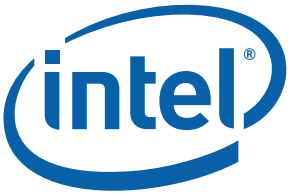

Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

June 2009

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

Document Number: 252046-024

Contents

Revision History

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

Revision	Description	Date
-022	• Removed Documentation Changes 1-6 • Added Documentation Changes 1-6	August 2008
-023	• Removed Documentation Changes 1-6 • Added Documentation Changes 1-21	March 2009
-024	• Removed Documentation Changes 1-21 • Added Documentation Changes 1-16	June 2009

§

Preface

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

Affected Documents

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

Nomenclature

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

Summary Tables of Changes

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

Codes Used in Summary Tables

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

Documentation Changes

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 3, Volume 2A
2	Updates to Chapter 5, Volume 2B
3	Updates to Chapter 2, Volume 3A
4	Updates to Chapter 3, Volume 3A
5	New Chapter 4 added to Volume 3A
6	Updates to Chapter 5, Volume 3A
7	Updates to Chapter 6, Volume 3A
8	Updates to Chapter 9, Volume 3A
9	Updates to Chapter 11, Volume 3A
10	Updates to Chapter 15, Volume 3A
11	Updates to Chapter 18, Volume 3A
12	Updates to Chapter 21, Volume 3B
13	Updates to Chapter 22, Volume 3B
14	Updates to Chapter 23, Volume 3B
15	Updates to Chapter 24, Volume 3B
16	Updates to Chapter 29, Volume 3B

Documentation Changes

1. Updates to Chapter 3, Volume 2A

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

...

CPUID—CPU Identification

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
0F A2	CPUID	Valid	Valid	Returns processor identification and feature information to the EAX, EBX, ECX, and EDX registers, as determined by input entered in EAX (in some cases, ECX as well).

Description

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction. This instruction operates the same in non-64-bit modes and 64-bit mode.

CPUID returns processor identification and feature information in the EAX, EBX, ECX, and EDX registers. The instruction's output is dependent on the contents of the EAX register upon execution (in some cases, ECX as well). For example, the following pseudocode loads EAX with 00H and causes CPUID to return a Maximum Return Value and the Vendor Identification String in the appropriate registers:

```
MOV EAX, 00H
CPUID
```

Table Table 3-20. shows information returned, depending on the initial value loaded into the EAX register. Table 3-21 shows the maximum CPUID input value recognized for each family of IA-32 processors on which CPUID is implemented.

Two types of information are returned: basic and extended function information. If a value entered for CPUID.EAX is invalid for a particular processor, the data for the highest basic information leaf is returned. For example, using the Intel Core i7 processor, the following is true:

```
CPUID.EAX = 05H (* Returns MONITOR/MWAIT leaf. *)
CPUID.EAX = 0AH (* Returns Architectural Performance Monitoring leaf. *)
CPUID.EAX = 0BH (* Returns Extended Topology Enumeration leaf. *)
CPUID.EAX = 0CH (* INVALID: Returns the same information as CPUID.EAX = 0BH. *)
CPUID.EAX = 80000008H (* Returns linear/physical address size data. *)
CPUID.EAX = 8000000AH (* INVALID: Returns same information as CPUID.EAX = 0BH. *)
```

When CPUID returns the highest basic leaf information as a result of an invalid input EAX value, any dependence on input ECX value in the basic leaf is honored.

Table 3-20. Information Returned by CPUID Instruction (Continued)

	EDX	Bits 10-0: Reserved Bit 11: SYSCALL/SYSRET available (when in 64-bit mode) Bits 19-12: Reserved = 0 Bit 20: Execute Disable Bit available Bits 26-21: Reserved = 0 Bit 27: RDTSCP and IA32_TSC_AUX are available if 1 Bits 28: Reserved = 0 Bit 29: Intel® 64 Architecture available if 1 Bits 31-30: Reserved = 0
80000002H	EAX EBX ECX EDX	Processor Brand String Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000003H	EAX EBX ECX EDX	Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000004H	EAX EBX ECX EDX	Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued Processor Brand String Continued
80000005H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Reserved = 0
80000006H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Bits 7-0: Cache Line size in bytes Bits 15-12: L2 Associativity field * Bits 31-16: Cache size in 1K units Reserved = 0
		NOTES: * L2 associativity field encodings: 00H - Disabled 01H - Direct mapped 02H - 2-way 04H - 4-way 06H - 8-way 08H - 16-way 0FH - Fully associative
80000007H	EAX EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 Bits 7-0: Reserved = 0 Bit 8: Invariant TSC available if 1 Bits 31-9: Reserved = 0

