-aside buffer (TLB) miss	
31H	MMX_ INSTRUCTION_ DATA_READS (Counter 0)	Number of MMX instruction data reads	
31H	MMX_ INSTRUCTION_ DATA_READ_ MISSES (Counter 1)	Number of MMX instruction data read misses	
32H	FLOATING_POINT_S TALLS_DURATION (Counter 0)	Number of clocks while pipe is stalled due to a floating-point freeze	

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
32H	TAKEN_BRANCHES (Counter 1)	Number of taken branches	
33H	D1_STARVATION_AND_FIFO_IS_EMPTY (Counter 0)	Number of times D1 stage cannot issue ANY instructions since the FIFO buffer is empty	The D1 stage can issue 0, 1, or 2 instructions per clock if those are available in an instructions FIFO buffer.
33H	D1_STARVATION_AND_ONLY_ONE_INSTRUCTION_IN_FIFO (Counter 1)	Number of times the D1 stage issues a single instruction (since the FIFO buffer had just one instruction ready)	The D1 stage can issue 0, 1, or 2 instructions per clock if those are available in an instructions FIFO buffer. When combined with the previously defined events, Instruction Executed (16H) and Instruction Executed in the V-pipe (17H), this event enables the user to calculate the numbers of time pairing rules prevented issuing of two instructions.
34H	MMX_INSTRUCTION_DATA_WRITES (Counter 0)	Number of data writes caused by MMX instructions	
34H	MMX_INSTRUCTION_DATA_WRITE_MISSES (Counter 1)	Number of data write misses caused by MMX instructions	

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
35H	PIPELINE_ FLUSHES_DUE_ TO_WRONG_ BRANCH_ PREDICTIONS (Counter 0)	Number of pipeline flushes due to wrong branch predictions resolved in either the E-stage or the WB-stage	The count includes any pipeline flush due to a branch that the pipeline did not follow correctly. It includes cases where a branch was not in the BTB, cases where a branch was in the BTB but was mispredicted, and cases where a branch was correctly predicted but to the wrong address. Branches are resolved in either the Execute stage (E-stage) or the Writeback stage (WB-stage). In the later case, the misprediction penalty is larger by one clock. The difference between the 35H event count in counter 0 and counter 1 is the number of E-stage resolved branches.
35H	PIPELINE_ FLUSHES_DUE_ TO_WRONG_ BRANCH_ PREDICTIONS_ RESOLVED_IN_ WB-STAGE (Counter 1)	Number of pipeline flushes due to wrong branch predictions resolved in the WB-stage	See note for event 35H (Counter 0).
36H	MISALIGNED_ DATA_MEMORY_ REFERENCE_ON_ MMX_ INSTRUCTIONS (Counter 0)	Number of misaligned data memory references when executing MMX instructions	
36H	PIPELINE_ ISTALL_FOR_MMX_ INSTRUCTION_ DATA_MEMORY_ READS (Counter 1)	Number clocks during pipeline stalls caused by waits form MMX instruction data memory reads	T3:

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
37H	MISPREDICTED_ OR_ UNPREDICTED_ RETURNS (Counter 1)	Number of returns predicted incorrectly or not predicted at all	The count is the difference between the total number of executed returns and the number of returns that were correctly predicted. Only RET instructions are counted (for example, IRET instructions are not counted).
37H	PREDICTED_ RETURNS (Counter 1)	Number of predicted returns (whether they are predicted correctly and incorrectly	Only RET instructions are counted (for example, IRET instructions are not counted).
38H	MMX_MULTIPLY_ UNIT_INTERLOCK (Counter 0)	Number of clocks the pipe is stalled since the destination of previous MMX multiply instruction is not ready yet	The counter will not be incremented if there is another cause for a stall. For each occurrence of a multiply interlock, this event will be counted twice (if the stalled instruction comes on the next clock after the multiply) or by once (if the stalled instruction comes two clocks after the multiply).
38H	MOVD/MOVQ_ STORE_STALL_ DUE_TO_ PREVIOUS_MMX_ OPERATION (Counter 1)	Number of clocks a MOVD/MOVQ instruction store is stalled in D2 stage due to a previous MMX operation with a destination to be used in the store instruction.	
39H	RETURNS (Counter 0)	Number or returns executed.	Only RET instructions are counted; IRET instructions are not counted. Any exception taken on a RET instruction and any interrupt recognized by the processor on the instruction boundary prior to the execution of the RET instruction will also cause this counter to be incremented.
39H	Reserved		

Table A-17. Events That Can Be Counted with Pentium Processor Performance-Monitoring Counters (Contd.)

Event Num.	Mnemonic Event Name	Description	Comments
3AH	BTB_FALSE_ ENTRIES (Counter 0)	Number of false entries in the Branch Target Buffer	False entries are causes for misprediction other than a wrong prediction.
3AH	BTB_MISS_ PREDICTION_ON_ NOT-TAKEN_ BRANCH (Counter 1)	Number of times the BTB predicted a not-taken branch as taken	
3BH	FULL_WRITE_ BUFFER_STALL_ DURATION_ WHILE_ EXECUTING_MMX_I NSTRUCTIONS (Counter 0)	Number of clocks while the pipeline is stalled due to full write buffers while executing MMX instructions	
3BH	STALL_ON_MMX_ INSTRUCTION_ WRITE_TO E-_OR_ M-STATE_LINE (Counter 1)	Number of clocks during stalls on MMX instructions writing to E- or M-state lines	

APPENDIX B
MODEL-SPECIFIC REGISTERS (MSRS)

This appendix lists MSRs provided in Intel Core 2 processor family, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium 4 and Intel Xeon processors, P6 family processors, and Pentium processors in Tables B-5, B-10, and B-11, respectively. All MSRs listed can be read with the RDMSR and written with the WRMSR instructions.

Register addresses are given in both hexadecimal and decimal. The register name is the mnemonic register name and the bit description describes individual bits in registers.

Model specific registers and its bit-fields may be supported for a finite range of processor families/models. To distinguish between different processor family and/or models, software must use CPUID.01H leaf function to query the combination of “display family” and “display model“ to determine model-specific availability of MSRs. The “display family” and “display model“ are defined in the instruction reference pages of CPUID. Table B-1 lists the signature values of “display family” and “display model“ for various processor families or processor number series.

Table B-1. CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_17H	Intel Xeon Processor 5200, 5400 series, Intel Core 2 Quad processor Q9650.
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad, Intel Core 2 Extreme, Intel Core 2 Duo processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_1CH	Intel Atom processor
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon Processor, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon Processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon Processor MP, Intel Pentium 4 processors

Table B-1. CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon Processor, Intel Pentium III Processor
06_03H, 06_05H	Intel Pentium II Xeon Processor, Intel Pentium II Processor
06_01H	Intel Pentium Pro Processor
05_01H, 05_02H, 05_04H	Intel Pentium Processor, Intel Pentium Processor with MMX Technology

B.1 ARCHITECTURAL MSRS

Many MSRs have carried over from one generation of IA-32 processors to the next and to Intel 64 processors. A subset of MSRs and associated bit fields, which do not change on future processor generations, are now considered architectural MSRs. For historical reasons (beginning with the Pentium 4 processor), these “architectural MSRs” were given the prefix “IA32_”. Table B-2 lists the architectural MSRs, their addresses, their current names, their names in previous IA-32 processors, and bit fields that are considered architectural. MSR addresses outside Table B-2 and certain bitfields in an MSR address that may overlap with architectural MSR addresses are model-specific. Code that accesses a machine specified MSR and that is executed on a processor that does not support that MSR will generate an exception.

Architectural MSR or individual bit fields in an architectural MSR may be introduced or transitioned at the granularity of certain processor family/model or the presence of certain CPUID feature flags. The right-most column of Table B-2 provides information on the introduction of each architectural MSR or its individual fields. This information is expressed either as signature values of “DF_DM“ (see Table B-1) or via CPUID flags.

Certain bit field position may be related to the maximum physical address width, the value of which is expressed as “MAXPHYWID“ in Table B-2. “MAXPHYWID“ is reported by CPUID.8000_0008H leaf.

Table B-2. IA-32 Architectural MSRs

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
0H	0	IA32_P5_MC_ADDR (P5_MC_ADDR)	See Appendix B.8, "MSRs in Pentium Processors."	**Pentium Processor (05_01H)**
1H	1	IA32_P5_MC_TYPE (P5_MC_TYPE)	See Appendix B.8, "MSRs in Pentium Processors."	DF_DM = 05_01H
6H	6	IA32_MONITOR_FILTER_SIZE	See Section 7.11.5, "Monitor/Mwait Address Range Determination."	0F_03H
10H	16	IA32_TIME_STAMP_ COUNTER (TSC)	See Section 18.10, "Time-Stamp Counter."	05_01H
17H	23	IA32_PLATFORM_ID (MSR_PLATFORM_ID)	**Platform ID. (RO)** The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	06_01H
		49:0	Reserved.	
		52:50	**Platform Id. (RO)** Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7	
		63:53	Reserved.	
1BH	27	IA32_APIC_BASE (APIC_BASE)		06_01H
		7:0	Reserved	
		8	BSP flag (RO)	
		10:9	Reserved	
		11	APIC Global Enable (R/W)	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		(MAXPHYWID - 1):12	APIC Base (R/W)	
		63: MAXPHYWID	Reserved	
3AH	58	IA32_FEATURE_CONTROL	**Control Features in Intel 64 Processor. (R/W)**	If CPUID.01H: ECX[bit 5 or bit 6] = 1
		0	Lock bit (R/WO): (1 = locked). When set, locks this MSR from being written, writes to this bit will result in GP(0). Note: Once the Lock bit is set, the contents of this register cannot be modified. Therefore the lock bit must be set after configuring support for Intel Virtualization Technology and prior to transferring control to an option ROM or the OS. Hence, once the Lock bit is set, the entire IA32_FEATURE_CONTROL_ MSR contents are preserved across RESET when PWRGOOD is not deasserted.	If CPUID.01H:ECX[bit 5 or bit 6] = 1
		1	Enable VMX inside SMX operation (R/WL): This bit enables a system executive to use VMX in conjunction with SMX to support Intel® Trusted Execution Technology. BIOS must set this bit only when the CPUID function 1 returns VMX feature flag and SMX feature flag set (ECX bits 5 and 6 respectively).	If CPUID.01H:ECX[bit 5 and bit 6] are set to 1

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		2	Enable VMX outside SMX operation (R/WL): This bit enables VMX for system executive that do not require SMX.. BIOS must set this bit only when the CPUID function 1 returns VMX feature flag set (ECX bit 5).	If CPUID.01H:ECX[bit 5 or bit 6] = 1
		7:3	Reserved	
		14:8	SENTER Local Function Enables (R/WL): When set, each bit in the field represents an enable control for a corresponding SENTER function. This bit is supported only if CPUID.1:ECX.[bit 6] is set	If CPUID.01H:ECX[bit 6] = 1
		15	SENTER Global Enable (R/WL): This bit must be set to enable SENTER leaf functions. This bit is supported only if CPUID.1:ECX.[bit 6] is set	If CPUID.01H:ECX[bit 6] = 1
		63:16	Reserved	
79H	121	IA32_BIOS_UPDT_TRIG (BIOS_UPDT_TRIG)	BIOS Update Trigger (R/W) Executing a WRMSR instruction to this MSR causes a microcode update to be loaded into the processor. See Section 8.11.6, "Microcode Update Loader." A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
8BH	139	IA32_BIOS_SIGN_ID (BIOS_SIGN/BBL_CR_D3)	BIOS Update Signature (RO) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
		31:0	Reserved	
		63:32	It is recommended that this field be pre-loaded with 0 prior to executing CPUID. If the field remains 0 following the execution of CPUID; this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature.	
9BH	155	IA32_SMM_MONITOR_CTL	SMM Monitor Configuration (R/W)	If CPUID.01H: ECX[bit 5 or bit 6] = 1
		0	Valid (R/W)	
		11:1	Reserved	
		31:12	MSEG Base (R/W)	
		63:32	Reserved	
C1H	193	IA32_PMC0 (PERFCTR0)	General Performance Counter 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
C2H	194	IA32_PMC1 (PERFCTR1)	General Performance Counter 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
E7H	231	IA32_MPERF	Maximum Qualified Performance Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		63:0	**C0_MCNT: C0 Maximum Frequency Clock Count.** Increments at fixed interval (relative to TSC freq.) when the logical processor is in C0. Cleared upon writes to IA32_MPERF, or overflow/wrap-around of IA32_APERF.	
E8H	232	IA32_APERF	Actual Performance Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	**C0_ACNT: C0 Actual Frequency Clock Count.** Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in C0. Cleared upon writes to IA32_APERF, or overflow/wrap-around of IA32_MPERF.	
FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (RO) Section 10.11.2.1, “IA32_MTRR_DEF_TYPE MSR.”	06_01H
		7:0	VCNT: The number of variable memory type ranges in the processor	
		8	Fixed range MTRRs are supported when set.	
		9	Reserved	
		10	WC Supported when set	
		63:11	Reserved	
174H	372	IA32_SYSENTER_CS	SYSENTER_CS_MSR (R/W)	06_01H
		15:0	CS Selector	
		63:16	Reserved	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
175H	373	IA32_SYSENTER_ESP	SYSENTER_ESP_MSR (R/W)	06_01H
176H	374	IA32_SYSENTER_EIP	SYSENTER_EIP_MSR (R/W)	06_01H
179H	377	IA32_MCG_CAP (MCG_CAP)	Global Machine Check Capability (RO)	06_01H
		7:0	Count: Number of reporting banks	
		8	MCG_CTL_P: IA32_MCG_CTL is present if this bit is set	
		9	MCG_EXT_P: Extended machine check state registers are present if this bit is set	
		10	Reserved	
		11	MCG_TES_P: Threshold-based error status register are present if this bit is set.	
		15:12	Reserved	
		23:16	MCG_EXT_CNT: Number of extended machine check state registers present.	
		63:24	Reserved	
17AH	378	IA32_MCG_STATUS (MCG_STATUS)	Global Machine Check Status (RO)	06_01H
17BH	379	IA32_MCG_CTL (MCG_CTL)	Global Machine Check Control (R/W)	06_01H
180H-185H	384-389	Reserved		06_0EH[1]
186H	390	IA32_PERFEVTSEL0 (PERFEVTSEL0)	Performance Event Select Register 0 (R/W)	06_0EH
		7:0	Event Select: Selects a performance event logic unit	
		15:8	UMask: Qualifies the microarchitectural condition to detect on the selected event logic.	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		16	USR: Counts while in privilege level is not ring 0.	
		17	OS: Counts while in privilege level is ring 0.	
		18	Edge: Enables edge detection if set	
		19	PC: enables pin control	
		20	INT: enables interrupt on counter overflow	
		21	Reserved	
		22	EN: enables the corresponding performance counter to commence counting when this bit is set	
		23	INV: invert the CMASK	
		31:24	CMASK: When CMASK is not zero, the corresponding performance counter increments each cycle if the event count is greater than or equal to the CMASK.	
		63:32	Reserved	
187H	391	IA32_PERFEVTSEL1 (PERFEVTSEL1)	Performance Event Select Register 1 (R/W)	06_0EH
188H-197H	392-407	Reserved		06_0EH[2]
198H	408	IA32_PERF_STATUS	(RO)	0F_03H
		15:0	Current performance State Value	
		63:16	Reserved	
199H	409	IA32_PERF_CTL	(R/W)	0F_03H
		15:0	Target performance State Value	
		31:16	Reserved	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		32	IDA Engage. (R/W) When set to 1: disengages IDA	06_0FH (Mobile)
		63:33	Reserved	
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation Control (R/W) See Section 13.5.3, "Software Controlled Clock Modulation."	0F_0H
		0	Reserved	
		3:1	On-Demand Clock Modulation Duty Cycle: Specific encoded values for target duty cycle modulation	
		4	On-Demand Clock Modulation Enable: Set 1 to enable modulation	
		63:16	Reserved	
19BH	411	IA32_THERM_INTERRUPT	Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the processor's thermal sensors and thermal monitor. See Section 13.5.2, "Thermal Monitor."	0F_0H
		0	High-Temperature Interrupt Enable	
		1	Low-Temperature Interrupt Enable	
		2	THERMTRIP # Interrupt Enable	
		3	FORCEPR# Interrupt Enable	
		4	Overheat Interrupt Enable	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		7:5	Reserved	
		14:8	Threshold #1 Value	
		15	Threshold #1 Interrupt Enable	
		22:16	Threshold #2 Value	
		23	Threshold #2 Interrupt Enable	
		63:24	Reserved	
19CH	412	IA32_THERM_STATUS	Thermal Status Information (RO) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 13.5.2, "Thermal Monitor"	0F_0H
		0	Thermal Status (RO):	
		1	Thermal Status Log (R/W):	
		2	PROCHOT # or FORCEPR# event (RO)	
		3	PROCHOT # or FORCEPR# log (R/WC0)	
		4	Out-of-Spec Status (RO)	
		5	Out-of-Spec Status log (R/WC0)	
		6	Thermal Threshold #1 Status (RO)	If CPUID.01H:ECX[8] = 1
		7	Thermal Threshold #1 log (R/WC0)	If CPUID.01H:ECX[8] = 1

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		8	Thermal Threshold #2 Status (RO)	If CPUID.01H:ECX[8] = 1
		9	Thermal Threshold #1 log (R/WC0)	If CPUID.01H:ECX[8] = 1
		15:10	Reserved	
		22:16	Digital Readout (RO)	If CPUID.06H:EAX[0] = 1
		26:23	Reserved	
		30:27	Resolution in Degrees Celsius (RO)	If CPUID.06H:EAX[0] = 1
		31	Reading Valid (RO)	If CPUID.06H:EAX[0] = 1
		63:32	Reserved	
1A0H	416	IA32_MISC_ENABLE	**Enable Misc. Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.	
		0	**Fast-Strings Enable.** When set, the fast-strings feature (for REP MOVS and REP STORS) is enabled (default); when clear, fast-strings are disabled.	0F_0H
		2:1	Reserved.	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		3	**Automatic Thermal Control Circuit Enable. (R/W)** 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation.. 0 = Disabled (default).	0F_0H
		6:4	Reserved	
		7	**Performance Monitoring Available. (R)** 1 = Performance monitoring enabled 0 = Performance monitoring disabled	0F_0H
		10:8	Reserved	
		11	**Branch Trace Storage Unavailable. (RO)** 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported	0F_0H
		12	**Precise Event Based Sampling (PEBS) Unavailable. (RO)** 1 = PEBS is not supported; 0 = PEBS is supported.	06_0FH
		15:13	Reserved	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		16	**Enhanced Intel SpeedStep Technology Enable. (R/W)** 0= Enhanced Intel SpeedStep Technology disabled 1 = Enhanced Intel SpeedStep Technology enabled	06_0DH
		17	Reserved	
		18	ENABLE MONITOR FSM. (R/W) When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[bit 3] = 1).	0F_03H
			If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	
		21:19		

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		22	**Limit CPUID Maxval. (R/W)** When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 3. BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 3. Before setting this bit, BIOS must execute the CPUID.0H and examine the maximum value returned in EAX[7:0]. If the maximum value is greater than 3, the bit is supported. Otherwise, the bit is not supported. Writing to this bit when the maximum value is greater than 3 may generate a #GP exception. Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 3.	0F_03H
		23	**xTPR Message Disable. (R/W)** When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.	if CPUID.01H:ECX[14] = 1
		33:24	Reserved	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		34	**XD Bit Disable. (R/W)** When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H: EDX[20]=0).	if CPUID.80000001H:EDX[20] = 1
			When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages. BIOS must not alter the contents of this bit location, if XD bit is not supported.. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.	
		63:35	Reserved	
1D9H	473	IA32_DEBUGCTL (MSR_DEBUGCTLA, MSR_DEBUGCTLB)	Trace/Profile Resource Control (R/W)	06_0EH
		0	LBR: Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	06_01H
		1	BTF: Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	06_01H
		5:2	Reserved	
		6	TR: Setting this bit to 1 enables branch trace messages to be sent.	06_0EH

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		7	BTS: Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	06_0EH
		8	BTINT: When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	06_0EH
		9	1: BTS_OFF_OS: When set, BTS or BTM is skipped if CPL = 0.	06_0FH
		10	BTS_OFF_USR: When set, BTS or BTM is skipped if CPL > 0.	06_0FH
		11	FREEZE_LBRS_ON_PMI: When set, the LBR stack is frozen on a PMI request.	If CPUID.01H: ECX[15] = 1
		12	FREEZE_PERFMON_ON_PMI: When set, each ENABLE bit of the global counter control MSR are frozen (address 3BFH) on a PMI request	If CPUID.01H: ECX[15] = 1
		13	Reserved	
		14	FREEZE_WHILE_SMM: When set, freezes perfmon and trace messages while in SMM	06_17H
		63:15	Reserved	
1F8H	473	IA32_PLATFORM_DCA_CAP	DCA Capability (R)	06_0FH
200H	512	IA32_MTRR_PHYSBASE0 (MTRRphysBase0)	See Section 10.11.2.3, “Variable Range MTRRs.”	06_01H
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	06_01H
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	06_01H

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	06_01H
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	06_01H
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	06_01H
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	06_01H
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	06_01H
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	06_01H
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	06_01H
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	06_01H
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	06_01H
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	06_01H
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	06_01H
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	06_01H
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	06_01H
250H	592	IA32_MTRR_FIX64K_00000	MTRRfix64K_00000	06_01H
258H	600	IA32_MTRR_FIX16K_80000	MTRRfix16K_80000	06_01H
259H	601	IA32_MTRR_FIX16K_A0000	MTRRfix16K_A0000	06_01H
268H	616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)	See Section 10.11.2.2, “Fixed Range MTRRs.”	06_01H
269H	617	IA32_MTRR_FIX4K_C8000	MTRRfix4K_C8000	06_01H
26AH	618	IA32_MTRR_FIX4K_D0000	MTRRfix4K_D0000	06_01H
26BH	619	IA32_MTRR_FIX4K_D8000	MTRRfix4K_D8000	06_01H
26CH	620	IA32_MTRR_FIX4K_E0000	MTRRfix4K_E0000	06_01H
26DH	621	IA32_MTRR_FIX4K_E8000	MTRRfix4K_E8000	06_01H

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
26EH	622	IA32_MTRR_FIX4K_F0000	MTRRfix4K_F0000	06_01H
26FH	623	IA32_MTRR_FIX4K_F8000	MTRRfix4K_F8000	06_01H
277H	631	IA32_CR_PAT	IA32_CR_PAT (R/W)	06_05H
		2:0	PA0	
		7:3	Reserved	
		10:8	PA1	
		15:11	Reserved	
		18:16	PA2	
		23:19	Reserved	
		26:24	PA3	
		31:27	Reserved	
		34:32	PA4	
		39:35	Reserved	
		42:40	PA5	
		47:43	Reserved	
		50:48	PA6	
		55:51	Reserved	
		58:56	PA7	
		63:59	Reserved	
2FFH	767	IA32_MTRR_DEF_TYPE	MTRRdefType (R/W)	06_01H
		2:0	Default Memory Type	
		9:3	Reserved	
		10	Fixed Range MTRR Enable	
		11	MTRR Enable	
		63:12	Reserved	
309H	777	IA32_FIXED_CTR0 (MSR_PERF_FIXED_CTR0)	Fixed-Function Performance Counter 0 (R/W): Counts Instr_Retired.Any	If CPUID.0AH: EDX[4:0] > 0

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
30AH	778	IA32_FIXED_CTR1 (MSR_PERF_FIXED_CTR1)	Fixed-Function Performance Counter 1 0 (R/W): Counts CPU_CLK_Unhalted.Core	If CPUID.0AH: EDX[4:0] > 1
30BH	779	IA32_FIXED_CTR2 (MSR_PERF_FIXED_CTR2)	Fixed-Function Performance Counter 0 0 (R/W): Counts CPU_CLK_Unhalted.Ref	If CPUID.0AH: EDX[4:0] > 2
345H	837	IA32_PERF_CAPABILITIES	RO	If CPUID.01H: ECX[15] = 1
		5:0	LBR format	
		6	PEBS Trap	If CPUID.0AH: EDX[4:0] > 1
		7	PEBSSaveArchRegs	If CPUID.0AH: EDX[4:0] > 0
		11:8	PEBS Record Format	If CPUID.0AH: EDX[4:0] > 1
		12	1: Freeze while SMM is supported	06_17H
		63:13	Reserved	
38DH	909	IA32_FIXED_CTR_CTL (MSR_PERF_FIXED_CTR_CTL)	Fixed-Function Performance Counter Control (R/W) Counter increments while the results of ANDing respective enable bit in IA32_PERF_GLOBAL_CTRL with the corresponding OS or USR bits in this MSR is true.	If CPUID.0AH: EAX[7:0] > 1
		0	EN0_OS: Enable Fixed Counter 0 to count while CPL = 0	
		1	EN0_Usr: Enable Fixed Counter 0 to count while CPL > 0	
		2	Reserved	
		3	EN0_PMI: Enable PMI when fixed counter 0 overflows	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		4	EN1_OS: Enable Fixed Counter 1to count while CPL = 0	
		5	EN1_Usr: Enable Fixed Counter 1to count while CPL > 0	
		6	Reserved	
		7	EN1_PMI: Enable PMI when fixed counter 1 overflows	
		8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0	
		9	EN2_Usr: Enable Fixed Counter 2 to count while CPL > 0	
		10	Reserved	
		11	EN2_PMI: Enable PMI when fixed counter 2 overflows	
		63:12	Reserved	
38EH	910	IA32_PERF_GLOBAL_STATUS (MSR_PERF_GLOBAL_STATUS)	Global Performance Counter Status (RO)	If CPUID.0AH: EAX[7:0] > 0
		0	Ovf_PMC0: Overflow status of IA32_PMC0	If CPUID.0AH: EAX[7:0] > 0
		1	Ovf_PMC1: Overflow status of IA32_PMC1	If CPUID.0AH: EAX[7:0] > 0
		31:2	Reserved	
		32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0	If CPUID.0AH: EAX[7:0] > 1
		33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1	If CPUID.0AH: EAX[7:0] > 1
		34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2	If CPUID.0AH: EAX[7:0] > 1
		61:35	Reserved	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		62	OvfBuf: DS SAVE area Buffer overflow status	If CPUID.0AH: EAX[7:0] > 0
		63	CondChg: status bits of this regisetr has changed	If CPUID.0AH: EAX[7:0] > 0
3BFH	911	IA32_PERF_GLOBAL_CTRL (MSR_PERF_GLOBAL_CTRL)	Global Performance Counter Control (R/W) Counter increments while the result of ANDing respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.	If CPUID.0AH: EAX[7:0] > 0
		0	EN_PMC0	If CPUID.0AH: EAX[7:0] > 0
		1	EN_PMC1	If CPUID.0AH: EAX[7:0] > 0
		31:2	Reserved	
		32	EN_FIXED_CTR0	If CPUID.0AH: EAX[7:0] > 1
		33	EN_FIXED_CTR1	If CPUID.0AH: EAX[7:0] > 1
		34	EN_FIXED_CTR2	If CPUID.0AH: EAX[7:0] > 1
		63:35	Reserved	
390H	912	IA32_PERF_GLOBAL_OVF_CTRL (MSR_PERF_GLOBAL_OVF_CTRL)	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0
		0	Set 1 to Clear Ovf_PMC0 bit	If CPUID.0AH: EAX[7:0] > 0
		1	Set 1 to Clear Ovf_PMC1 bit	If CPUID.0AH: EAX[7:0] > 0
		31:2	Reserved	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit	If CPUID.0AH: EAX[7:0] > 1