80000008H	EAX	Linear/Physical Address size Bits 7-0: #Physical Address Bits* Bits 15-8: #Linear Address Bits Bits 31-16: Reserved = 0
	EBX ECX EDX	Reserved = 0 Reserved = 0 Reserved = 0 **NOTES:** * If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field.

...

Table 3-24. More on Feature Information Returned in the EDX Register (Continued)

Bit #	Mnemonic	Description
0	FPU	**Floating Point Unit On-Chip.** The processor contains an x87 FPU.
1	VME	**Virtual 8086 Mode Enhancements.** Virtual 8086 mode enhancements, including CR4.VME for controlling the feature, CR4.PVI for protected mode virtual interrupts, software interrupt indirection, expansion of the TSS with the software indirection bitmap, and EFLAGS.VIF and EFLAGS.VIP flags.
2	DE	**Debugging Extensions.** Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.
3	PSE	**Page Size Extension.** Large pages of size 4 MByte are supported, including CR4.PSE for controlling the feature, the defined dirty bit in PDE (Page Directory Entries), optional reserved bit trapping in CR3, PDEs, and PTEs.
4	TSC	**Time Stamp Counter.** The RDTSC instruction is supported, including CR4.TSD for controlling privilege.
5	MSR	**Model Specific Registers RDMSR and WRMSR Instructions.** The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.
6	PAE	**Physical Address Extension.** Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2-MByte pages are supported instead of 4 Mbyte pages if PAE bit is 1.
7	MCE	**Machine Check Exception**. Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.
8	CX8	**CMPXCHG8B Instruction.** The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).

Bit #	Mnemonic	Description
9	APIC	**APIC On-Chip.** The processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default - some processors permit the APIC to be relocated).
10	Reserved	Reserved
11	SEP	**SYSENTER and SYSEXIT Instructions.** The SYSENTER and SYSEXIT and associated MSRs are supported.
12	MTRR	**Memory Type Range Registers.** MTRRs are supported. The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.
13	PGE	**Page Global Bit.** The global bit is supported in paging-structure entries (PDEs and PTEs) that map a page, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.
14	MCA	**Machine Check Architecture.** The Machine Check Architecture, which provides a compatible mechanism for error reporting in P6 family, Pentium 4, Intel Xeon processors, and future processors, is supported. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.
15	CMOV	**Conditional Move Instructions.** The conditional move instruction CMOV is supported. In addition, if x87 FPU is present as indicated by the CPUID.FPU feature bit, then the FCOMI and FCMOV instructions are supported
16	PAT	**Page Attribute Table.** Page Attribute Table is supported. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory accessed through a linear address on a 4KB granularity.
17	PSE-36	**36-Bit Page Size Extension.** 4-MByte pages addressing physical memory beyond 4 GBytes are supported with 32-bit paging. This feature indicates that upper bits of the physical address of a 4-MByte page are encoded in bits 20:13 of the page-directory entry. Such physical addresses are limited by MAXPHYADDR and may be up to 40 bits in size.
18	PSN	**Processor Serial Number.** The processor supports the 96-bit processor identification number feature and the feature is enabled.
19	CLFSH	**CLFLUSH Instruction.** CLFLUSH Instruction is supported.
20	Reserved	Reserved
21	DS	**Debug Store.** The processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and precise event-based sampling (PEBS) facilities (see Chapter 19, "Debugging and Performance Monitoring," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*).
22	ACPI	**Thermal Monitor and Software Controlled Clock Facilities.** The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	**Intel MMX Technology.** The processor supports the Intel MMX technology.