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit	If CPUID.0AH: EAX[7:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit	If CPUID.0AH: EAX[7:0] > 1
		61:35	Reserved	
		62	Set 1 to Clear OvfBuf: bit	If CPUID.0AH: EAX[7:0] > 0
		63	Set 1 to CondChg: bit	If CPUID.0AH: EAX[7:0] > 0
3F1H	1009	IA32_PEBS_ENABLE	PEBS Control (R/W)	06_0FH
		0	Enable PEBS on IA32_PMC0	
		63:1	Reserved	
400H	1024	IA32_MC0_CTL	MC0_CTL	P6 Family Processors
401H	1025	IA32_MC0_STATUS	MC0_STATUS	P6 Family Processors
402H	1026	IA32_MC0_ADDR[1]	MC0_ADDR	P6 Family Processors
403H	1027	IA32_MC0_MISC	MC0_MISC	P6 Family Processors
404H	1028	IA32_MC1_CTL	MC1_CTL	P6 Family Processors
405H	1029	IA32_MC1_STATUS	MC1_STATUS	P6 Family Processors
406H	1030	IA32_MC1_ADDR[3]	MC1_ADDR	P6 Family Processors
407H	1031	IA32_MC1_MISC	MC1_MISC	P6 Family Processors
408H	1032	IA32_MC2_CTL	MC2_CTL	P6 Family Processors
409H	1033	IA32_MC2_STATUS	MC2_STATUS	P6 Family Processors
40AH	1034	IA32_MC2_ADDR[1]	MC2_ADDR	P6 Family Processors

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
40BH	1035	IA32_MC2_MISC	MC2_MISC	P6 Family Processors
40CH	1036	IA32_MC3_CTL	MC4_CTL	P6 Family Processors
40DH	1037	IA32_MC3_STATUS	MC4_STATUS	P6 Family Processors
40EH	1038	IA32_MC3_ADDR[1]	MC4_ADDR	P6 Family Processors
40FH	1039	IA32_MC3_MISC	MC4_MISC	P6 Family Processors
410H	1040	IA32_MC4_CTL	MC3_CTL	P6 Family Processors
411H	1041	IA32_MC4_STATUS	MC3_STATUS	P6 Family Processors
412H	1038	IA32_MC4_ADDR[1]	MC3_ADDR	P6 Family Processors
413H	1039	IA32_MC4_MISC	MC3_MISC	P6 Family Processors
480H	1152	IA32_VMX_BASIC	**Reporting Register of Basic VMX Capabilities. (R/O)** See Appendix G.1, “Basic VMX Information”	If CPUID.01H:ECX.[bit 5] = 1
481H	1153	IA32_VMX_PINBASED_CTLS	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** See Appendix G.3.1, “Pin-Based VM-Execution Controls”	If CPUID.01H:ECX.[bit 5] = 1

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
482H	1154	IA32_VMX_PROCBASED_CTLS	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix G.3.2, "Primary Processor-Based VM-Execution Controls"	If CPUID.01H:ECX.[bit 5] = 1
483H	1155	IA32_VMX_EXIT_CTLS	**Capability Reporting Register of VM-exit Controls. (R/O)** See Appendix G.4, "VM-Exit Controls"	If CPUID.01H:ECX.[bit 5] = 1
484H	1156	IA32_VMX_ENTRY_CTLS	**Capability Reporting Register of VM-entry Controls. (R/O)** See Appendix G.5, "VM-Entry Controls"	If CPUID.01H:ECX.[bit 5] = 1
485H	1157	IA32_VMX_MISC_CTLS	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Appendix G.6, "Miscellaneous Data"	If CPUID.01H:ECX.[bit 5] = 1
486H	1158	IA32_VMX_CR0_FIXED0	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Appendix G.7, "VMX-Fixed Bits in CR0"	If CPUID.01H:ECX.[bit 5] = 1
487H	1159	IA32_VMX_CR0_FIXED1	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Appendix G.7, "VMX-Fixed Bits in CR0"	If CPUID.01H:ECX.[bit 5] = 1
488H	1160	IA32_VMX_CR4_FIXED0	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Appendix G.8, "VMX-Fixed Bits in CR4"	If CPUID.01H:ECX.[bit 5] = 1

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
489H	1161	IA32_VMX_CR4_FIXED1	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Appendix G.8, “VMX-Fixed Bits in CR4”	If CPUID.01H:ECX.[bit 5] = 1
48AH	1162	IA32_VMX_VMCS_ENUM	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Appendix G.9, “VMCS Enumeration”	If CPUID.01H:ECX.[bit 5] = 1
48BH	1163	IA32_VMX_PROCBASED_CTLS2	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix G.3.3, “Secondary Processor-Based VM-Execution Controls”	If (CPUID.01H:ECX.[bit 5] and IA32_VMX_PROCBASED_CTLS[bit 63])
48CH	1164	IA32_VMX_EPT_VPID_CAP	**Capability Reporting Register of EPT and VPID. (R/O)** See Appendix G.10, “VPID and EPT Capabilities”	If (CPUID.01H:ECX.[bit 5], IA32_VMX_PROCBASED_CTLS[bit 63], and either IA32_VMX_PROCBASED_CTLS2[bit 33] or IA32_VMX_PROCBASED_CTLS2[bit 37])
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	**Capability Reporting Register of Pin-based VM-execution Flex Controls. (R/O)** See Appendix G.3.1, “Pin-Based VM-Execution Controls”	If (CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 55])

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls. (R/O)** See Appendix G.3.2, “Primary Processor-Based VM-Execution Controls”	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 55])
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	**Capability Reporting Register of VM-exit Flex Controls. (R/O)** See Appendix G.4, “VM-Exit Controls”	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 55])
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	**Capability Reporting Register of VM-entry Flex Controls. (R/O)** See Appendix G.5, “VM-Entry Controls”	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC[bit 55])
600H	1536	IA32_DS_AREA	**DS Save Area. (R/W)** Points to the linear address of the first byte of the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 18.16.4, “Debug Store (DS) Mechanism.”	0F_0H
		63:0	The linear address of the first byte of the DS buffer management area, if IA-32e mode is active.	
		31:0	The linear address of the first byte of the DS buffer management area, if not in IA-32e mode.	
		63:32	Reserved iff not in IA-32e mode.	

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
C000_ 0080H		IA32_EFER	**Extended Feature Enables.**	If (CPUID.80000001 .EDX.[bit 20] or CPUID.80000001 .EDX.[bit29])
		0	**SYSCALL Enable. (R/W)** Enables SYSCALL/SYSRET instructions in 64-bit mode.	
		7:1	Reserved.	
		8	**IA-32e Mode Enable. (R/W)** Enables IA-32e mode operation.	
		9	Reserved.	
		10	**IA-32e Mode Active. (R)** Indicates IA-32e mode is active when set.	
		11	Execute Disable Bit Enable. (R)	
		63:12	Reserved	
C000_ 0081H		IA32_STAR	**System Call Target Address. (R/W)**	If CPUID.80000001 .EDX.[bit 29] = 1
C000_ 0082H		IA32_LSTAR	**IA-32e Mode System Call Target Address. (R/W)**	If CPUID.80000001 .EDX.[bit 29] = 1
C000_ 0084H		IA32_FMASK	**System Call Flag Mask. (R/W)**	If CPUID.80000001 .EDX.[bit 29] = 1
C000_ 0100H		IA32_FS_BASE	**Map of BASE Address of FS. (R/W)**	If CPUID.80000001 .EDX.[bit 29] = 1
C000_ 0101H		IA32_GS_BASE	**Map of BASE Address of GS. (R/W)**	If CPUID.80000001 .EDX.[bit 29] = 1

Table B-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
C000_ 0102H		IA32_KERNEL_GS_BASE	**Swap Target of BASE Address of GS. (R/W)**	If CPUID.80000001 .EDX.[bit 29] = 1

NOTES:

1. In processors based on Intel NetBurst microarchitecture, MSR addresses 180H-197H are supported, software must treat them as model-specific. Starting with Intel Core Duo processors, MSR addresses 180H-185H, 188H-197H are reserved.
2. In processors based on Intel NetBurst microarchitecture, MSR addresses 180H-197H are supported, software must treat them as model-specific. Starting with Intel Core Duo processors, MSR addresses 180H-185H, 188H-197H are reserved.
3. The *_ADDR MSRs may or may not be present; this depends on flag settings in IA32_MC*i*_STATUS. See Section 14.3.2.3 and Section 14.3.2.4 for more information.

B.2 MSRS IN THE INTEL® CORE™ 2 PROCESSOR FAMILY

Table B-3 lists model-specific registers (MSRs) for Intel Core 2 processor family and for Intel Xeon processors based on Intel Core microarchitecture, architectural MSR addresses are also included in Table B-3. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_0FH, see Table B-1.

MSRs listed in Table B-2 and Table B-3 are also supported by processors based on the Enhanced Intel Core microarchitecture. Processors based on the Enhanced Intel Core microarchitecture have the CPUID signature DisplayFamily_DisplayModel of 06_17H.

The column “Shared/Unique” applies to multi-core processors based on Intel Core microarchitecture. “Unique” means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. “Shared” means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ ADDR	Unique	See Appendix B.8, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_ TYPE	Unique	See Appendix B.8, “MSRs in Pentium Processors.”

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
6H	6	IA32_MONITOR_ FILTER_SIZE	Unique	See Section 7.11.5, "Monitor/Mwait Address Range Determination." andTable B-2
10H	16	IA32_TIME_ STAMP_COUNTER	Unique	See Section 18.10, "Time-Stamp Counter." and see Table B-2
17H	23	IA32_PLATFORM_I D	Shared	**Platform ID. (R)** See Table B-2.
17H	23	MSR_PLATFORM_I D	Shared	**Model Specific Platform ID. (R)**
		7:0		Reserved.
		12:8		**Maximum Qualified Ratio. (R)** The maximum allowed bus ratio.
		49:13		Reserved.
		52:50		See Table B-2.
		63:53		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 9.4.4, "Local APIC Status and Location." and Table B-2
2AH	42	MSR_EBL_CR_ POWERON	Shared	**Processor Hard Power-On Configuration. (R/W)** Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved
		1		**Data Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		2		**Response Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		3		**MCERR# Drive Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		4		**Address Parity Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		5		Reserved
		6		Reserved
		7		**BINIT# Driver Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		8		**Output Tri-state Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		9		**Execute BIST. (R/O)** 1 = Enabled; 0 = Disabled
		10		**MCERR# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		11		Intel TXT Capable Chipset. (R/O) 1 = Present; 0 = Not Present
		12		**BINIT# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		13		**Reserved**
		14		**1 MByte Power on Reset Vector. (R/O)** 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		**APIC Cluster ID. (R/O)**
		18		**N/2 Non-Integer Bus Ratio. (R/O)** 0 = Integer ratio; 1 = Non-integer ratio
		19		Reserved.
		21: 20		**Symmetric Arbitration ID. (R/O)**
		26:22		**Integer Bus Frequency Ratio. (R/O)**
3AH	58	IA32_FEATURE_ CONTROL	Unique	**Control Features in Intel 64Processor. (R/W).** see Table B-2

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
40H	64	MSR_ LASTBRANCH_0_F ROM_IP	Unique	**Last Branch Record 0 From IP. (R/W)** One of four pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction for one of the last four branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 18.8, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_ LASTBRANCH_1_F ROM_IP	Unique	**Last Branch Record 1 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_ LASTBRANCH_2_F ROM_IP	Unique	**Last Branch Record 2 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_ LASTBRANCH_3_F ROM_IP	Unique	**Last Branch Record 3 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_ LASTBRANCH_0_ TO_LIP	Unique	**Last Branch Record 0 To IP. (R/W)** One of four pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last four branches, exceptions, or interrupts taken by the processor.
61H	97	MSR_ LASTBRANCH_1_ TO_LIP	Unique	**Last Branch Record 1 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
62H	98	MSR_ LASTBRANCH_2_ TO_LIP	Unique	**Last Branch Record 2 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
63H	99	MSR_ LASTBRANCH_3_ TO_LIP	Unique	**Last Branch Record 3 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
79H	121	IA32_BIOS_ UPDT_TRIG	Unique	**BIOS Update Trigger Register. (R/W)** see Table B-2
8BH	139	IA32_BIOS_ SIGN_ID	Unique	**BIOS Update Signature ID. (RO)** see Table B-2
C1H	193	IA32_PMC0	Unique	**Performance counter register.** see Table B-2
C2H	194	IA32_PMC1	Unique	**Performance counter register.** see Table B-2
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO).** This field indicates the intended scaleable bus clock speed for processors based on Intel Core microarchitecture:
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333)
				133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.
		63:3		Reserved
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO).** This field indicates the intended scaleable bus clock speed for processors based on Enhanced Intel Core microarchitecture:

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) ▪ 110B: 400 MHz (FSB 1600) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 110B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.
		63:3		Reserved
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count. (RW)** see Table B-2
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count. (RW)** see Table B-2
FEH	254	IA32_MTRRCAP	Unique	see Table B-2
11EH	281	MSR_BBL_CR_ CTL3	Shared	
		0		**L2 Hardware Enabled. (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		8		**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.
		23		**L2 Not Present. (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	see Table B-2
175H	373	IA32_SYSENTER_ESP	Unique	see Table B-2
176H	374	IA32_SYSENTER_EIP	Unique	see Table B-2
179H	377	IA32_MCG_CAP	Unique	see Table B-2
17AH	378	IA32_MCG_STATUS	Unique	
		0		**RIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted
		1		**EIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2		**MCIP.** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_ PERFEVTSEL0	Unique	see Table B-2
187H	391	IA32_ PERFEVTSEL1	Unique	see Table B-2
198H	408	IA32_PERF_STAT US	Shared	see Table B-2
198H	408	MSR_PERF_STATU S	Shared	
		15:0		Current Performance State Value.
		30:16		Reserved.
		31		XE Operation (R/O). If set, XE operation is enabled. Default is cleared.
		39:32		Reserved.
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		45		Reserved
		46		Non-Integer Bus Ratio (R/O) Indicates non-integer bus ratio is enabled. Applies processors based on Enhanced Intel Core microarchitecture.
		63:47		Reserved.
199H	409	IA32_PERF_CTL	Unique	see Table B-2

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
19AH	410	IA32_CLOCK_ MODULATION	Unique	**Clock Modulation. (R/W)** see Table B-2 IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_ INTERRUPT	Unique	**Thermal Interrupt Control. (R/W)** see Table B-2
19CH	412	IA32_THERM_ STATUS	Unique	**Thermal Monitor Status. (R/W)** see Table B-2
19DH	413	MSR_THERM2_ CTL	Unique	
		15:0		Reserved.
		16		**TM_SELECT. (R/W)** Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:16		Reserved.
1A0	416	IA32_MISC_ ENABLE		**Enable Misc. Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0		**Fast-Strings Enable.** see Table B-2
		2:1		Reserved.
		3	Unique	**Automatic Thermal Control Circuit Enable. (R/W)** see Table B-2
		6:4		Reserved.
		7	Shared	**Performance Monitoring Available. (R)** see Table B-2
		8		Reserved.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		9		**Hardware Prefetcher Disable. (R/W)** When set, disables the hardware prefetcher operation on streams of data. When clear (default), enables the prefetch queue. Disabling of the hardware prefetcher may impact processor performance.
		10	Shared	**FERR# Multiplexing Enable. (R/W)** 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	**Branch Trace Storage Unavailable. (RO)** see Table B-2
		12	Shared	**Precise Event Based Sampling Unavailable. (RO)** see Table B-2
		13	Shared	**TM2 Enable. (R/W)** When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		15:14		Reserved.
		16	Shared	**Enhanced Intel SpeedStep Technology Enable. (R/W)** see Table B-2
		18	Shared	ENABLE MONITOR FSM. (R/W) see Table B-2
		19	Shared	**Adjacent Cache Line Prefetch Disable. (R/W)** When set to 1, the processor fetches the cache line that contains data currently required by the processor. When set to 0, the processor fetches cache lines that comprise a cache line pair (128 bytes). Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.
		20	Shared	**Enhanced Intel SpeedStep Technology Select Lock. (R/WO)** When set, this bit causes the following bits to become read-only: ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit), ▪ Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved.
		22	Shared	**Limit CPUID Maxval. (R/W)** see Table B-2
		23	Shared	**xTPR Message Disable. (R/W)** see Table B-2
		33:24		Reserved.
		34	Unique	**XD Bit Disable. (R/W)** see Table B-2
		36:35		Reserved.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		37	Unique	**DCU Prefetcher Disable. (R/W)** When set to 1, The DCU L1 data cache prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The DCU prefetcher is an L1 data cache prefetcher. When the DCU prefetcher detects multiple loads from the same line done within a time limit, the DCU prefetcher assumes the next line will be required. The next line is prefetched in to the L1 data cache from memory or L2.
		38	Shared	**IDA Disable. (R/W)** When set to 1 on processors that support IDA, the Intel Dynamic Acceleration feature (IDA) is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of IDA is enabled. **Note:** the power-on default value is used by BIOS to detect hardware support of IDA. If power-on default value is 1, IDA is available in the processor. If power-on default value is 0, IDA is not available.
		39	Unique	**IP Prefetcher Disable. (R/W)** When set to 1, The IP prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The IP prefetcher is an L1 data cache prefetcher. The IP prefetcher looks for sequential load history to determine whether to prefetch the next expected data into the L1 cache from memory or L2.
		63:40		Reserved.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
1C9H	457	MSR_ LASTBRANCH_ TOS	Unique	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0 (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	**Debug Control. (R/W)** see Table B-2
1DDH	477	MSR_LER_FROM_ LIP	Unique	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_ LIP	Unique	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYS BASE0	Unique	see Table B-2
201H	513	IA32_MTRR_PHYS MASK0	Unique	see Table B-2
202H	514	IA32_MTRR_PHYS BASE1	Unique	see Table B-2
203H	515	IA32_MTRR_PHYS MASK1	Unique	see Table B-2
204H	516	IA32_MTRR_PHYS BASE2	Unique	see Table B-2
205H	517	IA32_MTRR_PHYS MASK2	Unique	see Table B-2
206H	518	IA32_MTRR_PHYS BASE3	Unique	see Table B-2
207H	519	IA32_MTRR_PHYS MASK3	Unique	see Table B-2
208H	520	IA32_MTRR_PHYS BASE4	Unique	see Table B-2

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
209H	521	IA32_MTRR_PHYSMASK4	Unique	see Table B-2
20AH	522	IA32_MTRR_PHYSBASE5	Unique	see Table B-2
20BH	523	IA32_MTRR_PHYSMASK5	Unique	see Table B-2
20CH	524	IA32_MTRR_PHYSBASE6	Unique	see Table B-2
20DH	525	IA32_MTRR_PHYSMASK6	Unique	see Table B-2
20EH	526	IA32_MTRR_PHYSBASE7	Unique	see Table B-2
20FH	527	IA32_MTRR_PHYSMASK7	Unique	see Table B-2
250H	592	IA32_MTRR_FIX64K_00000	Unique	see Table B-2
258H	600	IA32_MTRR_FIX16K_80000	Unique	see Table B-2
259H	601	IA32_MTRR_FIX16K_A0000	Unique	see Table B-2
268H	616	IA32_MTRR_FIX4K_C0000	Unique	see Table B-2
269H	617	IA32_MTRR_FIX4K_C8000	Unique	see Table B-2
26AH	618	IA32_MTRR_FIX4K_D0000	Unique	see Table B-2
26BH	619	IA32_MTRR_FIX4K_D8000	Unique	see Table B-2
26CH	620	IA32_MTRR_FIX4K_E0000	Unique	see Table B-2
26DH	621	IA32_MTRR_FIX4K_E8000	Unique	see Table B-2
26EH	622	IA32_MTRR_FIX4K_F0000	Unique	see Table B-2

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
26FH	623	IA32_MTRR_FIX4K_F8000	Unique	see Table B-2
277H	631	IA32_CR_PAT	Unique	see Table B-2
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	**Default Memory Types. (R/W)** see Table B-2
309H	777	IA32_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0. (R/W)** see Table B-2
309H	777	MSR_PERF_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0. (R/W)**
30AH	778	IA32_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1. (R/W)** see Table B-2
30AH	778	MSR_PERF_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1. (R/W)**
30BH	779	IA32_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2. (R/W)** see Table B-2
30BH	779	MSR_PERF_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2. (R/W)**
345H	837	IA32_PERF_CAPABILITIES	Unique	see Table B-2. See Section 18.5.1, “IA32_DEBUGCTL MSR.”
345H	837	MSR_PERF_CAPABILITIES	Unique	RO. This applies to processors that do not support architectural perfmon version 2.
		5:0		LBR Format. see Table B-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. see Table B-2.
		63:8		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Unique	**Fixed-Function-Counter Control Register. (R/W)** see Table B-2
38DH	909	MSR_PERF_FIXED_CTR_CTRL	Unique	**Fixed-Function-Counter Control Register. (R/W)**
38EH	910	IA32_PERF_GLOBAL_STAUS	Unique	see Table B-2. See Section 18.14.2, “Global Counter Control Facilities.”
38EH	910	MSR_PERF_GLOBAL_STAUS	Unique	See Section 18.14.2, “Global Counter Control Facilities.”

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
38FH	911	IA32_PERF_ GLOBAL_CTRL	Unique	see Table B-2. See Section 18.14.2, "Global Counter Control Facilities."
38FH	911	MSR_PERF_ GLOBAL_CTRL	Unique	See Section 18.14.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_ GLOBAL_OVF_ CTRL	Unique	see Table B-2. See Section 18.14.2, "Global Counter Control Facilities."
390H	912	MSR_PERF_ GLOBAL_OVF_ CTRL	Unique	See Section 18.14.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ ENABLE	Unique	see Table B-2. See Section 18.14.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MC0_CTL	Unique	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_ STATUS	Unique	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Unique	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_ STATUS	Unique	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Unique	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
408H	1032	IA32_MC2_CTL	Unique	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_ STATUS	Unique	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	Unique	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_ STATUS	Unique	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC4_ADDR	Unique	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC3_CTL		See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC3_ STATUS		See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	MSR_MC3_ADDR	Unique	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	MSR_MC3_MISC	Unique	
414H	1044	MSR_MC5_CTL	Unique	

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
415H	1045	MSR_MC5_ STATUS	Unique	
416H	1046	MSR_MC5_ADDR	Unique	
417H	1047	MSR_MC5_MISC	Unique	
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities. (R/O)** see Table B-2. See Appendix G.1, “Basic VMX Information”
481H	1153	IA32_VMX_PINBA SED_CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** see Table B-2. See Appendix G.3, “VM-Execution Controls”
482H	1154	IA32_VMX_PROCB ASED_CTLS	Unique	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix G.3, “VM-Execution Controls”
483H	1155	IA32_VMX_EXIT_ CTLS	Unique	**Capability Reporting Register of VM-exit Controls. (R/O)** see Table B-2. See Appendix G.4, “VM-Exit Controls”
484H	1156	IA32_VMX_ ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls. (R/O)** see Table B-2. See Appendix G.5, “VM-Entry Controls”
485H	1157	IA32_VMX_MISC	Unique	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** see Table B-2. See Appendix G.6, “Miscellaneous Data”
486H	1158	IA32_VMX_CR0_ FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** see Table B-2. See Appendix G.7, “VMX-Fixed Bits in CR0”
487H	1159	IA32_VMX_CR0_ FIXED1	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** see Table B-2. See Appendix G.7, “VMX-Fixed Bits in CR0”
488H	1160	IA32_VMX_CR4_FI XED0	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** see Table B-2. See Appendix G.8, “VMX-Fixed Bits in CR4”

Table B-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
489H	1161	IA32_VMX_CR4_FIXED1	Unique	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** see Table B-2. See Appendix G.8, "VMX-Fixed Bits in CR4"
48AH	1162	IA32_VMX_VMCS_ENUM	Unique	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** see Table B-2. See Appendix G.9, "VMCS Enumeration"
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Unique	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix G.3, "VM-Execution Controls"
600H	1536	IA32_DS_AREA	Unique	**DS Save Area. (R/W).** see Table B-2 See Section 18.16.4, "Debug Store (DS) Mechanism."
C000_0080H		IA32_EFER	Unique	**Extended Feature Enables.** see Table B-2
C000_0081H		IA32_STAR	Unique	**System Call Target Address. (R/W).** see Table B-2
C000_0082H		IA32_LSTAR	Unique	**IA-32e Mode System Call Target Address. (R/W).** see Table B-2
C000_0084H		IA32_FMASK	Unique	**System Call Flag Mask. (R/W).** see Table B-2
C000_0100H		IA32_FS_BASE	Unique	**Map of BASE Address of FS. (R/W).** see Table B-2
C000_0101H		IA32_GS_BASE	Unique	**Map of BASE Address of GS. (R/W).** see Table B-2
C000_0102H		IA32_KERNEL_GSBASE	Unique	**Swap Target of BASE Address of GS. (R/W).** see Table B-2

B.3 MSRS IN THE INTEL® ATOM™ PROCESSOR FAMILY

Table B-4 lists model-specific registers (MSRs) for Intel Atom processor family, architectural MSR addresses are also included in Table B-4. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_0CH, see Table B-1.

The column “Shared/Unique” applies to logical processors sharing the same core in processors based on the Intel Atom microarchitecture. “Unique” means each logical processor has a separate MSR, or a bit field in an MSR governs only a logical processor. “Shared” means the MSR or the bit field in an MSR address governs the operation of both logical processors in the same core.