Bit #	Mnemonic	Description
24	FXSR	**FXSAVE and FXRSTOR Instructions.** The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions.
25	SSE	**SSE.** The processor supports the SSE extensions.
26	SSE2	**SSE2.** The processor supports the SSE2 extensions.
27	SS	**Self Snoop.** The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	**Multi-Threading.** The physical processor package is capable of supporting more than one logical processor.
29	TM	**Thermal Monitor.** The processor implements the thermal monitor automatic thermal control circuitry (TCC).
30	Reserved	Reserved
31	PBE	**Pending Break Enable.** The processor supports the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt. Bit 10 (PBE enable) in the IA32_MISC_ENABLE MSR enables this capability.

...

IA-32 Architecture Compatibility

CPUID is not supported in early models of the Intel486 processor or in any IA-32 processor earlier than the Intel486 processor.

Operation

```
IA32_BIOS_SIGN_ID MSR ← Update with installed microcode revision number;

CASE (EAX) OF
    EAX = 0:
        EAX ← Highest basic function input value understood by CPUID;
        EBX ← Vendor identification string;
        EDX ← Vendor identification string;
        ECX ← Vendor identification string;
    BREAK;
    EAX = 1H:
        EAX[3:0] ← Stepping ID;
        EAX[7:4] ← Model;
        EAX[11:8] ← Family;
        EAX[13:12] ← Processor type;
        EAX[15:14] ← Reserved;
        EAX[19:16] ← Extended Model;
        EAX[27:20] ← Extended Family;
        EAX[31:28] ← Reserved;
```

```
        EBX[7:0] ← Brand Index; (* Reserved if the value is zero. *)
        EBX[15:8] ← CLFLUSH Line Size;
        EBX[16:23] ← Reserved; (* Number of threads enabled = 2 if MT enable fuse set. *)
        EBX[24:31] ← Initial APIC ID;
        ECX ← Feature flags; (* See Figure 3-7. *)
        EDX ← Feature flags; (* See Figure 3-8. *)
BREAK;
EAX = 2H:
        EAX ← Cache and TLB information;
        EBX ← Cache and TLB information;
        ECX ← Cache and TLB information;
        EDX ← Cache and TLB information;
BREAK;
EAX = 3H:
        EAX ← Reserved;
        EBX ← Reserved;
        ECX ← ProcessorSerialNumber[31:0];
        (* Pentium lll processors only, otherwise reserved. *)
        EDX ← ProcessorSerialNumber[63:32];
        (* Pentium lll processors only, otherwise reserved. *
BREAK
EAX = 4H:
        EAX ← Deterministic Cache Parameters Leaf; (* See Table Table 3-20.. *)
        EBX ← Deterministic Cache Parameters Leaf;
        ECX ← Deterministic Cache Parameters Leaf;
        EDX ← Deterministic Cache Parameters Leaf;
BREAK;
EAX = 5H:
        EAX ← MONITOR/MWAIT Leaf; (* See Table Table 3-20.. *)
        EBX ← MONITOR/MWAIT Leaf;
        ECX ← MONITOR/MWAIT Leaf;
        EDX ← MONITOR/MWAIT Leaf;
BREAK;
EAX = 6H:
        EAX ← Thermal and Power Management Leaf; (* See Table Table 3-20.. *)
        EBX ← Thermal and Power Management Leaf;
        ECX ← Thermal and Power Management Leaf;
        EDX ← Thermal and Power Management Leaf;
BREAK;
EAX = 7H or 8H:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
        EDX ← Reserved = 0;
BREAK;
EAX = 9H:
        EAX ← Direct Cache Access Information Leaf; (* See Table Table 3-20.. *)
        EBX ← Direct Cache Access Information Leaf;
        ECX ← Direct Cache Access Information Leaf;
        EDX ← Direct Cache Access Information Leaf;
```

```
    BREAK;
    EAX = AH:
        EAX ← Architectural Performance Monitoring Leaf; (* See Table Table 3-20.. *)
        EBX ← Architectural Performance Monitoring Leaf;
        ECX ← Architectural Performance Monitoring Leaf;
        EDX ← Architectural Performance Monitoring Leaf;
        BREAK
    EAX = BH:
        EAX ← Extended Topology Enumeration Leaf; (* See Table Table 3-20.. *)
        EBX ← Extended Topology Enumeration Leaf;
        ECX ← Extended Topology Enumeration Leaf;
        EDX ← Extended Topology Enumeration Leaf;
    BREAK;
EAX = CH:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
        EDX ← Reserved = 0;
    BREAK;
    EAX = DH:
        EAX ← Processor Extended State Enumeration Leaf; (* See Table Table 3-20.. *)
        EBX ← Processor Extended State Enumeration Leaf;
        ECX ← Processor Extended State Enumeration Leaf;
        EDX ← Processor Extended State Enumeration Leaf;
    BREAK;
BREAK;
    EAX = 80000000H:
        EAX ← Highest extended function input value understood by CPUID;
        EBX ← Reserved;
        ECX ← Reserved;
        EDX ← Reserved;
    BREAK;
    EAX = 80000001H:
        EAX ← Reserved;
        EBX ← Reserved;
        ECX ← Extended Feature Bits (* See Table Table 3-20..*);
        EDX ← Extended Feature Bits (* See Table Table 3-20.. *);
    BREAK;
    EAX = 80000002H:
        EAX ← Processor Brand String;
        EBX ← Processor Brand String, continued;
        ECX ← Processor Brand String, continued;
        EDX ← Processor Brand String, continued;
    BREAK;
    EAX = 80000003H:
        EAX ← Processor Brand String, continued;
        EBX ← Processor Brand String, continued;
        ECX ← Processor Brand String, continued;
        EDX ← Processor Brand String, continued;
    BREAK;
```

```
    EAX = 80000004H:
        EAX ← Processor Brand String, continued;
        EBX ← Processor Brand String, continued;
        ECX ← Processor Brand String, continued;
        EDX ← Processor Brand String, continued;
    BREAK;
    EAX = 80000005H:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
        EDX ← Reserved = 0;
    BREAK;
    EAX = 80000006H:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Cache information;
        EDX ← Reserved = 0;
    BREAK;
    EAX = 80000007H:
        EAX ← Reserved = 0;
        EBX ← Reserved = 0;
        ECX ← Reserved = 0;
        EDX ← Reserved = Misc Feature Flags;
    BREAK;
    EAX = 80000008H:
        EAX ← Reserved = Physical Address Size Information;
        EBX ← Reserved = Virtual Address Size Information;
        ECX ← Reserved = 0;
        EDX ← Reserved = 0;
    BREAK;
    EAX >= 40000000H and EAX <= 4FFFFFFFH:
    DEFAULT: (* EAX = Value outside of recognized range for CPUID. *)
        (* If the highest basic information leaf data depend on ECX input value, ECX is honored.*)
        EAX ← Reserved; (* Information returned for highest basic information leaf. *)
        EBX ← Reserved; (* Information returned for highest basic information leaf. *)
        ECX ← Reserved; (* Information returned for highest basic information leaf. *)
        EDX ← Reserved; (* Information returned for highest basic information leaf. *)
    BREAK;
ESAC;

...
```