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Shared	See Appendix B.8, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	Shared	See Appendix B.8, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 7.11.5, “Monitor/Mwait Address Range Determination.” andTable B-2
10H	16	IA32_TIME_STAMP_COUNTER	Shared	See Section 18.10, “Time-Stamp Counter.” and see Table B-2
17H	23	IA32_PLATFORM_ID	Shared	**Platform ID. (R)** See Table B-2.
17H	23	MSR_PLATFORM_ID	Shared	**Model Specific Platform ID. (R)**
		7:0		Reserved.
		12:8		**Maximum Qualified Ratio. (R)** The maximum allowed bus ratio.
		63:13		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 9.4.4, “Local APIC Status and Location.” and Table B-2
2AH	42	MSR_EBL_CR_POWERON	Shared	**Processor Hard Power-On Configuration. (R/W)** Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		1		**Data Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		2		**Response Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		3		**AERR# Drive Enable. (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		4		**BERR# Enable for initiator bus requests. (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		5		Reserved
		6		Reserved
		7		**BINIT# Driver Enable. (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		8		Reserved
		9		**Execute BIST. (R/O)** 1 = Enabled; 0 = Disabled
		10		**AERR# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		11		Reserved
		12		**BINIT# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		13		**Reserved**
		14		**1 MByte Power on Reset Vector. (R/O)** 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		17:16		**APIC Cluster ID. (R/O)** Always 00B.
		19: 18		Reserved.
		21: 20		**Symmetric Arbitration ID. (R/O)** Always 00B.
		26:22		**Integer Bus Frequency Ratio. (R/O)**
3AH	58	IA32_FEATURE_ CONTROL	Unique	**Control Features in Intel 64Processor. (R/W).** see Table B-2
40H	64	MSR_ LASTBRANCH_0_F ROM_IP	Unique	**Last Branch Record 0 From IP. (R/W)** One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction for one of the last eight branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 18.8, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors).”
41H	65	MSR_ LASTBRANCH_1_F ROM_IP	Unique	**Last Branch Record 1 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_ LASTBRANCH_2_F ROM_IP	Unique	**Last Branch Record 2 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_ LASTBRANCH_3_F ROM_IP	Unique	**Last Branch Record 3 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_ LASTBRANCH_4_F ROM_IP	Unique	**Last Branch Record 4 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_ LASTBRANCH_5_F ROM_IP	Unique	**Last Branch Record 5 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
46H	70	MSR_ LASTBRANCH_6_FROM_IP	Unique	**Last Branch Record 6 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
47H	71	MSR_ LASTBRANCH_7_FROM_IP	Unique	**Last Branch Record 7 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_ LASTBRANCH_0_ TO_LIP	Unique	**Last Branch Record 0 To IP. (R/W)** One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last eight branches, exceptions, or interrupts taken by the processor.
61H	97	MSR_ LASTBRANCH_1_ TO_LIP	Unique	**Last Branch Record 1 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
62H	98	MSR_ LASTBRANCH_2_ TO_LIP	Unique	**Last Branch Record 2 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
63H	99	MSR_ LASTBRANCH_3_ TO_LIP	Unique	**Last Branch Record 3 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
64H	100	MSR_ LASTBRANCH_4_ TO_LIP	Unique	**Last Branch Record 4 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
65H	101	MSR_ LASTBRANCH_5_ TO_LIP	Unique	**Last Branch Record 5 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
66H	102	MSR_ LASTBRANCH_6_ TO_LIP	Unique	**Last Branch Record 6 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
67H	103	MSR_ LASTBRANCH_7_ TO_LIP	Unique	**Last Branch Record 7 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
79H	121	IA32_BIOS_ UPDT_TRIG	Unique	**BIOS Update Trigger Register. (R/W)** see Table B-2
8BH	139	IA32_BIOS_ SIGN_ID	Unique	**BIOS Update Signature ID. (RO)** see Table B-2
C1H	193	IA32_PMC0	Unique	**Performance counter register.** see Table B-2
C2H	194	IA32_PMC1	Unique	**Performance counter register.** see Table B-2
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO).** This field indicates the intended scaleable bus clock speed for processors based on Intel Atom microarchitecture:
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
		63:3		Reserved
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count. (RW)** see Table B-2
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count. (RW)** see Table B-2
11EH	281	MSR_BBL_CR_ CTL3	Shared	
		0		**L2 Hardware Enabled. (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		8		**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.
		23		**L2 Not Present. (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	see Table B-2
175H	373	IA32_SYSENTER_ESP	Unique	see Table B-2
176H	374	IA32_SYSENTER_EIP	Unique	see Table B-2
17AH	378	IA32_MCG_STATUS	Unique	
		0		**RIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted
		1		**EIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2		**MCIP.** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_PERFEVTSEL0	Unique	see Table B-2
187H	391	IA32_PERFEVTSEL1	Unique	see Table B-2
198H	408	IA32_PERF_STATUS	Shared	see Table B-2
198H	408	MSR_PERF_STATUS	Shared	
		15:0		Current Performance State Value.
		39:16		Reserved.
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		63:45		Reserved.
199H	409	IA32_PERF_CTL	Unique	see Table B-2
19AH	410	IA32_CLOCK_MODULATION	Unique	**Clock Modulation. (R/W)** see Table B-2 IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	**Thermal Interrupt Control. (R/W)** see Table B-2
19CH	412	IA32_THERM_STATUS	Unique	**Thermal Monitor Status. (R/W)** see Table B-2
19DH	413	MSR_THERM2_CTL	Shared	

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		15:0		Reserved.
		16		**TM_SELECT. (R/W)** Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:17		Reserved.
1A0	416	IA32_MISC_ ENABLE	Unique	**Enable Misc. Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0		**Fast-Strings Enable.** see Table B-2
		2:1		Reserved.
		3	Unique	**Automatic Thermal Control Circuit Enable. (R/W)** see Table B-2
		6:4		Reserved.
		7	Shared	**Performance Monitoring Available. (R)** see Table B-2
		8		Reserved.
		9		Reserved.
		10	Shared	**FERR# Multiplexing Enable. (R/W)** 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	**Branch Trace Storage Unavailable. (RO)** see Table B-2

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		12	Shared	**Precise Event Based Sampling Unavailable. (RO)** see Table B-2
		13	Shared	**TM2 Enable. (R/W)** When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.
		15:14		Reserved.
		16	Shared	**Enhanced Intel SpeedStep Technology Enable. (R/W)** see Table B-2
		18	Shared	ENABLE MONITOR FSM. (R/W) see Table B-2
		19		Reserved.
		20	Shared	**Enhanced Intel SpeedStep Technology Select Lock. (R/WO)** When set, this bit causes the following bits to become read-only: ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit), ▪ Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		21		Reserved.
		22	Unique	**Limit CPUID Maxval. (R/W)** see Table B-2
		23	Shared	**xTPR Message Disable. (R/W)** see Table B-2
		33:24		Reserved.
		34	Unique	**XD Bit Disable. (R/W)** see Table B-2
		63:35		Reserved.
1C9H	457	MSR_ LASTBRANCH_ TOS	Unique	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0 (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	**Debug Control. (R/W)** see Table B-2
1DDH	477	MSR_LER_FROM_ LIP	Unique	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_ LIP	Unique	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
277H	631	IA32_CR_PAT	Unique	see Table B-2
309H	777	IA32_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0. (R/W)** see Table B-2
30AH	778	IA32_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1. (R/W)** see Table B-2
30BH	779	IA32_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2. (R/W)** see Table B-2
345H	837	IA32_PERF_CAPA BILITIES	Shared	see Table B-2. See Section 18.5.1, “IA32_DEBUGCTL MSR.”
38DH	909	IA32_FIXED_CTR_ CTRL	Unique	**Fixed-Function-Counter Control Register. (R/W)** see Table B-2

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
38EH	910	IA32_PERF_ GLOBAL_STAUS	Unique	see Table B-2. See Section 18.14.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_ GLOBAL_CTRL	Unique	see Table B-2. See Section 18.14.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_ GLOBAL_OVF_ CTRL	Unique	see Table B-2. See Section 18.14.2, "Global Counter Control Facilities."
3F1H	1009	IA32_PEBS_ ENABLE	Unique	see Table B-2. See Section 18.14.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MC0_CTL	Shared	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_ STATUS	Shared	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Shared	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Shared	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_ STATUS	Shared	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Shared	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_ STATUS	Shared	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Shared	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
410H	1040	MSR_MC3_CTL	Shared	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
411H	1041	MSR_MC3_ STATUS	Shared	See Section 14.3.2.2, “IA32_MCi_STATUS MSRS.”
412H	1042	MSR_MC3_ADDR	Shared	See Section 14.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities. (R/O)** see Table B-2. See Appendix G.1, “Basic VMX Information”
481H	1153	IA32_VMX_PINBA SED_CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** see Table B-2. See Appendix G.3, “VM-Execution Controls”
482H	1154	IA32_VMX_PROCB ASED_CTLS	Unique	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix G.3, “VM-Execution Controls”
483H	1155	IA32_VMX_EXIT_ CTLS	Unique	**Capability Reporting Register of VM-exit Controls. (R/O)** see Table B-2. See Appendix G.4, “VM-Exit Controls”
484H	1156	IA32_VMX_ ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls. (R/O)** see Table B-2. See Appendix G.5, “VM-Entry Controls”
485H	1157	IA32_VMX_MISC	Unique	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** see Table B-2. See Appendix G.6, “Miscellaneous Data”
486H	1158	IA32_VMX_CR0_ FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** see Table B-2. See Appendix G.7, “VMX-Fixed Bits in CR0”

Table B-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
487H	1159	IA32_VMX_CR0_ FIXED1	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** see Table B-2. See Appendix G.7, "VMX-Fixed Bits in CR0"
488H	1160	IA32_VMX_CR4_FI XED0	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** see Table B-2. See Appendix G.8, "VMX-Fixed Bits in CR4"
489H	1161	IA32_VMX_CR4_FI XED1	Unique	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** see Table B-2. See Appendix G.8, "VMX-Fixed Bits in CR4"
48AH	1162	IA32_VMX_ VMCS_ENUM	Unique	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** see Table B-2. See Appendix G.9, "VMCS Enumeration"
48BH	1163	IA32_VMX_PROCB ASED_CTLS2	Unique	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix G.3, "VM-Execution Controls"
600H	1536	IA32_DS_AREA	Unique	**DS Save Area. (R/W).** see Table B-2 See Section 18.16.4, "Debug Store (DS) Mechanism."
C000_ 0080H		IA32_EFER	Unique	**Extended Feature Enables.** see Table B-2
C000_ 0081H		IA32_STAR	Unique	**System Call Target Address. (R/W).** see Table B-2
C000_ 0082H		IA32_LSTAR	Unique	**IA-32e Mode System Call Target Address. (R/W).** see Table B-2
C000_ 0084H		IA32_FMASK	Unique	**System Call Flag Mask. (R/W).** see Table B-2
C000_ 0100H		IA32_FS_BASE	Unique	**Map of BASE Address of FS. (R/W).** see Table B-2
C000_ 0101H		IA32_GS_BASE	Unique	**Map of BASE Address of GS. (R/W).** see Table B-2
C000_ 0102H		IA32_KERNEL_GS BASE	Unique	**Swap Target of BASE Address of GS. (R/W).** see Table B-2

B.4 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table B-5 lists MSRs (architectural and model-specific) that are defined across processor generations based on Intel NetBurst microarchitecture. The processor can be identified by its CPUID signatures of DisplayFamily encoding of 0FH, see Table B-1.

- MSRs with an “IA32_” prefix are designated as “architectural.” This means that the functions of these MSRs and their addresses remain the same for succeeding families of IA-32 processors.
- MSRs with an “MSR_” prefix are model specific with respect to address functionalities. The column “Model Availability” lists the model encoding value(s) within the Pentium 4 and Intel Xeon processor family at the specified register address. The model encoding value of a processor can be queried using CPUID. See “CPUID—CPU Identification” in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A*.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique[1]	Bit Description
Hex	Dec				
0H	0	IA32_P5_MC_ADDR	0, 1, 2, 3, 4, 6	Shared	See Appendix B.8, “MSRs in Pentium Processors.”
1H	1	IA32_P5_MC_TYPE	0, 1, 2, 3, 4, 6	Shared	See Appendix B.8, “MSRs in Pentium Processors.”
6H	6	IA32_MONITOR_FILTER_LINE_SIZE	3, 4, 6	Shared	See Section 7.11.5, “Monitor/Mwait Address Range Determination.”
10H	16	IA32_TIME_STAMP_COUNTER	0, 1, 2, 3, 4, 6	Unique	**Time Stamp Counter.** see Table B-2
					On earlier processors, only the lower 32 bits are writable. On any write to the lower 32 bits, the upper 32 bits are cleared. For processor family 0FH, models 3 and 4: all 64 bits are writable.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
17H	23	IA32_PLATFORM_ID	0, 1, 2, 3, 4, 6	Shared	**Platform ID. (R).** see Table B-2 The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	0, 1, 2, 3, 4, 6	Unique	**APIC Location and Status. (R/W)** see Table B-2. See Section 9.4.4, "Local APIC Status and Location."
2AH	42	MSR_EBC_HARD_ POWERON	0, 1, 2, 3, 4, 6	Shared	**Processor Hard Power-On Configuration.** (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0			**Output Tri-state Enabled. (R)** Indicates whether tri-state output is enabled (1) or disabled (0) as set by the strapping of SMI#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		1			**Execute BIST. (R)** Indicates whether the execution of the BIST is enabled (1) or disabled (0) as set by the strapping of INIT#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		2			**In Order Queue Depth. (R)** Indicates whether the in order queue depth for the system bus is 1 (1) or up to 12 (0) as set by the strapping of A7#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		3			**MCERR# Observation Disabled. (R)** Indicates whether MCERR# observation is enabled (0) or disabled (1) as determined by the strapping of A9#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		4			**BINIT# Observation Enabled. (R)** Indicates whether BINIT# observation is enabled (0) or disabled (1) as determined by the strapping of A10#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		6:5			**APIC Cluster ID. (R)** Contains the logical APIC cluster ID value as set by the strapping of A12# and A11#. The logical cluster ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		7			**Bus Park Disable. (R)** Indicates whether bus park is enabled (0) or disabled (1) as set by the strapping of A15#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		11:8			Reserved.
		13:12			**Agent ID. (R)** Contains the logical agent ID value as set by the strapping of BR[3:0]. The logical ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.
		63:14			Reserved.
2BH	43	MSR_EBC_SOFT_ POWERON	0, 1, 2, 3, 4, 6	Shared	**Processor Soft Power-On Configuration. (R/W)** Enables and disables processor features.
		0			**RCNT/SCNT On Request Encoding Enable. (R/W)** Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).
		1			**Data Error Checking Disable. (R/W)** Set to disable system data bus parity checking; clear to enable parity checking.
		2			**Response Error Checking Disable. (R/W)** Set to disable (default); clear to enable.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		3			**Address/Request Error Checking Disable. (R/W)** Set to disable (default); clear to enable.
		4			**Initiator MCERR# Disable. (R/W)** Set to disable MCERR# driving for initiator bus requests (default); clear to enable.
		5			**Internal MCERR# Disable. (R/W)** Set to disable MCERR# driving for initiator internal errors (default); clear to enable.
		6			**BINIT# Driver Disable.** (R/W) Set to disable BINIT# driver (default); clear to enable driver.
		63:7			Reserved.
2CH	44	MSR_EBC_ FREQUENCY_ID	2,3, 4, 6	Shared	**Processor Frequency Configuration.** The bit field layout of this MSR varies according to the MODEL value in the CPUID version information. The following bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding equal or greater than 2. (R) The field Indicates the current processor frequency configuration.
		15:0			Reserved.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		18:16			**Scalable Bus Speed. (R/W)** Indicates the intended scalable bus speed: Encoding Scalable Bus Speed 000B 100 MHz (Model 2) 000B 266 MHz (Model 3 or 4) 001B 133 MHz 010B 200 MHz 011B 166 MHz 100B 333 MHz (Model 6) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6. All other values are reserved.
		23:19			Reserved
		31:24			**Core Clock Frequency to System Bus Frequency Ratio. (R)** The processor core clock frequency to system bus frequency ratio observed at the de-assertion of the reset pin.
		63:25			Reserved.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
2CH	44	MSR_EBC_ FREQUENCY_ID	0, 1	Shared	**Processor Frequency Configuration. (R)** The bit field layout of this MSR varies according to the MODEL value of the CPUID version information. This bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding less than 2. Indicates current processor frequency configuration.
		20:0			Reserved.
		23:21			**Scalable Bus Speed. (R/W)** **Indicates the intended scalable bus speed:** Encoding Scalable Bus Speed 000B 100 MHz All others values reserved.
		63:24			Reserved.
3AH	58	IA32_FEATURE_ CONTROL	3, 4, 6	Unique	**Control Features in IA-32 Processor. (R/W).** see Table B-2 (If CPUID.01H:ECX.[bit 5])
79H	121	IA32_BIOS_UPDT_ TRIG	0, 1, 2, 3, 4, 6	Shared	**BIOS Update Trigger Register. (R/W)** see Table B-2
8BH	139	IA32_BIOS_SIGN_ID	0, 1, 2, 3, 4, 6	Unique	**BIOS Update Signature ID. (R/W)** see Table B-2
9BH	155	IA32_SMM_MONITOR_ CTL	3, 4, 6	Unique	**SMM Monitor Configuration. (R/W).** see Table B-2
FEH	254	IA32_MTRRCAP	0, 1, 2, 3, 4, 6	Unique	**MTRR Information.** See Section 10.11.1, “MTRR Feature Identification.”.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
174H	372	IA32_SYSENTER_CS	0, 1, 2, 3, 4, 6	Unique	**CS register target for CPL 0 code. (R/W).** see Table B-2 See Section 4.8.7, “Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions.”
175H	373	IA32_SYSENTER_ESP	0, 1, 2, 3, 4, 6	Unique	**Stack pointer for CPL 0 stack. (R/W).** see Table B-2 See Section 4.8.7, “Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions.”
176H	374	IA32_SYSENTER_EIP	0, 1, 2, 3, 4, 6	Unique	**CPL 0 code entry point. (R/W).** see Table B-2. See Section 4.8.7, “Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions.”
179H	377	IA32_MCG_CAP	0, 1, 2, 3, 4, 6	Unique	**Machine Check Capabilities. (R)** see Table B-2. See Section 14.3.1.1, “IA32_MCG_CAP MSR.”
17AH	378	IA32_MCG_STATUS	0, 1, 2, 3, 4, 6	Unique	**Machine Check Status. (R).** see Table B-2. **See Section 14.3.1.2, “IA32_MCG_STATUS MSR.”**
17BH	379	IA32_MCG_CTL			**Machine Check Feature Enable. (R/W).** see Table B-2 See Section 14.3.1.3, “IA32_MCG_CTL MSR.”
180H	384	MSR_MCG_RAX	0, 1, 2, 3, 4, 6	Unique	**Machine Check EAX/RAX Save State.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique[1]	Bit Description
Hex	Dec				
181H	385	MSR_MCG_RBX	0, 1, 2, 3, 4, 6	Unique	**Machine Check EBX/RBX Save State.** See Section 14.3.2.5, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
182H	386	MSR_MCG_RCX	0, 1, 2, 3, 4, 6	Unique	**Machine Check ECX/RCX Save State.** See Section 14.3.2.5, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
183H	387	MSR_MCG_RDX	0, 1, 2, 3, 4, 6	Unique	**Machine Check EDX/RDX Save State.** See Section 14.3.2.5, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
184H	388	MSR_MCG_RSI	0, 1, 2, 3, 4, 6	Unique	**Machine Check ESI/RSI Save State.** See Section 14.3.2.5, "IA32_MCG Extended Machine Check State MSRs."

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
185H	389	MSR_MCG_RDI	0, 1, 2, 3, 4, 6	Unique	**Machine Check EDI/RDI Save State.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
186H	390	MSR_MCG_RBP	0, 1, 2, 3, 4, 6	Unique	**Machine Check EBP/RBP Save State.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
187H	391	MSR_MCG_RSP	0, 1, 2, 3, 4, 6	Unique	**Machine Check ESP/RSP Save State.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
188H	392	MSR_MCG_RFLAGS	0, 1, 2, 3, 4, 6	Unique	**Machine Check EFLAGS/RFLAG Save State.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
189H	393	MSR_MCG_RIP	0, 1, 2, 3, 4, 6	Unique	**Machine Check EIP/RIP Save State.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
18AH	394	MSR_MCG_MISC	0, 1, 2, 3, 4, 6	Unique	**Machine Check Miscellaneous.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		0			**DS.** When set, the bit indicates that a page assist or page fault occurred during DS normal operation. The processors response is to shut down. The bit is used as an aid for debugging DS handling code. It is the responsibility of the user (BIOS or operating system) to clear this bit for normal operation.
		63:1			Reserved.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
18BH - 18FH	395	MSR_MCG_ RESERVED1 - MSR_MCG_ RESERVED5			Reserved.
190H	400	MSR_MCG_R8	0, 1, 2, 3, 4, 6	Unique	**Machine Check R8.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
191H	401	MSR_MCG_R9	0, 1, 2, 3, 4, 6	Unique	**Machine Check R9D/R9.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
192H	402	MSR_MCG_R10	0, 1, 2, 3, 4, 6	Unique	**Machine Check R10.** See Section 14.3.2.5, “IA32_MCG Extended Machine Check State MSRs.”
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
193H	403	MSR_MCG_R11	0, 1, 2, 3, 4, 6	Unique	**Machine Check R11.** See Section 14.3.2.5, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
194H	404	MSR_MCG_R12	0, 1, 2, 3, 4, 6	Unique	**Machine Check R12.** See Section 14.3.2.5, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
195H	405	MSR_MCG_R13	0, 1, 2, 3, 4, 6	Unique	**Machine Check R13.** See Section 14.3.2.5, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
196H	406	MSR_MCG_R14	0, 1, 2, 3, 4, 6	Unique	**Machine Check R14.** See Section 14.3.2.5, "IA32_MCG Extended Machine Check State MSRs."

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
197H	407	MSR_MCG_R15	0, 1, 2, 3, 4, 6	Unique	**Machine Check R15.** See Section 14.3.2.5, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
198H	408	IA32_PERF_STATUS	3, 4, 6	Unique	see Table B-2. See Section 13.1, "Enhanced Intel Speedstep® Technology."
199H	409	IA32_PERF_CTL	3, 4, 6	Unique	see Table B-2. See Section 13.1, "Enhanced Intel Speedstep® Technology."
19AH	410	IA32_CLOCK_ MODULATION	0, 1, 2, 3, 4, 6	Unique	**Thermal Monitor Control. (R/W)** see Table B-2. See Section 13.5.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_ INTERRUPT	0, 1, 2, 3, 4, 6	Unique	**Thermal Interrupt Control. (R/W)** See Section 13.5.2, "Thermal Monitor." and see Table B-2
19CH	412	IA32_THERM_STATUS	0, 1, 2, 3, 4, 6	Shared	**Thermal Monitor Status. (R/W)** See Section 13.5.2, "Thermal Monitor." and see Table B-2
19DH	413	MSR_THERM2_CTL			Thermal Monitor 2 Control.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
			3,	Shared	For Family F, Model 3 processors: When read, specifies the value of the target TM2 transition last written. When set, it sets the next target value for TM2 transition.
			4, 6	Shared	For Family F, Model 4 and Model 6 processors: When read, specifies the value of the target TM2 transition last written. Writes may cause #GP exceptions.
1A0H	416	IA32_MISC_ENABLE	0, 1, 2, 3, 4, 6	Shared	**Enable Miscellaneous Processor Features. (R/W)**
		0			Fast-Strings Enable. see Table B-2
		1			Reserved.
		2			**x87 FPU Fopcode Compatibility Mode Enable.**
		3			**Thermal Monitor 1 Enable.** See Section 13.5.2, “Thermal Monitor.” and see Table B-2.
		4			**Split-Lock Disable.** When set, the bit causes an #AC exception to be issued instead of a split-lock cycle. Operating systems that set this bit must align system structures to avoid split-lock scenarios. When the bit is clear (default), normal split-locks are issued to the bus.
					This debug feature is specific to the Pentium 4 processor.
		5			Reserved.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		6			**Third-Level Cache Disable.** (R/W) When set, the third-level cache is disabled; when clear (default) the third-level cache is enabled. This flag is reserved for processors that do not have a third-level cache.
					Note that the bit controls only the third-level cache; and only if overall caching is enabled through the CD flag of control register CR0, the page-level cache controls, and/or the MTRRs. See Section 10.5.4, "Disabling and Enabling the L3 Cache."
		7			**Performance Monitoring Available. (R).** see Table B-2
		8			**Suppress Lock Enable.** When set, assertion of LOCK on the bus is suppressed during a Split Lock access. When clear (default), LOCK is not suppressed.
		9			**Prefetch Queue Disable.** When set, disables the prefetch queue. When clear (default), enables the prefetch queue.
		10			**FERR# Interrupt Reporting Enable. (R/W)** When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					When this flag is set and the processor is in the stop-clock state (STPCLK# is asserted), asserting the FERR# pin signals to the processor that an interrupt (such as, INIT#, BINIT#, INTR, NMI, SMI#, or RESET#) is pending and that the processor should return to normal operation to handle the interrupt. This flag does not affect the normal operation of the FERR# pin (to indicate an unmasked floating-point error) when the STPCLK# pin is not asserted.
		11			**Branch Trace Storage Unavailable (BTS_UNAVILABLE). (R).** see Table B-2 When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.
		12			**PEBS_UNAVILABLE: Precise Event Based Sampling Unavailable. (R).** see Table B-2 When set, the processor does not support precise event-based sampling (PEBS); when clear, PEBS is supported.
		13	3		**TM2 Enable. (R/W)** When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		17:14			Reserved.
		18	3, 4, 6		**ENABLE MONITOR FSM. (R/W)** see Table B-2
		19			**Adjacent Cache Line Prefetch Disable. (R/W)** When set to 1, the processor fetches the cache line of the 128-byte sector containing currently required data. When set to 0, the processor fetches both cache lines in the sector.
					Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.
		21:20			Reserved.
		22	3, 4, 6		**Limit CPUID MAXVAL. (R/W)** see Table B-2

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					Setting this can cause unexpected behavior to software that depends on the availability of CPUID leaves greater than 3.
		23		Shared	**xTPR Message Disable. (R/W)** see Table B-2.
		24			**L1 Data Cache Context Mode. (R/W)** When set, the L1 data cache is placed in shared mode; when clear (default), the cache is placed in adaptive mode. This bit is only enabled for IA-32 processors that support Intel Hyper-Threading Technology. See Section 10.5.6, “L1 Data Cache Context Mode.” When L1 is running in adaptive mode and CR3s are identical, data in L1 is shared across logical processors. Otherwise, L1 is not shared and cache use is competitive. If the Context ID feature flag (ECX[10]) is set to 0 after executing CPUID with EAX = 1, the ability to switch modes is not supported. BIOS must not alter the contents of IA32_MISC_ENABLE[24].
		33:25			Reserved.
		34		Unique	**XD Bit Disable. (R/W)** see Table B-2.
		63:35			Reserved.
1A1H	417	MSR_PLATFORM_BRV	3, 4, 6	Shared	**Platform Feature Requirements. (R)**
		17:0			Reserved.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		18			**PLATFORM Requirements.** When set to 1, indicates the processor has specific platform requirements. The details of the platform requirements are listed in the respective data sheets of the processor.
		63:19			Reserved.
1D7H	471	MSR_LER_FROM_LIP	0, 1, 2, 3, 4, 6	Unique	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 18.6.7, "Last Exception Records."
		31:0			**From Linear IP.** Linear address of the last branch instruction.
		63:32			Reserved.
1D7H	471	63:0		Unique	**From Linear IP.** Linear address of the last branch instruction (If IA-32e mode is active).
1D8H	472	MSR_LER_TO_LIP	0, 1, 2, 3, 4, 6	Unique	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 18.6.7, "Last Exception Records."

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		31:0			**From Linear IP.** Linear address of the target of the last branch instruction.
		63:32			Reserved.
1D8H	472	63:0		Unique	**From Linear IP.** Linear address of the target of the last branch instruction (If IA-32e mode is active).
1D9H	473	MSR_DEBUGCTLA	0, 1, 2, 3, 4, 6	Unique	**Debug Control. (R/W)** Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 18.6.2, “MSR_DEBUGCTLA MSR.”
1DAH	474	MSR_LASTBRANCH _TOS	0, 1, 2, 3, 4, 6	Unique	**Last Branch Record Stack TOS. (R)** Contains an index (0-3 or 0-15) that points to the top of the last branch record stack (that is, that points the index of the MSR containing the most recent branch record). See Section 18.6.3, “LBR Stack for Processors Based on Intel NetBurst Microarchitecture”; and addresses 1DBH-1DEH and 680H-68FH.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
1DBH	475	MSR_LASTBRANCH_0	0, 1, 2	Unique	**Last Branch Record 0. (R/W)** One of four last branch record registers on the last branch record stack. It contains pointers to the source and destination instruction for one of the last four branches, exceptions, or interrupts that the processor took. MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3 at 1DBH-1DEH are available only on family 0FH, models 0H-02H. They have been replaced by the MSRs at 680H-68FH and 6C0H-6CFH.
					See Section 18.6, "Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)."
1DDH	477	MSR_LASTBRANCH_2	0, 1, 2	Unique	**Last Branch Record 2.** See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DEH	478	MSR_LASTBRANCH_3	0, 1, 2	Unique	**Last Branch Record 3.** See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
200H	512	IA32_MTRR_PHYSBASE0	0, 1, 2, 3, 4, 6	Shared	**Variable Range Base MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
201H	513	IA32_MTRR_PHYSMASK0	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
202H	514	IA32_MTRR_PHYSBASE1	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique[1]	Bit Description
Hex	Dec				
203H	515	IA32_MTRR_ PHYSMASK1	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
204H	516	IA32_MTRR_ PHYSBASE2	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
205H	517	IA32_MTRR_ PHYSMASK2	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs".
206H	518	IA32_MTRR_ PHYSBASE3	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
207H	519	IA32_MTRR_ PHYSMASK3	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
208H	520	IA32_MTRR_ PHYSBASE4	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
209H	521	IA32_MTRR_ PHYSMASK4	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
20AH	522	IA32_MTRR_ PHYSBASE5	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
20BH	523	IA32_MTRR_ PHYSMASK5	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
20CH	524	IA32_MTRR_ PHYSBASE6	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
20DH	525	IA32_MTRR_ PHYSMASK6	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
20EH	526	IA32_MTRR_ PHYSBASE7	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
20FH	527	IA32_MTRR_ PHYSMASK7	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 10.11.2.3, "Variable Range MTRRs."
250H	592	IA32_MTRR_FIX64K_ 00000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, "Fixed Range MTRRs."
258H	600	IA32_MTRR_FIX16K_ 80000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, "Fixed Range MTRRs."
259H	601	IA32_MTRR_FIX16K_ A0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, "Fixed Range MTRRs."
268H	616	IA32_MTRR_FIX4K_ C0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, "Fixed Range MTRRs."
269H	617	IA32_MTRR_FIX4K_ C8000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, "Fixed Range MTRRs".
26AH	618	IA32_MTRR_FIX4K_ D0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, "Fixed Range MTRRs".
26BH	619	IA32_MTRR_FIX4K_ D8000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, "Fixed Range MTRRs."
26CH	620	IA32_MTRR_FIX4K_ E0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, "Fixed Range MTRRs."
26DH	621	IA32_MTRR_FIX4K_ E8000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, "Fixed Range MTRRs."

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
26EH	622	IA32_MTRR_FIX4K_ F0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, “Fixed Range MTRRs.”
26FH	623	IA32_MTRR_FIX4K_ F8000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 10.11.2.2, “Fixed Range MTRRs.”
277H	631	IA32_CR_PAT	0, 1, 2, 3, 4, 6	Unique	**Page Attribute Table.** See Section 10.11.2.2, “Fixed Range MTRRs.”
2FFH	767	IA32_MTRR_DEF_ TYPE	0, 1, 2, 3, 4, 6	Shared	**Default Memory Types. (R/W)** see Table B-2 See Section 10.11.2.1, “IA32_MTRR_DEF_TYPE MSR.”
300H	768	MSR_BPU_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
301H	769	MSR_BPU_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
302H	770	MSR_BPU_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
303H	771	MSR_BPU_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
304H	772	MSR_MS_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
305H	773	MSR_MS_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
306H	774	MSR_MS_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
307H	775	MSR_MS_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
308H	776	MSR_FLAME_ COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
309H	777	MSR_FLAME_ COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
30AH	778	MSR_FLAME_ COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
30BH	779	MSR_FLAME_ COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
30CH	780	MSR_IQ_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
30DH	781	MSR_IQ_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
30EH	782	MSR_IQ_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
30FH	783	MSR_IQ_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
310H	784	MSR_IQ_COUNTER4	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
311H	785	MSR_IQ_COUNTER5	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.2, “Performance Counters.”
360H	864	MSR_BPU_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
361H	865	MSR_BPU_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
362H	866	MSR_BPU_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
363H	867	MSR_BPU_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
364H	868	MSR_MS_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
365H	869	MSR_MS_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
366H	870	MSR_MS_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
367H	871	MSR_MS_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
368H	872	MSR_FLAME_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
369H	873	MSR_FLAME_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
36AH	874	MSR_FLAME_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
36BH	875	MSR_FLAME_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
36CH	876	MSR_IQ_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
36DH	877	MSR_IQ_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
36EH	878	MSR_IQ_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
36FH	879	MSR_IQ_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
370H	880	MSR_IQ_CCCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
371H	881	MSR_IQ_CCCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.3, “CCCR MSRs.”
3A0H	928	MSR_BSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, “ESCR MSRs.”
3A1H	929	MSR_BSU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, “ESCR MSRs.”
3A2H	930	MSR_FSB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, “ESCR MSRs.”
3A3H	931	MSR_FSB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, “ESCR MSRs.”
3A4H	932	MSR_FIRM_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, “ESCR MSRs.”
3A5H	933	MSR_FIRM_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, “ESCR MSRs.”
3A6H	934	MSR_FLAME_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, “ESCR MSRs.”
3A7H	935	MSR_FLAME_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, “ESCR MSRs.”

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
3A8H	936	MSR_DAC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3A9H	937	MSR_DAC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3AAH	938	MSR_MOB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3ABH	939	MSR_MOB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3ACH	940	MSR_PMH_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3ADH	941	MSR_PMH_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3AEH	942	MSR_SAAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3AFH	943	MSR_SAAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3B0H	944	MSR_U2L_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3B1H	945	MSR_U2L_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3B2H	946	MSR_BPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3B3H	947	MSR_BPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3B4H	948	MSR_IS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3B5H	949	MSR_IS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3B6H	950	MSR_ITLB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3B7H	951	MSR_ITLB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3B8H	952	MSR_CRU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
3B9H	953	MSR_CRU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3BAH	954	MSR_IQ_ESCR0	0, 1, 2	Shared	See Section 18.16.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BBH	955	MSR_IQ_ESCR1	0, 1, 2	Shared	See Section 18.16.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BCH	956	MSR_RAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3BDH	957	MSR_RAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3BEH	958	MSR_SSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3C0H	960	MSR_MS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3C1H	961	MSR_MS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3C2H	962	MSR_TBPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3C3H	963	MSR_TBPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3C4H	964	MSR_TC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3C5H	965	MSR_TC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3C8H	968	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3C9H	969	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
3CAH	970	MSR_ALF_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3CBH	971	MSR_ALF_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3CCH	972	MSR_CRU_ESCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3CDH	973	MSR_CRU_ESCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3E0H	992	MSR_CRU_ESCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3E1H	993	MSR_CRU_ESCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3F0H	1008	MSR_TC_PRECISE _EVENT	0, 1, 2, 3, 4, 6	Shared	See Section 18.16.1, "ESCR MSRs."
3F1H	1009	MSR_PEBS_ENABLE	0, 1, 2, 3, 4, 6	Shared	**Precise Event-Based Sampling (PEBS). (R/W)** Controls the enabling of precise event sampling and replay tagging.
		12:0			See Table A-12.
		23:13			Reserved.
		24			**UOP Tag.** Enables replay tagging when set.
		25			**ENABLE_PEBS_MY_THR. (R/W)** Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 18.17.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is called ENABLE_PEBS in IA-32 processors that do not support Hyper-Threading Technology.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		26			**ENABLE_PEBS_OTH_THR. (R/W)** Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 18.17.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is reserved for IA-32 processors that do not support Hyper-Threading Technology.
		63:27			Reserved.
3F2H	1010	MSR_PEBS_MATRIX _VERT	0, 1, 2, 3, 4, 6	Shared	See Table A-12.
400H	1024	IA32_MC0_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MC0_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC0_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC0_STATUS register is clear.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
405H	1029	IA32_MC1_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.2, “IA32_MCi_STATUS MSRS.”
406H	1030	IA32_MC1_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
407H	1031	IA32_MC1_MISC		Shared	See Section 14.3.2.4, “IA32_MCi_MISC MSRs.” The IA32_MC1_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
409H	1033	IA32_MC2_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.2, “IA32_MCi_STATUS MSRS.”

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
40AH	1034	IA32_MC2_ADDR			See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40BH	1035	IA32_MC2_MISC			See Section 14.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC3_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique[1]	Bit Description
Hex	Dec				
40FH	1039	IA32_MC3_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.4, “IA32_MCi_MISC MSRs.” The IA32_MC3_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC4_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
411H	1041	IA32_MC4_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 14.3.2.2, “IA32_MCi_STATUS MSRS.”
412H	1042	IA32_MC4_ADDR			See Section 14.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC4_MISC			See Section 14.3.2.4, “IA32_MCi_MISC MSRs.” The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC4_STATUS register is clear.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	3, 4, 6	Unique	**Reporting Register of Basic VMX Capabilities. (R/O).** see Table B-2. See Appendix G.1, “Basic VMX Information”
481H	1153	IA32_VMX_PINBASED _CTLS	3, 4, 6	Unique	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O).** see Table B-2. See Appendix G.3, “VM-Execution Controls”
482H	1154	IA32_VMX_ PROCBASED_CTLS	3, 4, 6	Unique	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix G.3, “VM-Execution Controls” and see Table B-2.
483H	1155	IA32_VMX_EXIT_CTLS	3, 4, 6	Unique	**Capability Reporting Register of VM-exit Controls. (R/O)** See Appendix G.4, “VM-Exit Controls” and see Table B-2.
484H	1156	IA32_VMX_ENTRY_ CTLS	3, 4, 6	Unique	**Capability Reporting Register of VM-entry Controls. (R/O)** See Appendix G.5, “VM-Entry Controls” and see Table B-2.
485H	1157	IA32_VMX_MISC	3, 4, 6	Unique	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Appendix G.6, “Miscellaneous Data” and see Table B-2.
486H	1158	IA32_VMX_CR0_ FIXED0	3, 4, 6	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Appendix G.7, “VMX-Fixed Bits in CR0” and see Table B-2.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
487H	1159	IA32_VMX_CR0_ FIXED1	3, 4, 6	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Appendix G.7, “VMX-Fixed Bits in CR0” and see Table B-2.
488H	1160	IA32_VMX_CR4_ FIXED0	3, 4, 6	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Appendix G.8, “VMX-Fixed Bits in CR4” and see Table B-2.
489H	1161	IA32_VMX_CR4_ FIXED1	3, 4, 6	Unique	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Appendix G.8, “VMX-Fixed Bits in CR4” and see Table B-2.
48AH	1162	IA32_VMX_VMCS_ ENUM	3, 4, 6	Unique	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Appendix G.9, “VMCS Enumeration” and see Table B-2.
48BH	1163	IA32_VMX_ PROCBASED_CTLS2	3, 4, 6	Unique	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix G.3, “VM-Execution Controls” and see Table B-2.
600H	1536	IA32_DS_AREA	0, 1, 2, 3, 4, 6	Unique	**DS Save Area. (R/W).** see Table B-2. See Section 18.16.4, “Debug Store (DS) Mechanism.”
680H	1664	MSR_LASTBRANCH _0_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 0. (R/W)** One of 16 pairs of last branch record registers on the last branch record stack (680H-68FH). This part of the stack contains pointers to the source instruction for one of the last 16 branches, exceptions, or interrupts taken by the processor.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					The MSRs at 680H-68FH, 6C0H-6CfH are not available in processor releases before family 0FH, model 03H. These MSRs replace MSRs previously located at 1DBH-1DEH.which performed the same function for early releases.
					See Section 18.6, "Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)."
681H	1665	MSR_LASTBRANCH _1_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 1.** See description of MSR_LASTBRANCH_0 at 680H.
682H	1666	MSR_LASTBRANCH _2_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 2.** See description of MSR_LASTBRANCH_0 at 680H.
683H	1667	MSR_LASTBRANCH _3_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 3.** See description of MSR_LASTBRANCH_0 at 680H.
684H	1668	MSR_LASTBRANCH _4_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 4.** See description of MSR_LASTBRANCH_0 at 680H.
685H	1669	MSR_LASTBRANCH _5_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 5.** See description of MSR_LASTBRANCH_0 at 680H.
686H	1670	MSR_LASTBRANCH _6_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 6.** See description of MSR_LASTBRANCH_0 at 680H.
687H	1671	MSR_LASTBRANCH _7_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 7.** See description of MSR_LASTBRANCH_0 at 680H.
688H	1672	MSR_LASTBRANCH _8_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 8.** See description of MSR_LASTBRANCH_0 at 680H.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail- ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
689H	1673	MSR_LASTBRANCH _9_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 9.** See description of MSR_LASTBRANCH_0 at 680H.
68AH	1674	MSR_LASTBRANCH _10_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 10.** See description of MSR_LASTBRANCH_0 at 680H.
68BH	1675	MSR_LASTBRANCH _11_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 11.** See description of MSR_LASTBRANCH_0 at 680H.
68CH	1676	MSR_LASTBRANCH _12_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 12.** See description of MSR_LASTBRANCH_0 at 680H.
68DH	1677	MSR_LASTBRANCH _13_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 13.** See description of MSR_LASTBRANCH_0 at 680H.
68EH	1678	MSR_LASTBRANCH _14_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 14.** See description of MSR_LASTBRANCH_0 at 680H.
68FH	1679	MSR_LASTBRANCH _15_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 15.** See description of MSR_LASTBRANCH_0 at 680H.
6C0H	1728	MSR_LASTBRANCH _0_TO_LIP	3, 4, 6	Unique	**Last Branch Record 0. (R/W)** One of 16 pairs of last branch record registers on the last branch record stack (6C0H-6CFH). This part of the stack contains pointers to the destination instruction for one of the last 16 branches, exceptions, or interrupts that the processor took. See Section 18.6, “Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture).”

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
6C1H	1729	MSR_LASTBRANCH_1_TO_LIP	3, 4, 6	Unique	**Last Branch Record 1.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C2H	1730	MSR_LASTBRANCH_2_TO_LIP	3, 4, 6	Unique	**Last Branch Record 2.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C3H	1731	MSR_LASTBRANCH_3_TO_LIP	3, 4, 6	Unique	**Last Branch Record 3.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C4H	1732	MSR_LASTBRANCH_4_TO_LIP	3, 4, 6	Unique	**Last Branch Record 4.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C5H	1733	MSR_LASTBRANCH_5_TO_LIP	3, 4, 6	Unique	**Last Branch Record 5.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C6H	1734	MSR_LASTBRANCH_6_TO_LIP	3, 4, 6	Unique	**Last Branch Record 6.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C7H	1735	MSR_LASTBRANCH_7_TO_LIP	3, 4, 6	Unique	**Last Branch Record 7.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C8H	1736	MSR_LASTBRANCH_8_TO_LIP	3, 4, 6	Unique	**Last Branch Record 8.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C9H	1737	MSR_LASTBRANCH_9_TO_LIP	3, 4, 6	Unique	**Last Branch Record 9.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CAH	1738	MSR_LASTBRANCH_10_TO_LIP	3, 4, 6	Unique	**Last Branch Record 10.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CBH	1739	MSR_LASTBRANCH_11_TO_LIP	3, 4, 6	Unique	**Last Branch Record 11.** See description of MSR_LASTBRANCH_0 at 6C0H.

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
6CCH	1740	MSR_LASTBRANCH_12_TO_LIP	3, 4, 6	Unique	**Last Branch Record 12.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CDH	1741	MSR_LASTBRANCH_13_TO_LIP	3, 4, 6	Unique	**Last Branch Record 13.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CEH	1742	MSR_LASTBRANCH_14_TO_LIP	3, 4, 6	Unique	**Last Branch Record 14.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CFH	1743	MSR_LASTBRANCH_15_TO_LIP	3, 4, 6	Unique	**Last Branch Record 15.** See description of MSR_LASTBRANCH_0 at 6C0H.
C000_0080H		IA32_EFER	3, 4, 6	Unique	**Extended Feature Enables.** see Table B-2
C000_0081H		IA32_STAR	3, 4, 6	Unique	**System Call Target Address. (R/W)** see Table B-2
C000_0082H		IA32_LSTAR	3, 4, 6	Unique	**IA-32e Mode System Call Target Address. (R/W)** see Table B-2
C000_0084H		IA32_FMASK	3, 4, 6	Unique	**System Call Flag Mask. (R/W)** see Table B-2
C000_0100H		IA32_FS_BASE	3, 4, 6	Unique	**Map of BASE Address of FS. (R/W)** see Table B-2
C000_0101H		IA32_GS_BASE	3, 4, 6	Unique	**Map of BASE Address of GS. (R/W)** see Table B-2
C000_0102H		IA32_KERNEL_GSBASE	3, 4, 6	Unique	**Swap Target of BASE Address of GS. (R/W)** see Table B-2

Table B-5. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				

NOTES

1. For HT-enabled processors, there may be more than one logical processors per physical unit. If an MSR is Shared, this means that one MSR is shared between logical processors. If an MSR is unique, this means that each logical processor has its own MSR.

B.4.1 MSRs Unique to Intel Xeon Processor MP with L3 Cache

The MSRs listed in Table B-6 apply to Intel Xeon Processor MP with up to 8MB level three cache. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 3 or 4 (See CPUID instruction for more details.).

Table B-6. MSRs Unique to 64-bit Intel Xeon Processor MP with Up to an 8 MB L3 Cache

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique	Bit Description
107CCH		MSR_IFSB_BUSQ0	3, 4	Shared	**IFSB BUSQ Event Control and Counter Register. (R/W)** See Section 18.21, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache.”
107CDH		MSR_IFSB_BUSQ1	3, 4	Shared	**IFSB BUSQ Event Control and Counter Register. (R/W)**
107CEH		MSR_IFSB_SNPQ0	3, 4	Shared	**IFSB SNPQ Event Control and Counter Register. (R/W)** See Section 18.21, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache.”

Table B-6. MSRs Unique to 64-bit Intel Xeon Processor MP with Up to an 8 MB L3 Cache (Contd.)

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique	Bit Description
107CFH		MSR_IFSB_SNPQ1	3, 4	Shared	**IFSB SNPQ Event Control and Counter Register. (R/W)**
107D0H		MSR_EFSB_DRDY0	3, 4	Shared	**EFSB DRDY Event Control and Counter Register. (R/W)** See Section 18.21, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache” for details.
107D1H		MSR_EFSB_DRDY1	3, 4	Shared	**EFSB DRDY Event Control and Counter Register. (R/W)**
107D2H		MSR_IFSB_CTL6	3, 4	Shared	**IFSB Latency Event Control Register. (R/W)** See Section 18.21, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache” for details.
107D3H		MSR_IFSB_CNTR7	3, 4	Shared	**IFSB Latency Event Counter Register. (R/W)** See Section 18.21, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache.”

The MSRs listed in Table B-7 apply to Intel Xeon Processor 7100 series. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 6 (See CPUID instruction for more details.). The performance monitoring MSRs listed in Table B-7 are shared between logical processors in the same core, but are replicated for each core.

Table B-7. MSRs Unique to Intel Xeon Processor 7100 Series

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique	Bit Description
107CCH		MSR_EMON_L3_CTR_CTL0	6	Shared	**GBUSQ Event Control and Counter Register. (R/W)** See Section 18.22, "Performance Monitoring on Dual-Core Intel Xeon Processor 7100 Series."
107CDH		MSR_EMON_L3_CTR_CTL1	6	Shared	**GBUSQ Event Control and Counter Register. (R/W)**
107CEH		MSR_EMON_L3_CTR_CTL2	6	Shared	**GSNPQ Event Control and Counter Register. (R/W)** See Section 18.22, "Performance Monitoring on Dual-Core Intel Xeon Processor 7100 Series."
107CFH		MSR_EMON_L3_CTR_CTL3	6	Shared	**GSNPQ Event Control and Counter Register (R/W)**
107D0H		MSR_EMON_L3_CTR_CTL4	6	Shared	**FSB Event Control and Counter Register. (R/W)** See Section 18.22, "Performance Monitoring on Dual-Core Intel Xeon Processor 7100 Series" for details.
107D1H		MSR_EMON_L3_CTR_CTL5	6	Shared	**FSB Event Control and Counter Register. (R/W)**
107D2H		MSR_EMON_L3_CTR_CTL6	6	Shared	**FSB Event Control and Counter Register. (R/W)**
107D3H		MSR_EMON_L3_CTR_CTL7	6	Shared	**FSB Event Control and Counter Register. (R/W)**

B.5 MSRS IN INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Model-specific registers (MSRs) for Intel Core Solo, Intel Core Duo processors, and Dual-core Intel Xeon processor LV are listed in Table B-8. The column "Shared/Unique" applies to Intel Core Duo processor. "Unique" means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. "Shared" means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	P5_MC_ADDR	Unique	See Appendix B.8, "MSRs in Pentium Processors." and see Table B-2
1H	1	P5_MC_TYPE	Unique	See Appendix B.8, "MSRs in Pentium Processors." and see Table B-2
6H	6	IA32_MONITOR_FILTER_SIZE	Unique	See Section 7.11.5, "Monitor/Mwait Address Range Determination." and see Table B-2
10H	16	IA32_TIME_STAMP_COUNTER	Unique	See Section 18.10, "Time-Stamp Counter." and see Table B-2
17H	23	IA32_PLATFORM_ID	Shared	**Platform ID. (R)** see Table B-2 The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	Unique	See Section 9.4.4, "Local APIC Status and Location." and see Table B-2
2AH	42	MSR_EBL_CR_POWERON	Shared	**Processor Hard Power-On Configuration. (R/W)** Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved.
		1		**Data Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		2		**Response Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		3		**MCERR# Drive Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		4		**Address Parity Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		6: 5		Reserved
		7		**BINIT# Driver Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		8		**Output Tri-state Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		9		**Execute BIST. (R/O)** 1 = Enabled; 0 = Disabled
		10		**MCERR# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		11		Reserved
		12		**BINIT# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		13		Reserved
		14		**1 MByte Power on Reset Vector. (R/O)** 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		**APIC Cluster ID.** (R/O)
		18		**System Bus Frequency. (R/O)** 0 = 100 MHz 1 = Reserved
		19		Reserved.
		21: 20		**Symmetric Arbitration ID. (R/O)**
		26:22		**Clock Frequency Ratio. (R/O)**

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_ CONTROL	Unique	**Control Features in IA-32 Processor. (R/W)** see Table B-2
40H	64	MSR_ LASTBRANCH_0	Unique	**Last Branch Record 0. (R/W)** One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the 'to' address. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 18.8, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_ LASTBRANCH_1	Unique	**Last Branch Record 1. (R/W)** See description of MSR_LASTBRANCH_0.
42H	66	MSR_ LASTBRANCH_2	Unique	**Last Branch Record 2. (R/W)** See description of MSR_LASTBRANCH_0.
43H	67	MSR_ LASTBRANCH_3	Unique	**Last Branch Record 3. (R/W)** See description of MSR_LASTBRANCH_0.
44H	68	MSR_ LASTBRANCH_4	Unique	**Last Branch Record 4. (R/W)** See description of MSR_LASTBRANCH_0.
45H	69	MSR_ LASTBRANCH_5	Unique	**Last Branch Record 5. (R/W)** See description of MSR_LASTBRANCH_0.
46H	70	MSR_ LASTBRANCH_6	Unique	**Last Branch Record 6. (R/W)** See description of MSR_LASTBRANCH_0.
47H	71	MSR_ LASTBRANCH_7	Unique	**Last Branch Record 7. (R/W)** See description of MSR_LASTBRANCH_0.
79H	121	IA32_BIOS_ UPDT_TRIG	Unique	**BIOS Update Trigger Register (R/W).** see Table B-2
8BH	139	IA32_BIOS_ SIGN_ID	Unique	**BIOS Update Signature ID (RO).** see Table B-2
C1H	193	IA32_PMC0	Unique	**Performance counter register.** see Table B-2
C2H	194	IA32_PMC1	Unique	**Performance counter register.** see Table B-2

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed. (RO)** This field indicates the scaleable bus clock speed:
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 101B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.
		63:3		Reserved
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count. (RW).** see Table B-2
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count. (RW).** see Table B-2
FEH	254	IA32_MTRRCAP	Unique	see Table B-2
11EH	281	MSR_BBL_CR_ CTL3	Shared	
		0		**L2 Hardware Enabled. (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		23		**L2 Not Present. (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER _CS	Unique	see Table B-2
175H	373	IA32_SYSENTER _ESP	Unique	see Table B-2
176H	374	IA32_SYSENTER _EIP	Unique	see Table B-2
179H	377	IA32_MCG_CAP	Unique	see Table B-2
17AH	378	IA32_MCG_ STATUS	Unique	
		0		**RIPV.** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted
		1		**EIPV.** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP.** When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		63:3		Reserved.
186H	390	IA32_ PERFEVTSEL0	Unique	see Table B-2
187H	391	IA32_ PERFEVTSEL1	Unique	see Table B-2
198H	408	IA32_PERF_STAT US	Shared	see Table B-2
199H	409	IA32_PERF_CTL	Unique	see Table B-2
19AH	410	IA32_CLOCK_ MODULATION	Unique	**Clock Modulation. (R/W)** see Table B-2
19BH	411	IA32_THERM_ INTERRUPT	Unique	**Thermal Interrupt Control. (R/W)** see Table B-2 See Section 13.5.2, “Thermal Monitor.”
19CH	412	IA32_THERM_ STATUS	Unique	**Thermal Monitor Status. (R/W)** see Table B-2. See Section 13.5.2, “Thermal Monitor”.
19DH	413	MSR_THERM2_ CTL	Unique	
		15:0		Reserved.
		16		**TM_SELECT. (R/W)** Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16		Reserved.
1A0	416	IA32_MISC_ ENABLE		**Enable Miscellaneous Processor Features.** (R/W) Allows a variety of processor functions to be enabled and disabled.