2. Updates to Chapter 5, Volume 2B

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: xxxx*.

INVVPID— Invalidate Translations Based on VPID

...

- Single-context invalidation, retaining global translations: If the INVVPID type is 3, the logical processor invalidates all mappings tagged with the VPID specified in the INVVPID descriptor except global translations. In some cases, it may invalidate global translations (and mappings with other VPIDs) as well. See Section 4.10, "Caching Translation Information," in *IA-32 Intel Architecture Software Developer's Manual, Volumes 3A* for information about global translations.

...

3. Updates to Chapter 2, Volume 3A

Change bars show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

2.1.3 Task-State Segments and Task Gates

The TSS (see Figure 2-1) defines the state of the execution environment for a task. It includes the state of general-purpose registers, segment registers, the EFLAGS register, the EIP register, and segment selectors with stack pointers for three stack segments (one stack for each privilege level). The TSS also includes the segment selector for the LDT associated with the task and the base address of the paging-structure hierarchy.

All program execution in protected mode happens within the context of a task (called the current task). The segment selector for the TSS for the current task is stored in the task register. The simplest method for switching to a task is to make a call or jump to the new task. Here, the segment selector for the TSS of the new task is given in the CALL or JMP instruction. In switching tasks, the processor performs the following actions:

1. Stores the state of the current task in the current TSS.
2. Loads the task register with the segment selector for the new task.
3. Accesses the new TSS through a segment descriptor in the GDT.
4. Loads the state of the new task from the new TSS into the general-purpose registers, the segment registers, the LDTR, control register CR3 (base address of the paging-structure hierarchy), the EFLAGS register, and the EIP register.
5. Begins execution of the new task.

A task can also be accessed through a task gate. A task gate is similar to a call gate, except that it provides access (through a segment selector) to a TSS rather than a code segment.

2.1.5 Memory Management

System architecture supports either direct physical addressing of memory or virtual memory (through paging). When physical addressing is used, a linear address is treated as a physical address. When paging is used: all code, data, stack, and system segments (including the GDT and IDT) can be paged with only the most recently accessed pages being held in physical memory.

The location of pages (sometimes called page frames) in physical memory is contained in the paging structures. These structures reside in physical memory (see Figure 2-1 for the case of 32-bit paging).

The base physical address of the paging-structure hierarchy is contained in control register CR3. The entries in the paging structures determine the physical address of the base of a page frame, access rights and memory management information.

To use this paging mechanism, a linear address is broken into parts. The parts provide separate offsets into the paging structures and the page frame. A system can have a single hierarchy of paging structures or several. For example, each task can have its own hierarchy.

2.1.5.1 Memory Management in IA-32e Mode

In IA-32e mode, physical memory pages are managed by a set of system data structures. In compatibility mode and 64-bit mode, four levels of system data structures are used. These include:

- **The page map level 4 (PML4)** — An entry in a PML4 table contains the physical address of the base of a page directory pointer table, access rights, and memory management information. The base physical address of the PML4 is stored in CR3.
- **A set of page directory pointer tables** — An entry in a page directory pointer table contains the physical address of the base of a page directory table, access rights, and memory management information.
- **Sets of page directories** — An entry in a page directory table contains the physical address of the base of a page table, access rights, and memory management information.
- **Sets of page tables** — An entry in a page table contains the physical address of a page frame, access rights, and memory management information.

2.1.6 System Registers

To assist in initializing the processor and controlling system operations, the system architecture provides system flags in the EFLAGS register and several system registers:

- The system flags and IOPL field in the EFLAGS register control task and mode switching, interrupt handling, instruction tracing, and access rights. See also: Section 2.3, “System Flags and Fields in the EFLAGS Register.”
- The control registers (CR0, CR2, CR3, and CR4) contain a variety of flags and data fields for controlling system-level operations. Other flags in these registers are used to indicate support for specific processor capabilities within the operating system or executive. See also: Section 2.5, “Control Registers.”
- The debug registers (not shown in Figure 2-1) allow the setting of breakpoints for use in debugging programs and systems software. See also: Chapter 19, “Debugging and Performance Monitoring.”
- The GDTR, LDTR, and IDTR registers contain the linear addresses and sizes (limits) of their respective tables. See also: Section 2.4, “Memory-Management Registers.”
- The task register contains the linear address and size of the TSS for the current task. See also: Section 2.4, “Memory-Management Registers.”
- Model-specific registers (not shown in Figure 2-1).

The model-specific registers (MSRs) are a group of registers available primarily to operating-system or executive procedures (that is, code running at privilege level 0). These registers control items such as the debug extensions, the performance-monitoring counters, the machine- check architecture, and the memory type ranges (MTRRs).

The number and function of these registers varies among different members of the Intel 64 and IA-32 processor families. See also: Section 9.4, "Model-Specific Registers (MSRs)," and Appendix B, "Model-Specific Registers (MSRs)."

Most systems restrict access to system registers (other than the EFLAGS register) by application programs. Systems can be designed, however, where all programs and procedures run at the most privileged level (privilege level 0). In such a case, application programs would be allowed to modify the system registers.