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2:0		Reserved.
		3	Unique	**Automatic Thermal Control Circuit Enable. (R/W)** see Table B-2
		6:4		Reserved
		7	Shared	**Performance Monitoring Available. (R).** see Table B-2
		9:8		Reserved
		10	Shared	**FERR# Multiplexing Enable. (R/W)** 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	**Branch Trace Storage Unavailable. (RO). see Table B-2**
		12		Reserved.
		13	Shared	**TM2 Enable. (R/W)** When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	**Dec**			
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		15:14		Reserved
		16	Shared	**Enhanced Intel SpeedStep Technology Enable. (R/W)** 1 = Enhanced Intel SpeedStep Technology enabled
		18	Shared	**ENABLE MONITOR FSM. (R/W)** see Table B-2
		19		**Reserved.**
		22	Shared	**Limit CPUID Maxval. (R/W)** see Table B-2. Setting this bit may cause behavior in software that depends on the availability of CPUID leaves greater than 3.
		33:23		Reserved.
		34	Shared	**XD Bit Disable. (R/W)** see Table B-2
		63:35		Reserved.
1C9H	457	MSR_ LASTBRANCH_ TOS	Unique	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0 (at 40H)

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Unique	**Debug Control. (R/W)** Controls how several debug features are used. Bit definitions are discussed in the referenced section.
1DDH	477	MSR_LER_FROM_LIP	Unique	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1E0H	480	ROB_CR_BKUPTMPDR6	Unique	
		1:0		Reserved
		2		Fast String Enable bit. (Default, enabled)
200H	512	MTRRphysBase0	Unique	
201H	513	MTRRphysMask0	Unique	
202H	514	MTRRphysBase1	Unique	
203H	515	MTRRphysMask1	Unique	
204H	516	MTRRphysBase2	Unique	
205H	517	MTRRphysMask2	Unique	
206H	518	MTRRphysBase3	Unique	
207H	519	MTRRphysMask3	Unique	
208H	520	MTRRphysBase4	Unique	
209H	521	MTRRphysMask4	Unique	
20AH	522	MTRRphysBase5	Unique	
20BH	523	MTRRphysMask5	Unique	

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
20CH	524	MTRRphysBase6	Unique	
20DH	525	MTRRphysMask6	Unique	
20EH	526	MTRRphysBase7	Unique	
20FH	527	MTRRphysMask7	Unique	
250H	592	MTRRfix64K_ 00000	Unique	
258H	600	MTRRfix16K_ 80000	Unique	
259H	601	MTRRfix16K_ A0000	Unique	
268H	616	MTRRfix4K_ C0000	Unique	
269H	617	MTRRfix4K_ C8000	Unique	
26AH	618	MTRRfix4K_ D0000	Unique	
26BH	619	MTRRfix4K_ D8000	Unique	
26CH	620	MTRRfix4K_ E0000	Unique	
26DH	621	MTRRfix4K_ E8000	Unique	
26EH	622	MTRRfix4K_ F0000	Unique	
26FH	623	MTRRfix4K_ F8000	Unique	
2FFH	767	IA32_MTRR_DEF_ TYPE	Unique	**Default Memory Types. (R/W). see Table B-2.** See Section 10.11.2.1, “IA32_MTRR_DEF_TYPE MSR.”
400H	1024	IA32_MC0_CTL	Unique	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
401H	1025	IA32_MC0_ STATUS	Unique	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Unique	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_ STATUS	Unique	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Unique	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	Unique	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_ STATUS	Unique	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	Unique	See Section 14.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_ STATUS	Unique	See Section 14.3.2.2, "IA32_MCi_STATUS MSRS."

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
40EH	1038	MSR_MC4_ADDR	Unique	See Section 14.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC3_CTL		See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
411H	1041	MSR_MC3_ STATUS		See Section 14.3.2.2, “IA32_MCi_STATUS MSRS.”
412H	1042	MSR_MC3_ADDR	Unique	See Section 14.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	MSR_MC3_MISC	Unique	
414H	1044	MSR_MC5_CTL	Unique	
415H	1045	MSR_MC5_ STATUS	Unique	
416H	1046	MSR_MC5_ADDR	Unique	
417H	1047	MSR_MC5_MISC	Unique	
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities. (R/O). see Table B-2** See Appendix G.1, “Basic VMX Information” (If CPUID.01H:ECX.[bit 9])
481H	1153	IA32_VMX_PINBASED_CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** See Appendix G.3, “VM-Execution Controls” (If CPUID.01H:ECX.[bit 9])

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
482H	1154	IA32_VMX_PROCBASED_CTLS	Unique	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix G.3, "VM-Execution Controls" (If CPUID.01H:ECX.[bit 9])
483H	1155	IA32_VMX_EXIT_CTLS	Unique	**Capability Reporting Register of VM-exit Controls. (R/O)** See Appendix G.4, "VM-Exit Controls" (If CPUID.01H:ECX.[bit 9])
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls. (R/O)** See Appendix G.5, "VM-Entry Controls" (If CPUID.01H:ECX.[bit 9])
485H	1157	IA32_VMX_MISC	Unique	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Appendix G.6, "Miscellaneous Data" (If CPUID.01H:ECX.[bit 9])
486H	1158	IA32_VMX_CR0_FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Appendix G.7, "VMX-Fixed Bits in CR0" (If CPUID.01H:ECX.[bit 9])
487H	1159	IA32_VMX_CR0_FIXED1	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Appendix G.7, "VMX-Fixed Bits in CR0" (If CPUID.01H:ECX.[bit 9])
488H	1160	IA32_VMX_CR4_FIXED0	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Appendix G.8, "VMX-Fixed Bits in CR4" (If CPUID.01H:ECX.[bit 9])
489H	1161	IA32_VMX_CR4_FIXED1	Unique	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Appendix G.8, "VMX-Fixed Bits in CR4" (If CPUID.01H:ECX.[bit 9])

Table B-8. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
48AH	1162	IA32_VMX_ VMCS_ENUM	Unique	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Appendix G.9, "VMCS Enumeration" (If CPUID.01H:ECX.[bit 9])
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Unique	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix G.3, "VM-Execution Controls" (If CPUID.01H:ECX.[bit 9] and IA32_VMX_PROCBASED_CTLS[bit 63])
600H	1536	IA32_DS_AREA	Unique	**DS Save Area. (R/W)** **see Table B-2**. See Section 18.16.4, "Debug Store (DS) Mechanism."
		31:0		**DS Buffer Management Area.** Linear address of the first byte of the DS buffer management area.
		63:32		Reserved.
C000_ 0080H		IA32_EFER	Unique	**see Table B-2**
		10:0		Reserved.
		11		**Execute Disable Bit Enable.**
		63:12		Reserved

B.6 MSRS IN THE PENTIUM M PROCESSOR

Model-specific registers (MSRs) for the Pentium M processor are similar to those described in Section B.7 for P6 family processors. The following table describes new MSRs and MSRs whose behavior has changed on the Pentium M processor.

Table B-9. MSRs in Pentium M Processors

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Appendix B.8, “MSRs in Pentium Processors.”
1H	1	P5_MC_TYPE	See Appendix B.8, “MSRs in Pentium Processors.”
10H	16	IA32_TIME_STAMP_ COUNTER	See Section 18.10, “Time-Stamp Counter.” and **see Table B-2**
17H	23	IA32_PLATFORM_ID	**Platform ID. (R). see Table B-2** The operating system can use this MSR to determine “slot” information for the processor and the proper microcode update to load.
2AH	42	MSR_EBL_CR_POWERON	**Processor Hard Power-On Configuration.** (R/W) Enables and disables processor features. (R) Indicates current processor configuration.
		0	Reserved.
		1	**Data Error Checking Enable. (R)** 0 = Disabled Always 0 on the Pentium M processor.
		2	**Response Error Checking Enable.** (R) 0 = Disabled Always 0 on the Pentium M processor.
		3	**MCERR# Drive Enable. (R)** 0 = Disabled Always 0 on the Pentium M processor.
		4	**Address Parity Enable. (R)** 0 = Disabled Always 0 on the Pentium M processor.
		6:5	Reserved.
		7	**BINIT# Driver Enable. (R)** 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		8	**Output Tri-state Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		9	**Execute BIST. (R/O)** 1 = Enabled; 0 = Disabled

Table B-9. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		10	**MCERR# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		11	Reserved.
		12	**BINIT# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		13	**Reserved**
		14	**1 MByte Power on Reset Vector. (R/O)** 1 = 1 MByte; 0 = 4 GBytes Always 0 on the Pentium M processor.
		15	Reserved.
		17:16	APIC Cluster ID. (R/O) Always 00B on the Pentium M processor.
		18	**System Bus Frequency. (R/O)** 0 = 100 MHz 1 = Reserved Always 0 on the Pentium M processor.
		19	Reserved.
		21: 20	Symmetric Arbitration ID. (R/O) Always 00B on the Pentium M processor.
		26:22	Clock Frequency Ratio (R/O)
40H	64	MSR_LASTBRANCH_0	**Last Branch Record 0. (R/W)** One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the to address. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 18.8, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)"
41H	65	MSR_LASTBRANCH_1	**Last Branch Record 1. (R/W)** See description of MSR_LASTBRANCH_0.

Table B-9. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
42H	66	MSR_LASTBRANCH_2	**Last Branch Record 2. (R/W)** See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	**Last Branch Record 3. (R/W)** See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	**Last Branch Record 4. (R/W)** See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	**Last Branch Record 5. (R/W)** See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	**Last Branch Record 6. (R/W)** See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	**Last Branch Record 7. (R/W)** See description of MSR_LASTBRANCH_0.
119H	281	MSR_BBL_CR_CTL	
		63:0	Reserved.
11EH	281	MSR_BBL_CR_CTL3	
		0	**L2 Hardware Enabled. (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		4:1	Reserved.
		5	**ECC Check Enable. (RO)** This bit enables ECC checking on the cache data bus. ECC is always generated on write cycles. 0 = Disabled (default) 1 = Enabled For the Pentium M processor, ECC checking on the cache data bus is always enabled.
		7:6	Reserved.

Table B-9. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		8	**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9	Reserved.
		23	**L2 Not Present. (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24	Reserved.
179H	377	IA32_MCG_CAP	
		7:0	**Count. (RO)** Indicates the number of hardware unit error reporting banks available in the processor
		8	**IA32_MCG_CTL Present. (RO)** 1 = Indicates that the processor implements the MSR_MCG_CTL register found at MSR 17BH. 0 = Not supported.
		63:9	Reserved.
17AH	378	IA32_MCG_STATUS	
		0	**RIPV.** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted
		1	**EIPV.** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table B-9. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		2	**MCIP.** When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3	Reserved.
198H	408	IA32_PERF_STATUS	**see Table B-2**
199H	409	IA32_PERF_CTL	**see Table B-2**
19AH	410	IA32_CLOCK_ MODULATION	**Clock Modulation. (R/W). see Table B-2.** See Section 13.5.3, "Software Controlled Clock Modulation."
19BH	411	IA32_THERM_ INTERRUPT	**Thermal Interrupt Control. (R/W). see Table B-2.** **See Section 13.5.2, "Thermal Monitor."**
19CH	412	IA32_THERM_ STATUS	**Thermal Monitor Status. (R/W). see Table B-2** See Section 13.5.2, "Thermal Monitor."
19DH	413	MSR_THERM2_CTL	
		15:0	Reserved.
		16	**TM_SELECT.** (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16	Reserved
1A0	416	IA32_MISC_ENABLE	**Enable Miscellaneous Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.
		2:0	Reserved.

Table B-9. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		3	**Automatic Thermal Control Circuit Enable. (R/W)** 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows processor clocks to be automatically modulated based on the processor's thermal sensor operation. 0 = Disabled (default). The automatic thermal control circuit enable bit determines if the thermal control circuit (TCC) will be activated when the processor's internal thermal sensor determines the processor is about to exceed its maximum operating temperature. When the TCC is activated and TM1 is enabled, the processors clocks will be forced to a 50% duty cycle. BIOS must enable this feature. The bit should not be confused with the on-demand thermal control circuit enable bit.
		6:4	Reserved.
		7	**Performance Monitoring Available. (R)** 1 = Performance monitoring enabled 0 = Performance monitoring disabled
		9:8	Reserved.
		10	**FERR# Multiplexing Enable. (R/W)** 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
			Branch Trace Storage Unavailable. (RO) 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported

Table B-9. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		12	**Precise Event Based Sampling Unavailable. (RO)** 1 = Processor does not support precise event-based sampling (PEBS); 0 = PEBS is supported. The Pentium M processor does not support PEBS.
		15:13	Reserved.
		16	**Enhanced Intel SpeedStep Technology Enable. (R/W)** 1 = Enhanced Intel SpeedStep Technology enabled. On the Pentium M processor, this bit may be configured to be read-only.
		22:17	Reserved.
		23	**xTPR Message Disable. (R/W)** When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority. The default is processor specific.
		63:24	Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See also: ▪ MSR_LASTBRANCH_0 (at 40H) ▪ Section 18.8, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)"
1D9H	473	MSR_DEBUGCTLB	**Debug Control. (R/W)** Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 18.8, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."

Table B-9. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
1DDH	477	MSR_LER_TO_LIP	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 18.8, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors)” and Section 18.9.2, “Last Branch and Last Exception MSRs.”
1DEH	478	MSR_LER_FROM_LIP	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 18.8, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors)” and Section 18.9.2, “Last Branch and Last Exception MSRs.”
2FFH	767	IA32_MTRR_DEF_TYPE	**Default Memory Types. (R/W)** Sets the memory type for the regions of physical memory that are not mapped by the MTRRs. See Section 10.11.2.1, “IA32_MTRR_DEF_TYPE MSR.”
400H	1024	IA32_MC0_CTL	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
401H	1025	IA32_MC0_STATUS	See Section 14.3.2.2, “IA32_MCi_STATUS MSRS.”
402H	1026	IA32_MC0_ADDR	See Section 14.3.2.3., “IA32_MCi_ADDR MSRs”. The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
405H	1029	IA32_MC1_STATUS	See Section 14.3.2.2, “IA32_MCi_STATUS MSRS.”

Table B-9. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
406H	1030	IA32_MC1_ADDR	See Section 14.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
409H	1033	IA32_MC2_STATUS	See Chapter 14.3.2.2, “IA32_MCi_STATUS MSRS.”
40AH	1034	IA32_MC2_ADDR	See Section 14.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
40DH	1037	MSR_MC4_STATUS	See Section 14.3.2.2, “IA32_MCi_STATUS MSRS.”
40EH	1038	MSR_MC4_ADDR	See Section 14.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC3_CTL	See Section 14.3.2.1, “IA32_MCi_CTL MSRs.”
411H	1041	MSR_MC3_STATUS	See Section 14.3.2.2, “IA32_MCi_STATUS MSRS.”
412H	1042	MSR_MC3_ADDR	See Section 14.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table B-9. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
600H	1536	IA32_DS_AREA	**DS Save Area. (R/W). see Table B-2** Points to the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 18.16.4, "Debug Store (DS) Mechanism."
		31:0	**DS Buffer Management Area.** Linear address of the first byte of the DS buffer management area.
		63:32	Reserved.

B.7 MSRS IN THE P6 FAMILY PROCESSORS

The following MSRs are defined for the P6 family processors. The MSRs in this table that are shaded are available only in the Pentium II and Pentium III processors. Beginning with the Pentium 4 processor, some of the MSRs in this list have been designated as "architectural" and have had their names changed. See Table B-2 for a list of the architectural MSRs.

Table B-10. MSRs in the P6 Family Processors

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Appendix B.8, "MSRs in Pentium Processors."
1H	1	P5_MC_TYPE	See Appendix B.8, "MSRs in Pentium Processors."
10H	16	TSC	See Section 18.10, "Time-Stamp Counter."
17H	23	IA32_PLATFORM_ID	**Platform ID. (R)** The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
		49:0	Reserved.

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		52:50	**Platform Id. (R)** Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7
		56:53	L2 Cache Latency Read.
		59:57	Reserved.
		60	Clock Frequency Ratio Read.
		63:61	**Reserved.**
1BH	27	APIC_BASE	Section 9.4.4, “Local APIC Status and Location.”
		7:0	Reserved.
		8	**Boot Strap Processor indicator Bit.** 1 = BSP
		10:9	Reserved.
		11	**APIC Global Enable Bit - Permanent till reset.** 1 = Enabled 0 = Disabled
		31:12	APIC Base Address.
		63:32	Reserved.
2AH	42	EBL_CR_POWERON	**Processor Hard Power-On Configuration. (R/W)** Enables and disables processor features; (R) indicates current processor configuration.
		0	Reserved.[1]
		1	**Data Error Checking Enable. (R/W)** 1 = Enabled 0 = Disabled

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		2	**Response Error Checking Enable FRCERR Observation Enable. (R/W)** 1 = Enabled 0 = Disabled
		3	**AERR# Drive Enable. (R/W)** 1 = Enabled 0 = Disabled
		4	**BERR# Enable for Initiator Bus Requests. (R/W)** 1 = Enabled 0 = Disabled
		5	Reserved.
		6	**BERR# Driver Enable for Initiator Internal Errors. (R/W)** 1 = Enabled 0 = Disabled
		7	**BINIT# Driver Enable. (R/W)** 1 = Enabled 0 = Disabled
		8	**Output Tri-state Enabled. (R)** 1 = Enabled 0 = Disabled
		9	**Execute BIST. (R)** 1 = Enabled 0 = Disabled
		10	**AERR# Observation Enabled. (R)** 1 = Enabled 0 = Disabled
		11	Reserved.
		12	**BINIT# Observation Enabled. (R)** 1 = Enabled 0 = Disabled

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		13	**In Order Queue Depth. (R)** 1 = 1 0 = 8
		14	**1-MByte Power on Reset Vector. (R)** 1 = 1MByte 0 = 4GBytes
		15	**FRC Mode Enable. (R)** 1 = Enabled 0 = Disabled
		17:16	**APIC Cluster ID. (R)**
		19:18	**System Bus Frequency. (R)** 00 = 66MHz 10 = 100Mhz 01 = 133MHz 11 = Reserved
		21: 20	**Symmetric Arbitration ID. (R)**
		25:22	**Clock Frequency Ratio. (R)**
		26	**Low Power Mode Enable. (R/W)**
		27	**Clock Frequency Ratio.**
		63:28	Reserved.[1]
33H	51	TEST_CTL	**Test Control Register.**
		29:0	Reserved.
		30	**Streaming Buffer Disable.**
		31	**Disable LOCK#.** Assertion for split locked access.
79H	121	BIOS_UPDT_TRIG	BIOS Update Trigger Register.
88	136	BBL_CR_D0[63:0]	Chunk 0 data register D[63:0]: used to write to and read from the L2
89	137	BBL_CR_D1[63:0]	Chunk 1 data register D[63:0]: used to write to and read from the L2
8A	138	BBL_CR_D2[63:0]	Chunk 2 data register D[63:0]: used to write to and read from the L2

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
8BH	139	BIOS_SIGN/BBL_CR_D3[63:0]	**BIOS Update Signature Register or Chunk 3 data register D[63:0].** Used to write to and read from the L2 depending on the usage model
C1H	193	PerfCtr0 (PERFCTR0)	
C2H	194	PerfCtr1 (PERFCTR1)	
FEH	254	MTRRcap	
116	278	BBL_CR_ADDR [63:0]	Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses.
		BBL_CR_ADDR [63:32]	Reserved,
		BBL_CR_ADDR [31:3]	Address bits [35:3]
		BBL_CR_ADDR [2:0]	Reserved Set to 0.
118	280	BBL_CR_DECC[63:0]	Data ECC register D[7:0]: used to write ECC and read ECC to/from L2
119	281	BBL_CR_CTL	Control register: used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response
		BL_CR_CTL[63:22]	Reserved
		BBL_CR_CTL[21]	Processor number[2]

			Disable = 1
			Enable = 0
			Reserved

		BBL_CR_CTL[20:19]	User supplied ECC
		BBL_CR_CTL[18]	Reserved
		BBL_CR_CTL[17]	L2 Hit
		BBL_CR_CTL[16]	Reserved
		BBL_CR_CTL[15:14]	State from L2
		BBL_CR_CTL[13:12]	Modified - 11,Exclusive - 10, Shared - 01, Invalid - 00
		BBL_CR_CTL[11:10]	Way from L2
			Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11
		BBL_CR_CTL[9:8]	Way to L2
		BBL_CR_CTL[7]	Reserved
		BBL_CR_CTL[6:5]	State to L2

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		BBL_CR_CTL[4:0] 01100 01110 01111 00010 00011 010 + MESI encode 111 + MESI encode 100 + MESI encode	L2 Command Data Read w/ LRU update (RLU) Tag Read w/ Data Read (TRR) Tag Inquire (TI) L2 Control Register Read (CR) L2 Control Register Write (CW) Tag Write w/ Data Read (TWR) Tag Write w/ Data Write (TWW) Tag Write (TW)
11A	282	BBL_CR_TRIG	Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.
11B	283	BBL_CR_BUSY	Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY
11E	286	BBL_CR_CTL3	Control register 3: used to configure the L2 Cache
		BBL_CR_CTL3[63:26]	Reserved
		BBL_CR_CTL3[25]	Cache bus fraction (read only)
		BBL_CR_CTL3[24]	Reserved
		BBL_CR_CTL3[23]	L2 Hardware Disable (read only)
		BBL_CR_CTL3[22:20] 111 110 101 100 011 010 001 000	L2 Physical Address Range support 64GBytes 32GBytes 16GBytes 8GBytes 4GBytes 2GBytes 1GBytes 512MBytes
		BBL_CR_CTL3[19]	Reserved
		BBL_CR_CTL3[18]	Cache State error checking enable (read/write)

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		BBL_CR_CTL3[17:13	Cache size per bank (read/write)
		00001	256KBytes
		00010	512KBytes
		00100	1MByte
		01000	2MByte
		10000	4MBytes
		BBL_CR_CTL3[12:11]	Number of L2 banks (read only)
		BBL_CR_CTL3[10:9]	L2 Associativity (read only)
		00	Direct Mapped
		01	2 Way
		10	4 Way
		11	Reserved
		BBL_CR_CTL3[8]	L2 Enabled (read/write)
		BBL_CR_CTL3[7]	CRTN Parity Check Enable (read/write)
		BBL_CR_CTL3[6]	Address Parity Check Enable (read/write)
		BBL_CR_CTL3[5]	ECC Check Enable (read/write)
		BBL_CR_CTL3[4:1]	L2 Cache Latency (read/write)
		BBL_CR_CTL3[0]	L2 Configured (read/write)
174H	372	SYSENTER_CS_MSR	CS register target for CPL 0 code
175H	373	SYSENTER_ESP_MSR	Stack pointer for CPL 0 stack
176H	374	SYSENTER_EIP_MSR	CPL 0 code entry point
179H	377	MCG_CAP	
17AH	378	MCG_STATUS	
17BH	379	MCG_CTL	
186H	390	PerfEvtSel0 (EVNTSEL0)	
		7:0	**Event Select.** Refer to Performance Counter section for a list of event encodings.
		15:8	**UMASK (Unit Mask).** Unit mask register set to 0 to enable all count options.

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		16	**USER.** Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	**OS.** Controls the counting of events at Privilege level of 0.
		18	**E.** Occurrence/Duration Mode Select 1 = Occurrence 0 = Duration
		19	**PC.** Enabled the signaling of performance counter overflow via BP0 pin
		20	**INT.** Enables the signaling of counter overflow via input to APIC 1 = Enable 0 = Disable
		22	**ENABLE.** Enables the counting of performance events in both counters 1 = Enable 0 = Disable
		23	**INV.** Inverts the result of the CMASK condition 1 = Inverted 0 = Non-Inverted
		31:24	CMASK (Counter Mask).

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
187H	391	PerfEvtSel1 (EVNTSEL1)	
		7:0	**Event Select.** Refer to Performance Counter section for a list of event encodings.
		15:8	**UMASK (Unit Mask).** Unit mask register set to 0 to enable all count options.
		16	**USER.** Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	**OS.** Controls the counting of events at Privilege level of 0
		18	**E.** Occurrence/Duration Mode Select 1 = Occurrence 0 = Duration
		19	**PC.** Enabled the signaling of performance counter overflow via BP0 pin.
		20	**INT.** Enables the signaling of counter overflow via input to APIC 1 = Enable 0 = Disable
		23	**INV.** Inverts the result of the CMASK condition 1 = Inverted 0 = Non-Inverted
		31:24	**CMASK (Counter Mask).**
1D9H	473	DEBUGCTLMSR	
		0	Enable/Disable Last Branch Records
		1	Branch Trap Flag

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		2	Performance Monitoring/Break Point Pins
		3	Performance Monitoring/Break Point Pins
		4	Performance Monitoring/Break Point Pins
		5	Performance Monitoring/Break Point Pins
		6	Enable/Disable Execution Trace Messages
		31:7	Reserved
1DBH	475	LASTBRANCHFROMIP	
1DCH	476	LASTBRANCHTOIP	
1DDH	477	LASTINTFROMIP	
1DEH	478	LASTINTTOIP	
1E0H	480	ROB_CR_BKUPTMPDR6	
		1:0	Reserved
		2	Fast String Enable bit. Default is enabled
200H	512	MTRRphysBase0	
201H	513	MTRRphysMask0	
202H	514	MTRRphysBase1	
203H	515	MTRRphysMask1	
204H	516	MTRRphysBase2	
205H	517	MTRRphysMask2	
206H	518	MTRRphysBase3	
207H	519	MTRRphysMask3	
208H	520	MTRRphysBase4	
209H	521	MTRRphysMask4	
20AH	522	MTRRphysBase5	
20BH	523	MTRRphysMask5	
20CH	524	MTRRphysBase6	
20DH	525	MTRRphysMask6	
20EH	526	MTRRphysBase7	
20FH	527	MTRRphysMask7	

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
250H	592	MTRRfix64K_00000	
258H	600	MTRRfix16K_80000	
259H	601	MTRRfix16K_A0000	
268H	616	MTRRfix4K_C0000	
269H	617	MTRRfix4K_C8000	
26AH	618	MTRRfix4K_D0000	
26BH	619	MTRRfix4K_D8000	
26CH	620	MTRRfix4K_E0000	
26DH	621	MTRRfix4K_E8000	
26EH	622	MTRRfix4K_F0000	
26FH	623	MTRRfix4K_F8000	
2FFH	767	MTRRdefType	
		2:0	Default memory type
		10	Fixed MTRR enable
		11	MTRR Enable
400H	1024	MC0_CTL	
401H	1025	MC0_STATUS	
		63	MC_STATUS_V
		62	MC_STATUS_O
		61	MC_STATUS_UC
		60	MC_STATUS_EN. (Note: For MC0_STATUS only, this bit is hardcoded to 1.)
		59	MC_STATUS_MISCV
		58	MC_STATUS_ADDRV
		57	MC_STATUS_DAM
		31:16	MC_STATUS_MCACOD
		15:0	MC_STATUS_MSCOD
402H	1026	MC0_ADDR	
403H	1027	MC0_MISC	Defined in MCA architecture but not implemented in the P6 family processors

Table B-10. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
404H	1028	MC1_CTL	
405H	1029	MC1_STATUS	Bit definitions same as MC0_STATUS
406H	1030	MC1_ADDR	
407H	1031	MC1_MISC	Defined in MCA architecture but not implemented in the P6 family processors
408H	1032	MC2_CTL	
409H	1033	MC2_STATUS	Bit definitions same as MC0_STATUS
40AH	1034	MC2_ADDR	
40BH	1035	MC2_MISC	Defined in MCA architecture but not implemented in the P6 family processors
40CH	1036	MC4_CTL	
40DH	1037	MC4_STATUS	Bit definitions same as MC0_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.
40EH	1038	MC4_ADDR	Defined in MCA architecture but not implemented in P6 Family processors
40FH	1039	MC4_MISC	Defined in MCA architecture but not implemented in the P6 family processors
410H	1040	MC3_CTL	
411H	1041	MC3_STATUS	Bit definitions same as MC0_STATUS
412H	1042	MC3_ADDR	
413H	1043	MC3_MISC	Defined in MCA architecture but not implemented in the P6 family processors

NOTES

1. Bit 0 of this register has been redefined several times, and is no longer used in P6 family processors.
2. The processor number feature may be disabled by setting bit 21 of the BBL_CR_CTL MSR (model-specific register address 119h) to “1”. Once set, bit 21 of the BBL_CR_CTL may not be cleared. This bit is write-once. The processor number feature will be disabled until the processor is reset.
3. The Pentium III processor will prevent FSB frequency overclocking with a new shutdown mechanism. If the FSB frequency selected is greater than the internal FSB frequency the processor will shutdown. If the FSB selected is less than the internal FSB frequency the BIOS may choose to use bit 11 to implement its own shutdown policy.

B.8 MSRS IN PENTIUM PROCESSORS

The following MSRs are defined for the Pentium processors. The P5_MC_ADDR, P5_MC_TYPE, and TSC MSRs (named IA32_P5_MC_ADDR, IA32_P5_MC_TYPE, and IA32_TIME_STAMP_COUNTER in the Pentium 4 processor) are architectural; that is, code that accesses these registers will run on Pentium 4 and P6 family processors without generating exceptions (see Section B.1, “Architectural MSRs”). The CESR, CTR0, and CTR1 MSRs are unique to Pentium processors; code that accesses these registers will generate exceptions on Pentium 4 and P6 family processors.

Table B-11. MSRs in the Pentium Processor

Register Address			
Hex	Dec	Register Name	Bit Description
0H	0	P5_MC_ADDR	See Section 14.8.3, “Pentium Processor Machine-Check Exception Handling.”
1H	1	P5_MC_TYPE	See Section 14.8.3, “Pentium Processor Machine-Check Exception Handling.”
10H	16	TSC	See Section 18.10, “Time-Stamp Counter.”
11H	17	CESR	See Section 18.24.1, “Control and Event Select Register (CESR).”
12H	18	CTR0	Section 18.24.3, “Events Counted.”
13H	19	CTR1	Section 18.24.3, “Events Counted.”

APPENDIX C
MP INITIALIZATION FOR P6 FAMILY PROCESSORS

This appendix describes the MP initialization process for systems that use multiple P6 family processors. This process uses the MP initialization protocol that was introduced with the Pentium Pro processor (see Section 7.5, “Multiple-Processor (MP) Initialization”). For P6 family processors, this protocol is typically used to boot 2 or 4 processors that reside on single system bus; however, it can support from 2 to 15 processors in a multi-clustered system when the APIC busses are tied together. Larger systems are not supported.

C.1 OVERVIEW OF THE MP INITIALIZATION PROCESS FOR P6 FAMILY PROCESSORS

During the execution of the MP initialization protocol, one processor is selected as the bootstrap processor (BSP) and the remaining processors are designated as application processors (APs), see Section 7.5.1, “BSP and AP Processors.” Thereafter, the BSP manages the initialization of itself and the APs. This initialization includes executing BIOS initialization code and operating-system initialization code.

The MP protocol imposes the following requirements and restrictions on the system:

- An APIC clock (APICLK) must be provided.
- The MP protocol will be executed only after a power-up or RESET. If the MP protocol has been completed and a BSP has been chosen, subsequent INITs (either to a specific processor or system wide) do not cause the MP protocol to be repeated. Instead, each processor examines its BSP flag (in the APIC_BASE MSR) to determine whether it should execute the BIOS boot-strap code (if it is the BSP) or enter a wait-for-SIPI state (if it is an AP).
- All devices in the system that are capable of delivering interrupts to the processors must be inhibited from doing so for the duration of the MP initialization protocol. The time during which interrupts must be inhibited includes the window between when the BSP issues an INIT-SIPI-SIPI sequence to an AP and when the AP responds to the last SIPI in the sequence.

The following special-purpose interprocessor interrupts (IPIs) are used during the boot phase of the MP initialization protocol. These IPIs are broadcast on the APIC bus.

- Boot IPI (BIPI)—Initiates the arbitration mechanism that selects a BSP from the group of processors on the system bus and designates the remainder of the processors as APs. Each processor on the system bus broadcasts a BIPI to all the processors following a power-up or RESET.

- Final Boot IPI (FIPI)—Initiates the BIOS initialization procedure for the BSP. This IPI is broadcast to all the processors on the system bus, but only the BSP responds to it. The BSP responds by beginning execution of the BIOS initialization code at the reset vector.
- Startup IPI (SIPI)—Initiates the initialization procedure for an AP. The SIPI message contains a vector to the AP initialization code in the BIOS.

Table C-1 describes the various fields of the boot phase IPIs.

Table C-1. Boot Phase IPI Message Format

Type	Destination Field	Destination Shorthand	Trigger Mode	Level	Destination Mode	Delivery Mode	Vector (Hex)
BIPI	Not used	All including self	Edge	Deassert	Don't Care	Fixed (000)	40 to 4E*
FIPI	Not used	All including self	Edge	Deassert	Don't Care	Fixed (000)	10
SIPI	Used	All excluding self	Edge	Assert	Physical	StartUp (110)	00 to FF

NOTE:

* For all P6 family processors.

For BIPI messages, the lower 4 bits of the vector field contain the APIC ID of the processor issuing the message and the upper 4 bits contain the "generation ID" of the message. All P6 family processor will have a generation ID of 4H. BIPIs will therefore use vector values ranging from 40H to 4EH (4FH can not be used because FH is not a valid APIC ID).

C.2 MP INITIALIZATION PROTOCOL ALGORITHM

Following a power-up or RESET of a system, the P6 family processors in the system execute the MP initialization protocol algorithm to initialize each of the processors on the system bus. In the course of executing this algorithm, the following boot-up and initialization operations are carried out:

1. Each processor on the system bus is assigned a unique APIC ID, based on system topology (see Section 7.5.5, "Identifying Logical Processors in an MP System"). This ID is written into the local APIC ID register for each processor.
2. Each processor executes its internal BIST simultaneously with the other processors on the system bus. Upon completion of the BIST (at T0), each processor broadcasts a BIPI to "all including self" (see Figure 1).
3. APIC arbitration hardware causes all the APICs to respond to the BIPIs one at a time (at T1, T2, T3, and T4).
4. When the first BIPI is received (at time T1), each APIC compares the four least significant bits of the BIPI's vector field with its APIC ID. If the vector and APIC ID match, the processor selects itself as the BSP by setting the BSP flag in its

IA32_APIC_BASE MSR. If the vector and APIC ID do not match, the processor selects itself as an AP by entering the “wait for SIPI” state. (Note that in Figure 1, the BIPI from processor 1 is the first BIPI to be handled, so processor 1 becomes the BSP.)

5. The newly established BSP broadcasts an FIPI message to “all including self.” The FIPI is guaranteed to be handled only after the completion of the BIPIs that were issued by the non-BSP processors.

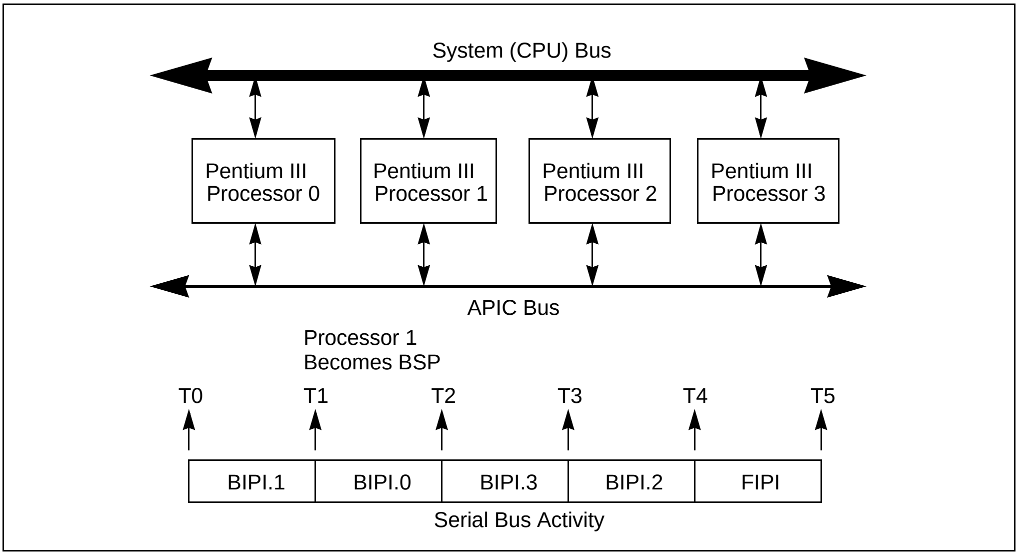

Figure C-1. MP System With Multiple Pentium III Processors

6. After the BSP has been established, the outstanding BIPIs are received one at a time (at T2, T3, and T4) and ignored by all processors.
7. When the FIPI is finally received (at T5), only the BSP responds to it. It responds by fetching and executing BIOS boot-strap code, beginning at the reset vector (physical address FFFF FFF0H).
8. As part of the boot-strap code, the BSP creates an ACPI table and an MP table and adds its initial APIC ID to these tables as appropriate.
9. At the end of the boot-strap procedure, the BSP broadcasts a SIPI message to all the APs in the system. Here, the SIPI message contains a vector to the BIOS AP initialization code (at 000V V000H, where VV is the vector contained in the SIPI message).
10. All APs respond to the SIPI message by racing to a BIOS initialization semaphore. The first one to the semaphore begins executing the initialization code. (See MP init code for semaphore implementation details.) As part of the AP initialization procedure, the AP adds its APIC ID number to the ACPI and MP tables as appro-

priate. At the completion of the initialization procedure, the AP executes a CLI instruction (to clear the IF flag in the EFLAGS register) and halts itself.

11. When each of the APs has gained access to the semaphore and executed the AP initialization code and all written their APIC IDs into the appropriate places in the ACPI and MP tables, the BSP establishes a count for the number of processors connected to the system bus, completes executing the BIOS boot-strap code, and then begins executing operating-system boot-strap and start-up code.
12. While the BSP is executing operating-system boot-strap and start-up code, the APs remain in the halted state. In this state they will respond only to INITs, NMIs, and SMIs. They will also respond to snoops and to assertions of the STPCLK# pin.

See Section 7.5.4, “MP Initialization Example,” for an annotated example the use of the MP protocol to boot IA-32 processors in an MP. This code should run on any IA-32 processor that used the MP protocol.

C.2.1 Error Detection and Handling During the MP Initialization Protocol

Errors may occur on the APIC bus during the MP initialization phase. These errors may be transient or permanent and can be caused by a variety of failure mechanisms (for example, broken traces, soft errors during bus usage, etc.). All serial bus related errors will result in an APIC checksum or acceptance error.

The MP initialization protocol makes the following assumptions regarding errors that occur during initialization:

- If errors are detected on the APIC bus during execution of the MP initialization protocol, the processors that detect the errors are shut down.
- The MP initialization protocol will be executed by processors even if they fail their BIST sequences.

APPENDIX D
PROGRAMMING THE LINT0 AND LINT1 INPUTS

The following procedure describes how to program the LINT0 and LINT1 local APIC pins on a processor after multiple processors have been booted and initialized (as described in Appendix C, “MP Initialization For P6 Family Processors,” and Appendix D, “Programming the LINT0 and LINT1 Inputs.” In this example, LINT0 is programmed to be the ExtINT pin and LINT1 is programmed to be the NMI pin.

D.1 CONSTANTS

The following constants are defined:

```
LVT1EQU 0FEE00350H
LVT2EQU 0FEE00360H
LVT3 EQU 0FEE00370H
SVR   EQU 0FEE000F0H
```

D.2 LINT[0:1] PINS PROGRAMMING PROCEDURE

Use the following to program the LINT[1:0] pins:

1. Mask 8259 interrupts.
2. Enable APIC via SVR (spurious vector register) if not already enabled.

```
MOV ESI, SVR                ; address of SVR
MOV EAX, [ESI]
OR  EAX, APIC_ENABLED       ; set bit 8 to enable (0 on reset)
MOV [ESI], EAX
```

3. Program LVT1 as an ExtINT which delivers the signal to the INTR signal of all processors cores listed in the destination as an interrupt that originated in an externally connected interrupt controller.

```
MOV ESI, LVT1
MOV EAX, [ESI]
AND EAX, 0FFFE58FFH; mask off bits 8-10, 12, 14 and 16
OR  EAX, 700H; Bit 16=0 for not masked, Bit 15=0 for edge
     ; triggered, Bit 13=0 for high active input
     ; polarity, Bits 8-10 are 111b for ExtINT
MOV [ESI], EAX; Write to LVT1
```

4. Program LVT2 as NMI, which delivers the signal on the NMI signal of all processor cores listed in the destination.

```
MOV ESI, LVT2
MOV EAX, [ESI]
AND EAX, 0FFFE58FFH; mask off bits 8-10 and 15
OR  EAX, 000000400H; Bit 16=0 for not masked, Bit 15=0 edge
        ; triggered, Bit 13=0 for high active input
        ; polarity, Bits 8-10 are 100b for NMI
MOV [ESI], EAX; Write to LVT2
        ;Unmask 8259 interrupts and allow NMI.
```

APPENDIX E
INTERPRETING MACHINE-CHECK ERROR CODES

Encoding of the model-specific and other information fields is different for 06H and 0FH processor families. The differences are documented in the following sections.

E.1 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY 06H MACHINE ERROR CODES FOR MACHINE CHECK

Table E.1 provides information for interpreting additional family 06H model-specific fields for external bus errors. These errors are reported in the IA32_MCi_STATUS MSRs. They are reported architecturally) as compound errors with a general form of *0000 1PPT RRRR IILL* in the MCA error code field. See Chapter 14 for information on the interpretation of compound error codes.

Table E-1. Incremental Decoding Information: Processor Family 06H Machine Error Codes For Machine Check

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	0-15		
Model specific errors	16-18	Reserved	Reserved
Model specific errors	19-24	Bus queue request type	000000 for BQ_DCU_READ_TYPE error 000010 for BQ_IFU_DEMAND_TYPE error 000011 for BQ_IFU_DEMAND_NC_TYPE error 000100 for BQ_DCU_RFO_TYPE error 000101 for BQ_DCU_RFO_LOCK_TYPE error 000110 for BQ_DCU_ITOM_TYPE error 001000 for BQ_DCU_WB_TYPE error 001010 for BQ_DCU_WCEVICT_TYPE error 001011 for BQ_DCU_WCLINE_TYPE error 001100 for BQ_DCU_BTM_TYPE error

Table E-1. Incremental Decoding Information: Processor Family 06H Machine Error Codes For Machine Check (Contd.)

Type	Bit No.	Bit Function	Bit Description
			001101 for BQ_DCU_INTACK_TYPE error 001110 for BQ_DCU_INVALL2_TYPE error 001111 for BQ_DCU_FLUSHL2_TYPE error 010000 for BQ_DCU_PART_RD_TYPE error 010010 for BQ_DCU_PART_WR_TYPE error 010100 for BQ_DCU_SPEC_CYC_TYPE error 011000 for BQ_DCU_IO_RD_TYPE error 011001 for BQ_DCU_IO_WR_TYPE error 011100 for BQ_DCU_LOCK_RD_TYPE error 011110 for BQ_DCU_SPLOCK_RD_TYPE error 011101 for BQ_DCU_LOCK_WR_TYPE error
Model specific errors	27-25	Bus queue error type	000 for BQ_ERR_HARD_TYPE error 001 for BQ_ERR_DOUBLE_TYPE error 010 for BQ_ERR_AERR2_TYPE error 100 for BQ_ERR_SINGLE_TYPE error 101 for BQ_ERR_AERR1_TYPE error
Model specific errors	28	FRC error	1 if FRC error active
	29	BERR	1 if BERR is driven
	30	Internal BINIT	1 if BINIT driven for this processor
	31	Reserved	Reserved
Other information	32-34	Reserved	Reserved
	35	External BINIT	1 if BINIT is received from external bus.
	36	Response parity error	This bit is asserted in IA32_MC*i*_STATUS if this component has received a parity error on the RS[2:0]# pins for a response transaction. The RS signals are checked by the RSP# external pin.
	37	Bus BINIT	This bit is asserted in IA32_MC*i*_STATUS if this component has received a hard error response on a split transaction one access that has needed to be split across the 64-bit external bus interface into two accesses).

Table E-1. Incremental Decoding Information: Processor Family 06H Machine Error Codes For Machine Check (Contd.)

Type	Bit No.	Bit Function	Bit Description
	38	Timeout BINIT	This bit is asserted in IA32_MC*i*_STATUS if this component has experienced a ROB time-out, which indicates that no micro-instruction has been retired for a predetermined period of time. A ROB time-out occurs when the 15-bit ROB time-out counter carries a 1 out of its high order bit. The timer is cleared when a micro-instruction retires, an exception is detected by the core processor, RESET is asserted, or when a ROB BINIT occurs. The ROB time-out counter is prescaled by the 8-bit PIC timer which is a divide by 128 of the bus clock the bus clock is 1:2, 1:3, 1:4 of the core clock). When a carry out of the 8-bit PIC timer occurs, the ROB counter counts up by one. While this bit is asserted, it cannot be overwritten by another error.
	39-41	Reserved	Reserved
	42	Hard error	This bit is asserted in IA32_MC*i*_STATUS if this component has initiated a bus transactions which has received a hard error response. While this bit is asserted, it cannot be overwritten.
	43	IERR	This bit is asserted in IA32_MC*i*_STATUS if this component has experienced a failure that causes the IERR pin to be asserted. While this bit is asserted, it cannot be overwritten.

Table E-1. Incremental Decoding Information: Processor Family 06H Machine Error Codes For Machine Check (Contd.)

Type	Bit No.	Bit Function	Bit Description
	44	AERR	This bit is asserted in IA32_MC*i*_STATUS if this component has initiated 2 failing bus transactions which have failed due to Address Parity Errors AERR asserted). While this bit is asserted, it cannot be overwritten.
	45	UECC	The Uncorrectable ECC error bit is asserted in IA32_MC*i*_STATUS for uncorrected ECC errors. While this bit is asserted, the ECC syndrome field will not be overwritten.
	46	CECC	The correctable ECC error bit is asserted in IA32_MC*i*_STATUS for corrected ECC errors.
	47-54	ECC syndrome	The ECC syndrome field in IA32_MCi_STATUS contains the 8-bit ECC syndrome only if the error was a correctable/uncorrectable ECC error and there wasn't a previous valid ECC error syndrome logged in IA32_MCi_STATUS. A previous valid ECC error in IA32_MCi_STATUS is indicated by IA32_MCi_STATUS.bit45 uncorrectable error occurred) being asserted. After processing an ECC error, machine-check handling software should clear IA32_MCi_STATUS.bit45 so that future ECC error syndromes can be logged.
	55-56	Reserved	Reserved.
Status register validity indicators[1]	57-63		

NOTES:

[1] These fields are architecturally defined. Refer to Chapter 14, "Machine-Check Architecture," for more information.

E.2 INCREMENTAL DECODING INFORMATION: PROCESSOR FAMILY 0FH MACHINE ERROR CODES FOR MACHINE CHECK

Table E-2 provides information for interpreting additional family 0FH model-specific fields for external bus errors. These errors are reported in the IA32_MCi_STATUS

MSRs. They are reported architecturally) as compound errors with a general form of *0000 1PPT RRRR IILL* in the MCA error code field. See Chapter 14 for information on the interpretation of compound error codes.

Table E-2. Incremental Decoding Information: Processor Family 0FH Machine Error Codes For Machine Check

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	0-15		
Model-specific error codes	16	FSB address parity	Address parity error detected: 1 = Address parity error detected 0 = No address parity error
	17	Response hard fail	Hardware failure detected on response
	18	Response parity	Parity error detected on response
	19	PIC and FSB data parity	Data Parity detected on either PIC or FSB access
	20	Processor Signature = 00000F04H: Invalid PIC request All other processors: Reserved	Processor Signature = 00000F04H. Indicates error due to an invalid PIC request access was made to PIC space with WB memory): 1 = Invalid PIC request error 0 = No Invalid PIC request error Reserved
	21	Pad state machine	The state machine that tracks P and N data-strobe relative timing has become unsynchronized or a glitch has been detected.
	22	Pad strobe glitch	Data strobe glitch
Type	**Bit No.**	**Bit Function**	**Bit Description**
	23	Pad address glitch	Address strobe glitch
Other Information	24-56	Reserved	Reserved
Status register validity indicators[1]	57-63		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 14, “Machine-Check Architecture,” for more information.

Table E-9 provides information on interpreting additional family 0FH, model specific fields for memory hierarchy errors. These errors are reported in one of the IA32_MCi_STATUS MSRs. These errors are reported, architecturally, as compound errors with a general form of *0000 0001 RRRR TTLL* in the MCA error code field. See Chapter 14 for how to interpret the compound error code.

E.2.1 Model-Specific Machine Check Error Codes for Intel Xeon Processor MP 7100 Series

Intel Xeon processor MP 7100 series has 5 register banks which contains information related to Machine Check Errors. MCi_STATUS[63:0] refers to all 5 register banks. MC0_STATUS[63:0] through MC3_STATUS[63:0] is the same as on previous generation of Intel Xeon processors within Family 0FH. MC4_STATUS[63:0] is the main error logging for the processor’s L3 and front side bus errors. It supports the L3 Errors, Bus and Interconnect Errors Compound Error Codes in the MCA Error Code Field.

Table E-3. MCi_STATUS Register Bit Definition

Bit Field Name	Bits	Description
MCA_Error_Code	15:0	Specifies the machine check architecture defined error code for the machine check error condition detected. The machine check architecture defined error codes are guaranteed to be the same for all Intel Architecture processors that implement the machine check architecture. See tables below
Model_Specific_Error_Code	31:16	Specifies the model specific error code that uniquely identifies the machine check error condition detected. The model specific error codes may differ among Intel Architecture processors for the same Machine Check Error condition. See tables below
Other_Info	56:32	The functions of the bits in this field are implementation specific and are not part of the machine check architecture. Software that is intended to be portable among Intel Architecture processors should not rely on the values in this field.
PCC	57	Processor Context Corrupt flag indicates that the state of the processor might have been corrupted by the error condition detected and that reliable restarting of the processor may not be possible. When clear, this flag indicates that the error did not affect the processor's state. This bit will always be set for MC errors which are not corrected.
ADDRV	58	MC_ADDR register valid flag indicates that the MC_ADDR register contains the address where the error occurred. When clear, this flag indicates that the MC_ADDR register does not contain the address where the error occurred. The MC_ADDR register should not be read if the ADDRV bit is clear.
MISCV	59	MC_MISC register valid flag indicates that the MC_MISC register contains additional information regarding the error. When clear, this flag indicates that the MC_MISC register does not contain additional information regarding the error. MC_MISC should not be read if the MISCV bit is not set.