...

2.5 CONTROL REGISTERS

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

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

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

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

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

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

See also: Chapter 4, “Paging.”

- **CR4** — Contains a group of flags that enable several architectural extensions, and indicate operating system or executive support for specific processor capabilities. The control registers can be read and loaded (or modified) using the move-to-or-from-control-registers forms of the MOV instruction. In protected mode, the MOV instructions allow the control registers to be read or loaded (at privilege level 0 only). This restriction means that application programs or operating-system procedures (running at privilege levels 1, 2, or 3) are prevented from reading or loading the control registers.
- **CR8** — Provides read and write access to the Task Priority Register (TPR). It specifies the priority threshold value that operating systems use to control the priority class of external interrupts allowed to interrupt the processor. This register is available only in 64-bit mode. However, interrupt filtering continues to apply in compatibility mode.

...

8086 mode.

WP **Write Protect (bit 16 of CR0)** — Inhibits supervisor-level procedures from writing into user-level read-only pages when set; allows supervisor-level procedures to write into user-level read-only pages when clear (regardless of the U/S bit setting; see Section 4.1.3 and Section 4.6). This flag facilitates implementation of the copy-on-write method of creating a new process (forking) used by operating systems such as UNIX.

...

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

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

See also: Section 9.9, “Mode Switching.”

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

See also: Chapter 11, “Memory Cache Control” (for more about the use of the PCD flag) and Section 4.9, “Paging and Memory Typing” (for a discussion of a companion PCD flag in page-directory and page-table entries).

PWT **Page-level Write-Through (bit 3 of CR3)** — Controls the write-through or write-back caching policy of the first paging structure of the current paging-structure hierarchy. When the PWT flag is set, write-through caching is enabled; when the flag is clear, write-back caching is enabled. This flag affects only internal caches (both L1 and L2, when present). The processor ignores this flag

if paging is not used (the PG flag in register CR0 is clear) or the CD (cache disable) flag in CR0 is set.

See also: Section 11.5, "Cache Control" (for more information about the use of this flag), and Section 4.9, "Paging and Memory Typing" (for a discussion of a companion PCD flag in the page-directory and page-table entries).

...

PSE **Page Size Extensions (bit 4 of CR4)** — Enables 4-MByte pages with 32-bit paging when set; restricts 32-bit paging to pages to 4 KBytes when clear.

See also: Section 4.3, "32-Bit Paging."

PAE **Physical Address Extension (bit 5 of CR4)** — When set, enables paging to produce physical addresses with more than 32 bits. When clear, restricts physical addresses to 32 bits. PAE must be set before entering IA-32e mode.

See also: Chapter 4, "Paging."

4. Updates to Chapter 3, Volume 3A

Change bars show changes to Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

3.3 PHYSICAL ADDRESS SPACE

In protected mode, the IA-32 architecture provides a normal physical address space of 4 GBytes (2^{32} bytes). This is the address space that the processor can address on its address bus. This address space is flat (unsegmented), with addresses ranging continuously from 0 to FFFFFFFFH. This physical address space can be mapped to read-write memory, read-only memory, and memory mapped I/O. The memory mapping facilities described in this chapter can be used to divide this physical memory up into segments and/or pages.

Starting with the Pentium Pro processor, the IA-32 architecture also supports an extension of the physical address space to 2^{36} bytes (64 GBytes); with a maximum physical address of FFFFFFFFFH. This extension is invoked in either of two ways:

- Using the physical address extension (PAE) flag, located in bit 5 of control register CR4.
- Using the 36-bit page size extension (PSE-36) feature (introduced in the Pentium III processors).

Physical address support has since been extended beyond 36 bits. See Chapter 4, "Paging" for more information about 36-bit physical addressing.

...

5. New Chapter 4 added to Volume 3A

New Chapter 4 has been added to the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

CHAPTER 4
PAGING

Chapter 3 explains how segmentation converts logical addresses to linear addresses. **Paging** (or linear-address translation) is the process of translating linear addresses so that they can be used to access memory or I/O devices. Paging translates each linear address to a **physical address** and determines, for each translation, what accesses to the linear address are allowed (the address's **access rights**) and the type of caching used for such accesses (the address's **memory type**).