Table E-3. MCi_STATUS Register Bit Definition

EN	60	Error enabled flag indicates that reporting of the machine check exception for this error was enabled by the associated flag bit of the MC_CTL register. Note that correctable errors do not have associated enable bits in the MC_CTL register so the EN bit should be clear when a correctable error is logged.
UC	61	Error uncorrected flag indicates that the processor did not correct the error condition. When clear, this flag indicates that the processor was able to correct the event condition.
OVER	62	Machine check overflow flag indicates that a machine check error occurred while the results of a previous error were still in the register bank (i.e., the VAL bit was already set in the MC_STATUS register). The processor sets the OVER flag and software is responsible for clearing it. Enabled errors are written over disabled errors, and uncorrected errors are written over corrected events. Uncorrected errors are not written over previous valid uncorrected errors.
VAL	63	MC_STATUS register valid flag indicates that the information within the MC_STATUS register is valid. When this flag is set, the processor follows the rules given for the OVER flag in the MC_STATUS register when overwriting previously valid entries. The processor sets the VAL flag and software is responsible for clearing it.

E.2.1.1 Processor Machine Check Status Register MCA Error Code Definition

Intel Xeon processor MP 7100 series use compound MCA Error Codes for logging its CBC internal machine check errors, L3 Errors, and Bus/Interconnect Errors. It defines additional Machine Check error types (IA32_MC4_STATUS[15:0]) beyond those defined in Chapter 14. Table E-4 lists these model-specific MCA error codes. Error code details are specified in MC4_STATUS [31:16] (see Section E.2.3), the "Model Specific Error Code" field. The information in the "Other_Info" field (MC4_STATUS[56:32]) is common to the three processor error types and contains a correctable event count and specifies the MC4_MISC register format.

Table E-4. Model-Specific MCA Error Code(Sheet 1 of 2)

Processor MCA_Error_Code (MC4_STATUS[15:0])			
Type	**Error Code**	**Binary Encoding**	**Meaning**
C	**Internal Error**	**0000 0100 0000 0000**	**Internal Error Type Code**
A	**L3 Tag Error**	**0000 0001 0000 1011**	**L3 Tag Error Type Code**

Table E-4. Model-Specific MCA Error Code(Sheet 2 of 2)

Processor MCA_Error_Code (MC4_STATUS[15:0])			
B	**Bus and Interconnect Error**	0000 100x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations
		0000 101x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations
		0000 110x 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations
		0000 1110 0000 1111	**Bus and Interconnection Error Type Code**
		0000 1111 0000 1111	Not used but this encoding is reserved for compatibility with other MCA implementations

The **Bold faced** binary encodings are the only encodings used by the processor for MC4_STATUS[15:0].

5.2.2 Other_Info Field (all MCA Error Types)

The MC4_STATUS[56:32] field is common to the processor's three MCA error types (A, B & C):

Table E-5. Other Information Field Bit Definition

Bit Field Name	Bits	Description
39:32	8-bit Correctable Event Count	Holds a count of the number of correctable events since cold reset. This is a saturating counter; the counter begins at 1 (with the first error) and saturates at a count of 255.
41:40	MC4_MISC format type	The value in this field specifies the format of information in the MC4_MISC register. Currently, only two values are defined. Valid only when MISCV is asserted.
43:42	–	Reserved
51:44	ECC syndrome	ECC syndrome value for a correctable ECC event when the “Valid ECC syndrome” bit is asserted

Table E-5. Other Information Field Bit Definition

52	Valid ECC syndrome	Set when correctable ECC event supplies the ECC syndrome
54:53	Threshold-Based Error Status	00: No tracking - No hardware status tracking is provided for the structure reporting this event. 01: Green - Status tracking is provided for the structure posting the event; the current status is green (below threshold). 10: Yellow - Status tracking is provided for the structure posting the event; the current status is yellow (above threshold). 11: Reserved for future use Valid only if Valid bit (bit 63) is set Undefined if the UC bit (bit 61) is set
56:55	–	Reserved

E.2.3 Processor Model Specific Error Code Field

E.2.3.1 MCA Error Type A: L3 Error

Note: The Model Specific Error Code field in MC4_STATUS (bits 31:16)

Table E-6. Type A: L3 Error Codes

Bit Num	Sub-Field Name	Description	Legal Value(s)
18:16	L3 Error Code	Describes the L3 error encountered	000 - No error 001 - More than one way reporting a correctable event 010 - More than one way reporting an uncorrectable error 011 - More than one way reporting a tag hit 100 - No error 101 - One way reporting a correctable event 110 - One way reporting an uncorrectable error 111 - One or more ways reporting a correctable event while one or more ways are reporting an uncorrectable error
20:19	–	Reserved	00
31:21	–	Fixed pattern	0010_0000_000

E.2.3.2 Processor Model Specific Error Code Field Type B: Bus and Interconnect Error

Note: The Model Specific Error Code field in MC4_STATUS (bits 31:16)

Table E-7. Type B Bus and Interconnect Error Codes

Bit Num	Sub-Field Name	Description
16	FSB Request Parity	Parity error detected during FSB request phase
17	Core0 Addr Parity	Parity error detected on Core 0 request’s address field
18	Core1 Addr Parity	Parity error detected on Core 1 request’s address field
19		Reserved
20	FSB Response Parity	Parity error on FSB response field detected
21	FSB Data Parity	FSB data parity error on inbound data detected
22	Core0 Data Parity	Data parity error on data received from Core 0 detected
23	Core1 Data Parity	Data parity error on data received from Core 1 detected
24	IDS Parity	Detected an Enhanced Defer parity error (phase A or phase B)
25	FSB Inbound Data ECC	Data ECC event to error on inbound data (correctable or uncorrectable)
26	FSB Data Glitch	Pad logic detected a data strobe ‘glitch’ (or sequencing error)
27	FSB Address Glitch	Pad logic detected a request strobe ‘glitch’ (or sequencing error)
31:28	---	Reserved

Exactly one of the bits defined in the preceding table will be set for a Bus and Interconnect Error. The Data ECC can be correctable or uncorrectable (the MC4_STATUS.UC bit, of course, distinguishes between correctable and uncorrectable cases with the Other_Info field possibly providing the ECC Syndrome for correctable errors). All other errors for this processor MCA Error Type are uncorrectable.

5.2.3.3 Processor Model Specific Error Code Field Type C: Cache Bus Controller Error

Table E-8. Type C Cache Bus Controller Error Codes

MC4_STATUS[31:16] (MSCE) Value		Error Description
0000_0000_0000_0001	0x0001	Inclusion Error from Core 0
0000_0000_0000_0010	0x0002	Inclusion Error from Core 1
0000_0000_0000_0011	0x0003	Write Exclusive Error from Core 0
0000_0000_0000_0100	0x0004	Write Exclusive Error from Core 1
0000_0000_0000_0101	0x0005	Inclusion Error from FSB
0000_0000_0000_0110	0x0006	SNP Stall Error from FSB
0000_0000_0000_0111	0x0007	Write Stall Error from FSB

Table E-8. Type C Cache Bus Controller Error Codes

0000_0000_0000_1000 0x0008	FSB Arb Timeout Error
0000_0000_0000_1001 0x0009	CBC OOD Queue Underflow/overflow
0000_0001_0000_0000 0x0100	Enhanced Intel SpeedStep Technology TM1-TM2 Error
0000_0010_0000_0000 0x0200	Internal Timeout error
0000_0011_0000_0000 0x0300	Internal Timeout Error
0000_0100_0000_0000 0x0400	Intel® Cache Safe Technology Queue Full Error or Disabled-ways-in-a-set overflow
1100_0000_0000_0001 0xC001	Correctable ECC event on outgoing FSB data
1100_0000_0000_0010 0xC002	Correctable ECC event on outgoing Core 0 data
1100_0000_0000_0100 0xC004	Correctable ECC event on outgoing Core 1 data
1110_0000_0000_0001 0xE001	Uncorrectable ECC error on outgoing FSB data
1110_0000_0000_0010 0xE002	Uncorrectable ECC error on outgoing Core 0 data
1110_0000_0000_0100 0xE004	Uncorrectable ECC error on outgoing Core 1 data
— all other encodings —	Reserved

All errors - except for the correctable ECC types - in this table are uncorrectable. The correctable ECC events may supply the ECC syndrome in the Other_Info field of the MC4_STATUS MSR

Table E-9. Decoding Family 0FH Machine Check Codes for Memory Hierarchy Errors

Type	Bit No.	Bit Function	Bit Description
MCA error codes[1]	0-15		
Model specific error codes	16-17	Tag Error Code	Contains the tag error code for this machine check error: 00 = No error detected 01 = Parity error on tag miss with a clean line 10 = Parity error/multiple tag match on tag hit 11 = Parity error/multiple tag match on tag miss
	18-19	Data Error Code	Contains the data error code for this machine check error: 00 = No error detected 01 = Single bit error 10 = Double bit error on a clean line 11 = Double bit error on a modified line
	20	L3 Error	This bit is set if the machine check error originated in the L3 it can be ignored for invalid PIC request errors): 1 = L3 error 0 = L2 error
	21	Invalid PIC Request	Indicates error due to invalid PIC request access was made to PIC space with WB memory): 1 = Invalid PIC request error 0 = No invalid PIC request error
	22-31	Reserved	Reserved
Other Information	32-39	8-bit Error Count	Holds a count of the number of errors since reset. The counter begins at 0 for the first error and saturates at a count of 255.
	40-56	Reserved	Reserved
Status register validity indicators[1]	57-63		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 14, “Machine-Check Architecture,” for more information.

APPENDIX F
APIC BUS MESSAGE FORMATS

This appendix describes the message formats used when transmitting messages on the serial APIC bus. The information described here pertains only to the Pentium and P6 family processors.

F.1 BUS MESSAGE FORMATS

The local and I/O APICs transmit three types of messages on the serial APIC bus: EOI message, short message, and non-focused lowest priority message. The purpose of each type of message and its format are described below.

F.2 EOI MESSAGE

Local APICs send 14-cycle EOI messages to the I/O APIC to indicate that a level triggered interrupt has been accepted by the processor. This interrupt, in turn, is a result of software writing into the EOI register of the local APIC. Table F-1 shows the cycles in an EOI message.

Table F-1. EOI Message (14 Cycles)

Cycle	Bit1	Bit0	
1	1	1	11 = EOI
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	V7	V6	Interrupt vector V7 - V0
7	V5	V4	
8	V3	V2	
9	V1	V0	
10	C	C	Checksum for cycles 6 - 9
11	0	0	
12	A	A	Status Cycle 0
13	A1	A1	Status Cycle 1
14	0	0	Idle

The checksum is computed for cycles 6 through 9. It is a cumulative sum of the 2-bit (Bit1:Bit0) logical data values. The carry out of all but the last addition is added to the sum. If any APIC computes a different checksum than the one appearing on the bus in cycle 10, it signals an error, driving 11 on the APIC bus during cycle 12. In this case, the APICs disregard the message. The sending APIC will receive an appropriate error indication (see Section 9.5.3, “Error Handling”) and resend the message. The status cycles are defined in Table F-4.

F.2.1 Short Message

Short messages (21-cycles) are used for sending fixed, NMI, SMI, INIT, start-up, ExtINT and lowest-priority-with-focus interrupts. Table F-2 shows the cycles in a short message.

Table F-2. Short Message (21 Cycles)

Cycle	Bit1	Bit0	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination Mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	
17	C	C	Checksum for cycles 6-16
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	0	0	Idle

If the physical delivery mode is being used, then cycles 15 and 16 represent the APIC ID and cycles 13 and 14 are considered don't care by the receiver. If the logical delivery mode is being used, then cycles 13 through 16 are the 8-bit logical destination field.

For shorthands of “all-incl-self” and “all-excl-self,” the physical delivery mode and an arbitration priority of 15 (D0:D3 = 1111) are used. The agent sending the message is the only one required to distinguish between the two cases. It does so using internal information.

When using lowest priority delivery with an existing focus processor, the focus processor identifies itself by driving 10 during cycle 19 and accepts the interrupt. This is an indication to other APICs to terminate arbitration. If the focus processor has not been found, the short message is extended on-the-fly to the non-focused lowest-priority message. Note that except for the EOI message, messages generating a checksum or an acceptance error (see Section 9.5.3, “Error Handling”) terminate after cycle 21.

F.2.2 Non-focused Lowest Priority Message

These 34-cycle messages (see Table F-3) are used in the lowest priority delivery mode when a focus processor is not present. Cycles 1 through 20 are same as for the short message. If during the status cycle (cycle 19) the state of the (A:A) flags is 10B, a focus processor has been identified, and the short message format is used (see Table F-2). If the (A:A) flags are set to 00B, lowest priority arbitration is started and the 34-cycles of the non-focused lowest priority message are competed. For other combinations of status flags, refer to Section F.2.3, “APIC Bus Status Cycles.”

Table F-3. Non-Focused Lowest Priority Message (34 Cycles)

Cycle	Bit0	Bit1	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	

Table F-3. Non-Focused Lowest Priority Message (34 Cycles) (Contd.)

Cycle	Bit0	Bit1	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	
17	C	C	Checksum for cycles 6-16
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	P7	0	P7 - P0 = Inverted Processor Priority
22	P6	0	
23	P5	0	
24	P4	0	
25	P3	0	
26	P2	0	
27	P1	0	
28	P0	0	
29	ArbID3	0	Arbitration ID 3 -0
30	ArbID2	0	
31	ArbID1	0	
32	ArbID0	0	
33	A2	A2	Status Cycle
34	0	0	Idle

Cycles 21 through 28 are used to arbitrate for the lowest priority processor. The processors participating in the arbitration drive their inverted processor priority on the bus. Only the local APICs having free interrupt slots participate in the lowest priority arbitration. If no such APIC exists, the message will be rejected, requiring it to be tried at a later time.

Cycles 29 through 32 are also used for arbitration in case two or more processors have the same lowest priority. In the lowest priority delivery mode, all combinations of errors in cycle 33 (A2 A2) will set the “accept error” bit in the error status register (see Figure 9-9). Arbitration priority update is performed in cycle 20, and is not affected by errors detected in cycle 33. Only the local APIC that wins in the lowest priority arbitration, drives cycle 33. An error in cycle 33 will force the sender to resend the message.

F.2.3 APIC Bus Status Cycles

Certain cycles within an APIC bus message are status cycles. During these cycles the status flags (A:A) and (A1:A1) are examined. Table F-4 shows how these status flags are interpreted, depending on the current delivery mode and existence of a focus processor.

Table F-4. APIC Bus Status Cycles Interpretation

Delivery Mode	A Status	A1 Status	A2 Status	Update ArbID and Cycle#	Message Length	Retry
EOI	00: CS_OK	10: Accept	XX:	Yes, 13	14 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 13	14 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	14 Cycle	Yes
	11: CS_Error	XX:	XX:	No	14 Cycle	Yes
	10: Error	XX:	XX:	No	14 Cycle	Yes
	01: Error	XX:	XX:	No	14 Cycle	Yes
Fixed	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes
NMI, SMI, INIT, ExtINT, Start-Up	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes

Table F-4. APIC Bus Status Cycles Interpretation (Contd.)

Delivery Mode	A Status	A1 Status	A2 Status	Update ArbID and Cycle#	Message Length	Retry
Lowest	00: CS_OK, NoFocus	11: Do Lowest	10: Accept	Yes, 20	34 Cycle	No
	00: CS_OK, NoFocus	11: Do Lowest	11: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	11: Do Lowest	0X: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	10: End and Retry	XX:	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	0X: Error	XX:	No	34 Cycle	Yes
	10: CS_OK, Focus	XX:	XX:	Yes, 20	34 Cycle	No
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes

APPENDIX G
VMX CAPABILITY REPORTING FACILITY

The ability of a processor to support VMX operation and related instructions is indicated by CPUID.1:ECX.VMX[bit 5] = 1. A value 1 in this bit indicates support for VMX features.

Support for specific features detailed in Chapter 20 and other VMX chapters is determined by reading values from a set of capability MSRs. These MSRs are indexed starting at MSR address 480H. VMX capability MSRs are read-only; an attempt to write them (with WRMSR) produces a general-protection exception (#GP(0)). They do not exist on processors that do not support VMX operation; an attempt to read them (with RDMSR) on such processors produces a general-protection exception (#GP(0)).

G.1 BASIC VMX INFORMATION

The IA32_VMX_BASIC MSR (index 480H) consists of the following fields:

- Bits 31:0 contain the 32-bit VMCS revision identifier used by the processor. Logical processors that use the same VMCS revision identifier use the same size for VMCS regions (see next item)
- Bits 44:32 report the number of bytes that software should allocate for the VMXON region and any VMCS region. It is a value greater than 0 and at most 4096 (bit 44 is set if and only if bits 43:32 are clear).
- Bit 48 indicates the width of the physical addresses that may be used for the VMXON region, each VMCS, and data structures referenced by pointers in a VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions). If the bit is 0, these addresses are limited to the processor's physical-address width.[1] If the bit is 1, these addresses are limited to 32 bits. This bit is always 0 for processors that support Intel 64 architecture and is always 1 for processors that do not support Intel 64 architecture.
- If bit 49 is read as 1, the logical processor supports the dual-monitor treatment of system-management interrupts and system-management mode. See Section 26.15 for details of this treatment.
- Bits 53:50 report the memory type that the logical processor uses to access the VMCS for VMREAD and VMWRITE and to access the VMCS, data structures referenced by pointers in the VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions), and the MSEG header during VM entries, VM exits, and in VMX non-root operation.[2]

[1] On processors that support Intel 64 architecture, the pointer must not set bits beyond the processor's physical address width.

The first processors to support VMX operation use the write-back type. The values used are given in Table G-1.

Table G-1. Memory Types Used For VMCS Access

Value(s)	Field
0	Uncacheable (UC)
1–5	Not used
6	Write Back (WB)
7–15	Not used

If software needs to access these data structures (e.g., to modify the contents of the MSR bitmaps), it can configure the paging structures to map them into the linear-address space. If it does so, it should establish mappings that use the memory type reported in this MSR.[1]

- If bit 54 is read as 1, the logical processor reports information in the VM-exit instruction-information field on VM exits due to execution of the INS and OUTS instructions. This reporting is done only if this bit is read as 1.
- Bit 55 is read as 1 if any VMX controls that default to 1 may be cleared to 0. See Appendix G.2 for details. It also reports support for the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix G.3.1, Appendix G.3.2, Appendix G.4, and Appendix G.5 for details.
- The values of bits 47:45 and bits 63:56 are reserved and are read as 0.

G.2 RESERVED CONTROLS AND DEFAULT SETTINGS

As noted in Chapter 20, “Virtual-Machine Control Structures”, certain VMX controls are reserved and must be set to a specific value (0 or 1) determined by the processor. The specific value to which a reserved control must be set is its **default setting**.

[2] If the MTRRs are disabled by clearing the E bit (bit 11) in the IA32_MTRR_DEF_TYPE MSR, the logical processor uses the UC memory type to access the indicated data structures, regardless of the value reported in bits 53:50 in the IA32_VMX_BASIC MSR. The processor will also use the UC memory type if the setting of CR0.CD on this logical processor (or another logical processor on the same physical processor) would cause it to do so for all memory accesses. Note that the values of IA32_MTRR_DEF_TYPE.E and CR0.CD do not affect the value reported in IA32_VMX_BASIC[53:50].

[1] Alternatively, software may map any of these regions or structures with the UC memory type. (This may be necessary for the MSEG header.) Doing so is discouraged unless necessary as it will cause the performance of software accesses to those structures to suffer. Note that the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with the exceptions noted.

Software can discover the default setting of a reserved control by consulting the appropriate VMX capability MSR (see Appendix G.3 through Appendix G.5).

Future processors may define new functionality for one or more reserved controls. Such processors would allow each newly defined control to be set either to 0 or to 1. Software that does not desire a control's new functionality should set the control to its default setting. For that reason, it is useful for software to know the default settings of the reserved controls.

Default settings partition the various controls into the following classes:

- **Always-flexible**. These have never been reserved.
- **Default0**. These are (or have been) reserved with a default setting of 0.
- **Default1**. They are (or have been) reserved with a default setting of 1.

As noted in Appendix G.1, a logical processor uses bit 55 of the IA32_VMX_BASIC MSR to indicate whether any of the default1 controls may be 0:

- If bit 55 of the IA32_VMX_BASIC MSR is read as 0, all the default1 controls are reserved and must be 1. VM entry will fail if any of these controls are 1 (see Section 22.2.1).
- If bit 55 of the IA32_VMX_BASIC MSR is read as 1, not all the default1 controls are reserved, and some (but not necessarily all) may be 0. The CPU supports four (4) new VMX capability MSRs: IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix G.3 through Appendix G.5 for details. (These MSRs are not supported if bit 55 of the IA32_VMX_BASIC MSR is read as 0.)

See Section 27.5.1 for recommended software algorithms for proper capability detection of the default1 controls.

G.3 VM-EXECUTION CONTROLS

There are separate capability MSRs for the pin-based VM-execution controls, the primary processor-based VM-execution controls, and the secondary processor-based VM-execution controls. These are described in Appendix G.3.1, Appendix G.3.2, and Appendix G.3.3, respectively.

G.3.1 Pin-Based VM-Execution Controls

The IA32_VMX_PINBASED_CTLS MSR (index 481H) reports on the allowed settings of most of the pin-based VM-execution controls (see Section 20.6.1):

- Bits 31:0 indicate the **allowed 0-settings** of these controls. VM entry fails if bit X is 0 in the pin-based VM-execution controls and bit X is 1 in this MSR.

  Exceptions are made for the pin-based VM-execution controls in the default1 class (see Appendix G.2). These are bits 1, 2, and 4; the corresponding bits of the IA32_VMX_PINBASED_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

- — If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any pin-based VM-execution control in the default1 class is 0.
- — If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLS MSR (see below) reports which of the pin-based VM-execution controls in the default1 class can be 0 on VM entry.

- Bits 63:32 indicate the **allowed 1-settings** of these controls. VM entry fails if bit X is 1 in the pin-based VM-execution controls and bit 32+X is 0 in this MSR.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLS MSR (index 48DH) reports on the allowed settings of all of the pin-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X is 0 in the pin-based VM-execution controls and bit X is 1 in this MSR. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the pin-based VM-execution controls and bit 32+X is 0 in this MSR.

Note that it is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the pin-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_PINBASED_CTLS MSR. (The IA32_VMX_TRUE_PINBASED_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_TRUE_PINBASED_CTLS MSR. Assuming that software knows that the default1 class of pin-based VM-execution controls contains bits 1, 2, and 4, there is no need for software to consult the IA32_VMX_PINBASED_CTLS MSR.

G.3.2 Primary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS MSR (index 482H) reports on the allowed settings of most of the primary processor-based VM-execution controls (see Section 20.6.2):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X is 0 in the primary processor-based VM-execution controls and bit X is 1 in this MSR.

  Exceptions are made for the primary processor-based VM-execution controls in the default1 class (see Appendix G.2). These are bits 1, 4–6, 8, 13–16, and 26; the corresponding bits of the IA32_VMX_PROCBASED_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

  - — If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any of the primary processor-based VM-execution controls in the default1 class is 0.
  - — If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLS MSR (see below) reports which of the primary processor-based VM-execution controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the primary processor-based VM-execution controls and bit 32+X is 0 in this MSR.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLS MSR (index 48EH) reports on the allowed settings of all of the primary processor-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X is 0 in the primary processor-based VM-execution controls and bit X is 1 in this MSR. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the primary processor-based VM-execution controls and bit 32+X is 0 in this MSR.

Note that it is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the primary processor-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the primary processor-based VM-execution controls is contained in the IA32_VMX_PROCBASED_CTLS MSR. (The IA32_VMX_TRUE_PROCBASED_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_TRUE_PROCBASED_CTLS MSR. Assuming that software knows that the default1 class of pin-based VM-execution controls contains bits 1, 4–6, 8, 13–16, and 26, there is no need for software to consult the IA32_VMX_PROCBASED_CTLS MSR.

G.3.3 Secondary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS2 MSR (index 48BH) reports on the allowed settings of the secondary processor-based VM-execution controls (see Section 20.6.2). VM entries perform the following checks:

- Bits 31:0 indicate the allowed 0-settings of these controls. These bits are always 0. This fact indicates that VM entry does not fail simply because if the “activate secondary controls” primary processor-based VM-execution control is 1 and some bit is 0 in the secondary processor-based VM-execution controls.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if the “activate secondary controls” primary processor-based VM-execution control is

1, bit X is 1 in the secondary processor-based VM-execution controls, and bit 32+X is 0 in this MSR.

The IA32_VMX_PROCBASED_CTLS2 MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1).

G.4 VM-EXIT CONTROLS

The IA32_VMX_EXIT_CTLS MSR (index 483H) reports on the allowed settings of most of the VM-exit controls (see Section 20.7.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X is 0 in the VM-exit controls and bit X is 1 in this MSR.

  Exceptions are made for the VM-exit controls in the default1 class (see Appendix G.2). These are bits 0–8, 10, 11, 13, 14, 16, and 17; the corresponding bits of the IA32_VMX_EXIT_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any VM-exit control in the default1 class is 0.

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (see below) reports which of the VM-exit controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the VM-exit controls and bit 32+X is 0 in this MSR.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (index 48FH) reports on the allowed settings of all of the VM-exit controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X is 0 in the VM-exit controls and bit X is 1 in this MSR. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the VM-exit controls and bit 32+X is 0 in this MSR.

Note that it is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the VM-exit controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the VM-exit controls is contained in the IA32_VMX_EXIT_CTLS MSR. (The IA32_VMX_TRUE_EXIT_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the VM-exit controls is contained in the IA32_VMX_TRUE_EXIT_CTLS MSR. Assuming that software knows that the default1 class of VM-exit controls contains bits 0–8, 10, 11, 13, 14, 16, and 17, there is no need for software to consult the IA32_VMX_EXIT_CTLS MSR.

G.5 VM-ENTRY CONTROLS

The IA32_VMX_ENTRY_CTLS MSR (index 484H) reports on the allowed settings of most of the VM-entry controls (see Section 20.8.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X is 0 in the VM-entry controls and bit X is 1 in this MSR.

  Exceptions are made for the VM-entry controls in the default1 class (see Appendix G.2). These are bits 0–8 and 12; the corresponding bits of the IA32_VMX_ENTRY_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any VM-entry control in the default1 class is 0.

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLS MSR (see below) reports which of the VM-entry controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the VM-entry controls and bit 32+X is 0 in this MSR.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1,
the IA32_VMX_TRUE_ENTRY_CTLS MSR (index 490H) reports on the allowed settings of all of the VM-entry controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X is 0 in the VM-entry controls and bit X is 1 in this MSR. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the VM-entry controls and bit 32+X is 0 in this MSR.

Note that it is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the VM-entry controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_ENTRY_CTLS MSR. (The IA32_VMX_TRUE_ENTRY_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_TRUE_ENTRY_CTLS MSR. Assuming that software knows that the default1 class of VM-entry controls contains bits 0–8 and 12, there is no need for software to consult the IA32_VMX_ENTRY_CTLS MSR.

G.6 MISCELLANEOUS DATA

The IA32_VMX_MISC MSR (index 485H) consists of the following fields:

- Bits 4:0 report a value X that specifies the relationship between the rate of the VMX-preemption timer and that of the timestamp counter (TSC). Specifically, the

VMX-preemption timer (if it is active) counts down by 1 every time bit X in the TSC changes due to a TSC increment.

- Bits 8:6 report, as a bitmap, the activity states supported by the implementation:
  - Bit 6 reports (if set) the support for activity state 1 (HLT).
  - Bit 7 reports (if set) the support for activity state 2 (shutdown).
  - Bit 8 reports (if set) the support for activity state 3 (wait-for-SIPI).

  If an activity state is not supported, the implementation causes a VM entry to fail if it attempts to establish that activity state. Note that all implementations support VM entry to activity state 0 (active).
- Bits 24:16 indicate the number of CR3-target values supported by the processor. This number is a value between 0 and 256, inclusive (bit 24 is set if and only if bits 23:16 are clear).
- Bits 27:25 is used to compute the recommended maximum number of MSRs that should appear in the VM-exit MSR-store list, the VM-exit MSR-load list, or the VM-entry MSR-load list. Specifically, if the value bits 27:25 of IA32_VMX_MISC is N, then 512 * (N + 1) is the recommended maximum number of MSRs to be included in each list. If the limit is exceeded, undefined processor behavior may result (including a machine check during the VMX transition).
- Bits 63:32 report the 32-bit MSEG revision identifier used by the processor.
- Bit 5, bits 15:9, and bits 31:28 are reserved and are read as 0.

G.7 VMX-FIXED BITS IN CR0

The IA32_VMX_CR0_FIXED0 MSR (index 486H) and IA32_VMX_CR0_FIXED1 MSR (index 487H) indicate how bits in CR0 may be set in VMX operation. They report on bits in CR0 that are allowed to be 0 and to be 1, respectively, in VMX operation. If bit X is 1 in IA32_VMX_CR0_FIXED0, then that bit of CR0 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in IA32_VMX_CR0_FIXED1, then that bit of CR0 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in IA32_VMX_CR0_FIXED0, then that bit is also 1 in IA32_VMX_CR0_FIXED1; if bit X is 0 in IA32_VMX_CR0_FIXED1, then that bit is also 0 in IA32_VMX_CR0_FIXED0. Thus, each bit in CR0 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in IA32_VMX_CR0_FIXED0 and 1 in IA32_VMX_CR0_FIXED1).

G.8 VMX-FIXED BITS IN CR4

The IA32_VMX_CR4_FIXED0 MSR (index 488H) and IA32_VMX_CR4_FIXED1 MSR (index 489H) indicate how bits in CR4 may be set in VMX operation. They report on bits in CR4 that are allowed to be 0 and 1, respectively, in VMX operation. If bit X is 1 in IA32_VMX_CR4_FIXED0, then that bit of CR4 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in IA32_VMX_CR4_FIXED1, then that bit of CR4 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in IA32_VMX_CR4_FIXED0, then that bit is also 1 in IA32_VMX_CR4_FIXED1; if bit X is 0 in IA32_VMX_CR4_FIXED1, then that bit is also 0 in IA32_VMX_CR4_FIXED0. Thus, each bit in CR4 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in IA32_VMX_CR4_FIXED0 and 1 in IA32_VMX_CR4_FIXED1).

G.9 VMCS ENUMERATION

The IA32_VMX_VMCS_ENUM MSR (index 48AH) provides information to assist software in enumerating fields in the VMCS.

As noted in Section 20.10.2, each field in the VMCS is associated with a 32-bit encoding which is structured as follows:

- Bits 31:15 are reserved (must be 0).
- Bits 14:13 indicate the field’s width.
- Bit 12 is reserved (must be 0).
- Bits 11:10 indicate the field’s type.
- Bits 9:1 is an index field that distinguishes different fields with the same width and type.
- Bit 0 indicates access type.

IA32_VMX_VMCS_ENUM indicates to software the highest index value used in the encoding of any field supported by the processor:

- Bits 9:1 contain the highest index value used for any VMCS encoding.
- Bit 0 and bits 63:10 are reserved and are read as 0.

G.10 VPID AND EPT CAPABILITIES

The IA32_VMX_EPT_VPID_CAP MSR (index 48CH) reports information about the capabilities of the logical processor with regard to virtual-processor identifiers (VPIDs, Chapter 24) and extended page tables (EPT, Chapter 25):

- If bit 0 is read as 1, the logical processor allows software to configure EPT paging-structure entries in which bits 2:0 have value 100b (indicating an execute-only translation).
- Bit 6 indicates support for a page-walk length of 3.
- If bit 8 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be uncacheable (UC); see Section 20.6.11.
- If bit 14 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be write-back (WB).

- If bit 16 is read as 1, the logical processor allows software to configure EPT PDEs to map a 2-Mbyte page (by setting bit 7).
- Support for the INVEPT instruction (seeChapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B* and Section 25.6.2).
  - — If bit 20 is read as 1, the INVEPT instruction is supported.
  - — If bit 25 is read as 1, the single-context INVEPT type is supported.
  - — If bit 26 is read as 1, the all-context INVEPT type is supported.
- Support for the INVVPID instruction (see Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B* and Section 24.3).
  - — If bit 32 is read as 1, the INVVPID instruction is supported.
  - — If bit 40 is read as 1, the individual-address INVVPID type is supported.
  - — If bit 41 is read as 1, the single-context INVVPID type is supported.
  - — If bit 42 is read as 1, the all-context INVVPID type is supported.
  - — If bit 43 is read as 1, the single-context-retaining-globals INVVPID type is supported.
- Bits 5:1, bit 7, bits 13:9, bit 15, bits 19:17, bits 24:21, bits 31:27, bits 39:33, and bits 63:44 are reserved and are read as 0.

The IA32_VMX_EPT_VPID_CAP MSR exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1) and that support either the 1-setting of the "enable EPT" VM-execution control (only if bit 33 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1) or the 1-setting of the "enable VPID" VM-execution control (only if bit 37 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1).

APPENDIX H
FIELD ENCODING IN VMCS

Every component of the VMCS is encoded by a 32-bit field that can be used by VMREAD and VMWRITE. Section 20.10.2 describes the structure of the encoding space (the meanings of the bits in each 32-bit encoding).

This appendix enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.)

H.1 16-BIT FIELDS

A value of 0 in bits 14:13 of an encoding indicates a 16-bit field. Only guest-state areas and the host-state area contain 16-bit fields. As noted in Section 20.10.2, each 16-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

H.1.1 16-Bit Control Field

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. There is only one such 16-bit field as given in Table H-1.

Table H-1. Encoding for 32-Bit Control Fields (0000_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Virtual-processor identifier (VPID)[1]	000000000B	00000000H

NOTES:

[1] This field exists only on processors that support the 1-setting of the “enable VPID” VM-execution control.

H.1.2 16-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table H-2 enumerates 16-bit guest-state fields.

Table H-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES selector	000000000B	00000800H

Table H-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest CS selector	000000001B	00000802H
Guest SS selector	000000010B	00000804H
Guest DS selector	000000011B	00000806H
Guest FS selector	000000100B	00000808H
Guest GS selector	000000101B	0000080AH
Guest LDTR selector	000000110B	0000080CH
Guest TR selector	000000111B	0000080EH

H.1.3 16-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table H-3 enumerates the 16-bit host-state fields.

Table H-3. Encodings for 16-Bit Host-State Fields (0000_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host ES selector	000000000B	00000C00H
Host CS selector	000000001B	00000C02H
Host SS selector	000000010B	00000C04H
Host DS selector	000000011B	00000C06H
Host FS selector	000000100B	00000C08H
Host GS selector	000000101B	00000C0AH
Host TR selector	000000110B	00000C0CH

H.2 64-BIT FIELDS

A value of 1 in bits 14:13 of an encoding indicates a 64-bit field. There are 64-bit fields only for controls and for guest state. As noted in Section 20.10.2, every 64-bit field has two encodings, which differ on bit 0, the access type. Thus, each such field has an even encoding for full access and an odd encoding for high access.

H.2.1 64-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table H-4 enumerates the 64-bit control fields.

Table H-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)

Field Name	Index	Encoding
Address of I/O bitmap A (full)	000000000B	00002000H
Address of I/O bitmap A (high)	000000000B	00002001H
Address of I/O bitmap B (full)	000000001B	00002002H
Address of I/O bitmap B (high)	000000001B	00002003H
Address of MSR bitmaps (full)[1]	000000010B	00002004H
Address of MSR bitmaps (high)[1]	000000010B	00002005H
VM-exit MSR-store address (full)	000000011B	00002006H
VM-exit MSR-store address (high)	000000011B	00002007H
VM-exit MSR-load address (full)	000000100B	00002008H
VM-exit MSR-load address (high)	000000100B	00002009H
VM-entry MSR-load address (full)	000000101B	0000200AH
VM-entry MSR-load address (high)	000000101B	0000200BH
Executive-VMCS pointer (full)	000000110B	0000200CH
Executive-VMCS pointer (high)	000000110B	0000200DH
TSC offset (full)	000001000B	00002010H
TSC offset (high)	000001000B	00002011H
Virtual-APIC address (full)[2]	000001001B	00002012H
Virtual-APIC address (high)[2]	000001001B	00002013H
APIC-access address (full)[3]	000001010B	00002014H
APIC-access address (high)[3]	000001010B	00002015H
EPT pointer (EPTP; full)[4]	000001101B	0000201AH
EPT pointer (EPTP; high)[4]	000001101B	0000201BH

NOTES:

1. This field exists only on processors that support the 1-setting of the “use MSR bitmaps” VM-execution control.
2. This field exists only on processors that support either the 1-setting of the “use TPR shadow” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.

4. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

H.2.2 64-Bit Read-Only Data Field

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. There is only one such 64-bit field as given in Table H-5.(As with other 64-bit fields, this one has two encodings.)

Table H-5. Encodings for 64-Bit Read-Only Data Field (0010_01xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest-physical address (full)	000000000B	00002400H
Guest-physical address (high)	000000000B	00002401H

H.2.3 64-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table H-6 enumerates the 64-bit guest-state fields.

Table H-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
VMCS link pointer (full)	000000000B	00002800H
VMCS link pointer (high)	000000000B	00002801H
Guest IA32_DEBUGCTL (full)	000000001B	00002802H
Guest IA32_DEBUGCTL (high)	000000001B	00002803H
Guest IA32_PAT (full)	000000010B	00002804H
Guest IA32_PAT (high)	000000010B	00002805H
Guest IA32_EFER (full)	000000011B	00002806H
Guest IA32_EFER (high)	000000011B	00002807H
Guest IA32_PERF_GLOBAL_CTRL (full)	000000100B	00002808H
Guest IA32_PERF_GLOBAL_CTRL (high)	000000100B	00002809H
Guest PDPTE0 (full)	000000101B	0000280AH
Guest PDPTE0 (high)	000000101B	0000280BH
Guest PDPTE1 (full)	000000110B	0000280CH
Guest PDPTE1 (high)	000000110B	0000280DH

Table H-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest PDPTE2 (full)	000000111B	0000280EH
Guest PDPTE2 (high)	000000111B	0000280FH
Guest PDPTE3 (full)	000001000B	00002810H
Guest PDPTE3 (high)	000001000B	00002811H

H.2.4 64-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table H-7 enumerates the 64-bit control fields.

Table H-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

Field Name	Index	Encoding
Host IA32_PAT (full)	000000000B	00002C00H
Host IA32_PAT (high)	000000000B	00002C01H
Host IA32_EFER (full)	000000001B	00002C02H
Host IA32_EFER (high)	000000001B	00002C03H
Host IA32_PERF_GLOBAL_CTRL (full)	000000010B	00002C04H
Host IA32_PERF_GLOBAL_CTRL (high)	000000010B	00002C05H

H.3 32-BIT FIELDS

A value of 2 in bits 14:13 of an encoding indicates a 32-bit field. As noted in Section 20.10.2, each 32-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

H.3.1 32-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table H-8 enumerates the 32-bit control fields.

Table H-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Pin-based VM-execution controls	000000000B	00004000H

Table H-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Primary processor-based VM-execution controls	000000001B	00004002H
Exception bitmap	000000010B	00004004H
Page-fault error-code mask	000000011B	00004006H
Page-fault error-code match	000000100B	00004008H
CR3-target count	000000101B	0000400AH
VM-exit controls	000000110B	0000400CH
VM-exit MSR-store count	000000111B	0000400EH
VM-exit MSR-load count	000001000B	00004010H
VM-entry controls	000001001B	00004012H
VM-entry MSR-load count	000001010B	00004014H
VM-entry interruption-information field	000001011B	00004016H
VM-entry exception error code	000001100B	00004018H
VM-entry instruction length	000001101B	0000401AH
TPR threshold[1]	000001110B	0000401CH
Secondary processor-based VM-execution controls[2]	000001111b	0000401EH

NOTES:

1. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control.

H.3.2 32-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table H-9 enumerates the 32-bit read-only data fields.

Table H-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
VM-instruction error	000000000B	00004400H
Exit reason	000000001B	00004402H
VM-exit interruption information	000000010B	00004404H
VM-exit interruption error code	000000011B	00004406H

Table H-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
IDT-vectoring information field	000000100B	00004408H
IDT-vectoring error code	000000101B	0000440AH
VM-exit instruction length	000000110B	0000440CH
VM-exit instruction information	000000111B	0000440EH

H.3.3 32-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table H-10 enumerates the 32-bit guest-state fields.

Table H-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES limit	000000000B	00004800H
Guest CS limit	000000001B	00004802H
Guest SS limit	000000010B	00004804H
Guest DS limit	000000011B	00004806H
Guest FS limit	000000100B	00004808H
Guest GS limit	000000101B	0000480AH
Guest LDTR limit	000000110B	0000480CH
Guest TR limit	000000111B	0000480EH
Guest GDTR limit	000001000B	00004810H
Guest IDTR limit	000001001B	00004812H
Guest ES access rights	000001010B	00004814H
Guest CS access rights	000001011B	00004816H
Guest SS access rights	000001100B	00004818H
Guest DS access rights	000001101B	0000481AH
Guest FS access rights	000001110B	0000481CH
Guest GS access rights	000001111B	0000481EH
Guest LDTR access rights	000010000B	00004820H
Guest TR access rights	000010001B	00004822H
Guest interruptibility state	000010010B	00004824H

Table H-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest activity state	000010011B	00004826H
Guest SMBASE	000010100B	00004828H
Guest IA32_SYSENTER_CS	000010101B	0000482AH
VMX-preemption timer value	000010111B	0000482EH

The limit fields for GDTR and IDTR are defined to be 32 bits in width even though these fields are only 16-bits wide in the Intel 64 and IA-32 architectures. VM entry ensures that the high 16 bits of both these fields are cleared to 0.

H.3.4 32-Bit Host-State Field

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. There is only one such 32-bit field as given in Table H-11.

Table H-11. Encoding for 32-Bit Host-State Field (0100_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host IA32_SYSENTER_CS	000000000B	00004C00H

H.4 NATURAL-WIDTH FIELDS

A value of 3 in bits 14:13 of an encoding indicates a natural-width field. As noted in Section 20.10.2, each of these fields allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

H.4.1 Natural-Width Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table H-12 enumerates the natural-width control fields.

Table H-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
CR0 guest/host mask	000000000B	00006000H
CR4 guest/host mask	000000001B	00006002H
CR0 read shadow	000000010B	00006004H

Table H-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
CR4 read shadow	000000011B	00006006H
CR3-target value 0	000000100B	00006008H
CR3-target value 1	000000101B	0000600AH
CR3-target value 2	000000110B	0000600CH
CR3-target value 3[1]	000000111B	0000600EH

NOTES:

[1] If a future implementation supports more than 4 CR3-target values, they will be encoded consecutively following the 4 encodings given here.

H.4.2 Natural-Width Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table H-13 enumerates the natural-width read-only data fields.

Table H-13. Encodings for Natural-Width Read-Only Data Fields (0110_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
Exit qualification	000000000B	00006400H
I/O RCX	000000001B	00006402H
I/O RSI	000000010B	00006404H
I/O RDI	000000011B	00006406H
I/O RIP	000000100B	00006408H
Guest-linear address	000000101B	0000640AH

H.4.3 Natural-Width Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table H-14 enumerates the natural-width guest-state fields.

Table H-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest CR0	000000000B	00006800H

Table H-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest CR3	000000001B	00006802H
Guest CR4	000000010B	00006804H
Guest ES base	000000011B	00006806H
Guest CS base	000000100B	00006808H
Guest SS base	000000101B	0000680AH
Guest DS base	000000110B	0000680CH
Guest FS base	000000111B	0000680EH
Guest GS base	000001000B	00006810H
Guest LDTR base	000001001B	00006812H
Guest TR base	000001010B	00006814H
Guest GDTR base	000001011B	00006816H
Guest IDTR base	000001100B	00006818H
Guest DR7	000001101B	0000681AH
Guest RSP	000001110B	0000681CH
Guest RIP	000001111B	0000681EH
Guest RFLAGS	000010000B	00006820H
Guest pending debug exceptions	000010001B	00006822H
Guest IA32_SYSENTER_ESP	000010010B	00006824H
Guest IA32_SYSENTER_EIP	000010011B	00006826H

The base-address fields for ES, CS, SS, and DS in the guest-state area are defined to be natural-width (with 64 bits on processors supporting Intel 64 architecture) even though these fields are only 32-bits wide in the Intel 64 architecture. VM entry ensures that the high 32 bits of these fields are cleared to 0.

H.4.4 Natural-Width Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table H-15 enumerates the natural-width host-state fields.

Table H-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host CR0	000000000B	00006C00H
Host CR3	000000001B	00006C02H
Host CR4	000000010B	00006C04H
Host FS base	000000011B	00006C06H
Host GS base	000000100B	00006C08H
Host TR base	000000101B	00006C0AH
Host GDTR base	000000110B	00006C0CH
Host IDTR base	000000111B	00006C0EH
Host IA32_SYSENTER_ESP	000001000B	00006C10H
Host IA32_SYSENTER_EIP	000001001B	00006C12H
Host RSP	000001010B	00006C14H
Host RIP	000001011B	00006C16H

APPENDIX I
VMX BASIC EXIT REASONS

Every VM exit writes a 32-bit exit reason to the VMCS (see Section 20.9.1). Certain VM-entry failures also do this (see Section 22.7). The low 16 bits of the exit-reason field form the basic exit reason which provides basic information about the cause of the VM exit or VM-entry failure.

Table I-1 lists values for basic exit reasons and explains their meaning. Entries apply to VM exits, unless otherwise noted.

Table I-1. Basic Exit Reasons

Basic Exit Reason	Description
0	**Exception or non-maskable interrupt (NMI).** Either: 1: Guest software caused an exception and the bit in the exception bitmap associated with exception's vector was 1. 2: An NMI was delivered to the logical processor and the "NMI exiting" VM-execution control was 1. This case includes executions of BOUND that cause #BR, executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UD2 (they cause #UD).
1	**External interrupt.** An external interrupt arrived and the "external-interrupt exiting" VM-execution control was 1.
2	**Triple fault.** The logical processor encountered an exception while attempting to call the double-fault handler and that exception did not itself cause a VM exit due to the exception bitmap.
3	**INIT signal.** An INIT signal arrived
4	**Start-up IPI (SIPI).** A SIPI arrived while the logical processor was in the "wait-for-SIPI" state.
5	**I/O system-management interrupt (SMI).** An SMI arrived immediately after retirement of an I/O instruction and caused an SMM VM exit (see Section 26.15.2).
6	**Other SMI.** An SMI arrived and caused an SMM VM exit (see Section 26.15.2) but not immediately after retirement of an I/O instruction.
7	**Interrupt window.** At the beginning of an instruction, RFLAGS.IF was 1; events were not blocked by STI or by MOV SS; and the "interrupt-window exiting" VM-execution control was 1.
8	**NMI window.** At the beginning of an instruction, there was no virtual-NMI blocking; events were not blocked by MOV SS; and the "NMI-window exiting" VM-execution control was 1.
9	**Task switch.** Guest software attempted a task switch.
10	**CPUID.** Guest software attempted to execute CPUID.

Table I-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
11	**GETSEC.** Guest software attempted to execute GETSEC.
12	**HLT.** Guest software attempted to execute HLT and the "HLT exiting" VM-execution control was 1.
13	**INVD.** Guest software attempted to execute INVD.
14	**INVLPG.** Guest software attempted to execute INVLPG and the "INVLPG exiting" VM-execution control was 1.
15	**RDPMC.** Guest software attempted to execute RDPMC and the "RDPMC exiting" VM-execution control was 1.
16	**RDTSC.** Guest software attempted to execute RDTSC and the "RDTSC exiting" VM-execution control was 1.
17	**RSM.** Guest software attempted to execute RSM in SMM.
18	**VMCALL.** VMCALL was executed either by guest software (causing an ordinary VM exit) or by the executive monitor (causing an SMM VM exit; see Section 26.15.2).
19	**VMCLEAR.** Guest software attempted to execute VMCLEAR.
20	**VMLAUNCH.** Guest software attempted to execute VMLAUNCH.
21	**VMPTRLD.** Guest software attempted to execute VMPTRLD.
22	**VMPTRST.** Guest software attempted to execute VMPTRST.
23	**VMREAD.** Guest software attempted to execute VMREAD.
24	**VMRESUME.** Guest software attempted to execute VMRESUME.
25	**VMWRITE.** Guest software attempted to execute VMWRITE.
26	**VMXOFF.** Guest software attempted to execute VMXOFF.
27	**VMXON.** Guest software attempted to execute VMXON.
28	**Control-register accesses.** Guest software attempted to access CR0, CR3, CR4, or CR8 using CLTS, LMSW, or MOV CR and the VM-execution control fields indicate that a VM exit should occur (see Section 21.1 for details). This basic exit reason is not used for trap-like VM exits following executions of the MOV to CR8 instruction when the "use TPR shadow" VM-execution control is 1.
29	**MOV DR.** Guest software attempted a MOV to or from a debug register and the "MOV-DR exiting" VM-execution control was 1.
30	**I/O instruction.** Guest software attempted to execute an I/O instruction and either: 1: The "use I/O bitmaps" VM-execution control was 0 and the "unconditional I/O exiting" VM-execution control was 1. 2: The "use I/O bitmaps" VM-execution control was 1 and a bit in the I/O bitmap associated with one of the ports accessed by the I/O instruction was 1.

Table I-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
31	**RDMSR.** Guest software attempted to execute RDMSR and either: 1: The "use MSR bitmaps" VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. 3: The value of RCX was in the range 00000000H – 00001FFFH and the *n*th bit in read bitmap for low MSRs is 1, where *n* was the value of RCX. 4: The value of RCX is in the range C0000000H – C0001FFFH and the *n*th bit in read bitmap for high MSRs is 1, where *n* is the value of RCX & 00001FFFH.
32	**WRMSR.** Guest software attempted to execute WRMSR and either: 1: The "use MSR bitmaps" VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. 3: The value of RCX was in the range 00000000H – 00001FFFH and the *n*th bit in write bitmap for low MSRs is 1, where *n* was the value of RCX. 4: The value of RCX is in the range C0000000H – C0001FFFH and the *n*th bit in write bitmap for high MSRs is 1, where *n* is the value of RCX & 00001FFFH.
33	**VM-entry failure due to invalid guest state.** A VM entry failed one of the checks identified in Section 22.3.1.
34	**VM-entry failure due to MSR loading.** A VM entry failed in an attempt to load MSRs. See Section 22.4.
36	**MWAIT.** Guest software attempted to execute MWAIT and the "MWAIT exiting" VM-execution control was 1.
37	**Monitor trap flag.** A VM entry occurred due to the 1-setting of the "monitor trap flag" VM-execution control and injection of an MTF VM exit as part of VM entry. See Section 21.7.2.
39	**MONITOR.** Guest software attempted to execute MONITOR and the "MONITOR exiting" VM-execution control was 1.
40	**PAUSE.** Guest software attempted to execute PAUSE and the "PAUSE exiting" VM-execution control was 1.
41	**VM-entry failure due to machine check.** A machine check occurred during VM entry (see Section 22.8).
43	**TPR below threshold.** The logical processor determined that the value of the TPR shadow was below that of the TPR threshold VM-execution control field while the "use TPR shadow" VM-execution control was 1 in one of the following cases: ▪ After guest software executed MOV to CR8 (see Section 21.1.3). ▪ As part of a TPR-shadow update (see Section 21.5.3.3). ▪ After VM entry with the 1-setting of the "virtualize APIC accesses" VM-execution control (see Section 22.6.7).

Table I-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
44	**APIC access.** Guest software attempted to access memory at a physical address on the APIC-access page and the "virtualize APIC accesses" VM-execution control was 1 (see Section 21.2).
46	**Access to GDTR or IDTR.** Guest software attempted to execute LGDT, LIDT, SGDT, or SIDT and the "descriptor-table exiting" VM-execution control was 1.
47	**Access to LDTR or TR.** Guest software attempted to execute LLDT, LTR, SLDT, or STR and the "descriptor-table exiting" VM-execution control was 1.
48	**EPT violation.** An attempt to access memory with a guest-physical address was disallowed by the configuration of the EPT paging structures.
49	**EPT misconfiguration.** An attempt to access memory with a guest-physical address encountered a misconfigured EPT paging-structure entry.
50	**INVEPT.** Guest software attempted to execute INVEPT.
52	**VMX-preemption timer expired.** The preemption timer counted down to zero.
53	**INVVPID.** Guest software attempted to execute INVVPID.
54	**WBINVD.** Guest software attempted to execute WBINVD and the "WBINVD exiting" VM-execution control was 1.
55	**XSETBV.** Guest software attempted to execute XSETBV.

INDEX FOR VOLUMES 3A & 3B