Intel-64 processors support three different paging modes. These modes are identified and defined in Section 4.1. Section 4.2 gives an overview of the translation mechanism that is used in all modes. Section 4.3, Section 4.4, and Section 4.5 discuss the three paging modes in detail.

Section 4.6 details how paging determines and uses access rights. Section 4.7 discusses exceptions that may be generated by paging (page-fault exceptions). Section 4.8 considers data which the processor writes in response to linear-address accesses (accessed and dirty flags).

Section 4.9 describes how paging determines the memory types used for accesses to linear addresses. Section 4.10 provides details of how a processor may cache information about linear-address translation. Section 4.11 outlines interactions between paging and certain VMX features. Section 4.12 gives an overview of how paging can be used to implement virtual memory.

4.1 PAGING MODES AND CONTROL BITS

Paging behavior is controlled by the following control bits:

- The WP and PG flags in control register CR0 (bit 16 and bit 31, respectively).
- The PSE, PAE, and PGE flags in control register CR4 (bit 4, bit 5, and bit 7, respectively).
- The LME and NXE flags in the IA32_EFER MSR (bit 8 and bit 11, respectively).

Software enables paging by using the MOV to CR0 instruction to set CR0.PG. Before dong so, software should ensure that control register CR3 contains the physical address of the first paging structure that the processor will use for linear-address translation (see Section 4.2) and that that structure is initialized as desired. See Table Table 4-3., Table Table 4-7., and Table Table 4-12. for the use of CR3 in the different paging modes.

Section 4.1.1 describes how the values of CR0.PG, CR4.PAE, and IA32_EFER.LME determine whether paging is in use and, if so, which of three paging modes is in use. Section 4.1.2 explains how to manage these bits to establish or make changes in paging modes. Section 4.1.3 discusses how CR0.WP, CR4.PSE, CR4.PGE, and IA32_EFER.NXE modify the operation of the different paging modes.

4.1.1 Three Paging Modes

If CR0.PG = 0, paging is not used. The logical processor treats all linear addresses as if they were physical addresses. CR4.PAE and IA32_EFER.LME are ignored by the processor, as are CR0.WP, CR4.PSE, and CR4.PGE, and IA32_EFER.NXE.

Paging is enabled if CR0.PG = 1. Paging can be enabled only if protection is enabled (CR0.PE = 1). If paging is enabled, one of three paging modes is used. The values of CR4.PAE and IA32_EFER.LME determine which paging mode is used:

- If CR0.PG = 1 and CR4.PAE = 0, **32-bit paging** is used. 32-bit paging is detailed in Section 4.3. 32-bit paging uses CR0.WP, CR4.PSE, and CR4.PGE as described in Section 4.1.3.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0, **PAE paging** is used. PAE paging is detailed in Section 4.4. PAE paging uses CR0.WP, CR4.PGE, and IA32_EFER.NXE as described in Section 4.1.3.
- If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 1, **IA-32e paging** is used.[1] IA-32e paging is detailed in Section 4.5. IA-32e paging uses CR0.WP, CR4.PGE, and IA32_EFER.NXE as described in Section 4.1.3. IA-32e paging is available only on processors that support the Intel 64 architecture.

The three paging modes differ with regard to the following details:

- Linear-address width. The size of the linear addresses that can be translated.
- Physical-address width. The size of the physical addresses produced by paging.
- Page size. The granularity at which linear addresses are translated. Linear addresses on the same page are translated to corresponding physical addresses on the same page.
- Support for execute-disable access rights. In some paging modes, software can be prevented from fetching instructions from pages that are otherwise readable.

Table Table 4-1. illustrates the key differences between the three paging modes.

Table 4-1. Properties of Different Paging Modes

Paging Mode	CR0.PG	CR4.PAE	LME in IA32_EFER	Linear-Address Width	Physical-Address Width[1]	Page Size(s)	Supports Execute-Disable?
None	0	N/A	N/A	32	32	N/A	No
32-bit	1	0	0[2]	32	Up to 40[3]	4-KByte 4-MByte[4]	No
PAE	1	1	0	32	Up to 52	4-KByte 2-MByte	Yes[5]
IA-32e	1	1	2	48	Up to 52	4-KByte 2-MByte	Yes[5]

NOTES:

1. The physical-address width is always bounded by MAXPHYADDR; see Section 4.1.4.
2. The processor ensures that IA32_EFER.LME must be 0 if CR0.PG = 1 and CR4.PAE = 0.
3. 32-bit paging supports physical-address widths of more than 32 bits only for 4-MByte pages and only if the PSE-36 mechanism is supported; see Section 4.1.4 and Section 4.3.
4. 4-MByte pages are used with 32-bit paging only if CR4.PSE = 1; see Section 4.3.
5. Execute-disable access rights are applied only if IA32_EFER.NXE = 1; see Section 4.6.

---

1. The LMA flag in the IA32_EFER MSR (bit 10) is a status bit that indicates whether the logical processor is in IA-32e mode (and thus using IA-32e paging). The processor always sets IA32_EFER.LMA to CR0.PG & IA32_EFER.LME. Software cannot directly modify IA32_EFER.LMA; an execution of WRMSR to the IA32_EFER MSR ignores bit 10 of its source operand.

Because they are used only if IA32_EFER.LME = 0, 32-bit paging and PAE paging is used only in legacy protected mode. Because legacy protected mode cannot produce linear addresses larger than 32 bits, 32-bit paging and PAE paging translate 32-bit linear addresses.

Because it is used only if IA32_EFER.LME = 1, IA-32e paging is used only in IA-32e mode. (In fact, it is the use of IA-32e paging that defines IA-32 mode.) IA-32 mode has two sub-modes:

- Compatibility mode. This mode uses only 32-bit linear addresses. IA-32e paging treats bits 47:32 of such an address as all 0.
- 64-bit mode. While this mode produces 64-bit linear addresses, the processor ensures that bits 63:47 of such an address are identical.[1] IA-32e paging does not use bits 63:48 of such addresses.

4.1.2 Paging-Mode Enabling

If CR0.PG = 1, a logical processor is in one of three paging modes, depending on the values of CR4.PAE and IA32_EFER.LME. Figure 4-1 illustrates how software can enable these modes and make transitions between them. The following items identify certain limitations and other details:

- IA32_EFER.LME cannot be modified while paging is enabled (CR0.PG = 1). Attempts to do so using WRMSR cause a general-protection exception (#GP(0)).
- Paging cannot be enabled (by setting CR0.PG to 1) while CR4.PAE = 0 and IA32_EFER.LME = 1. Attempts to do so using MOV to CR0 cause a general-protection exception (#GP(0)).
- CR4.PAE cannot be cleared while IA-32e paging is active (CR0.PG = 1 and IA32_EFER.LME = 1). Attempts to do so using MOV to CR4 cause a general-protection exception (#GP(0)).
- Software can always disable paging by clearing CR0.PG with MOV to CR0.
- Software can make transitions between 32-bit paging and PAE paging by changing the value of CR4.PAE with MOV to CR4.
- Software cannot make transitions directly between IA-32e paging and either of the other two paging modes. It must first disable paging (by clearing CR0.PG with MOV to CR0), then set CR4.PAE and IA32_EFER.LME to the desired values (with MOV to CR4 and WRMSR), and then re-enable paging (by setting CR0.PG with MOV to CR0). As noted earlier, an attempt to clear either CR4.PAE or IA32_EFER.LME cause a general-protection exception (#GP(0)).
- VMX transitions allow transitions between paging modes that are not possible using MOV to CR or WRMSR. This is because VMX transitions can load CR0, CR4, and IA32_EFER in one operation. See Section 4.11.1.

4.1.3 Paging-Mode Modifiers

Details of how each paging mode operates are determined by the following control bits:

- The WP flag in CR0 (bit 16).

1. Such an address is called **canonical**. Use of a non-canonical linear address in 64-bit mode produces a general-protection exception (#GP(0)); the processor does not attempt to translate non-canonical linear addresses using IA-32e paging.

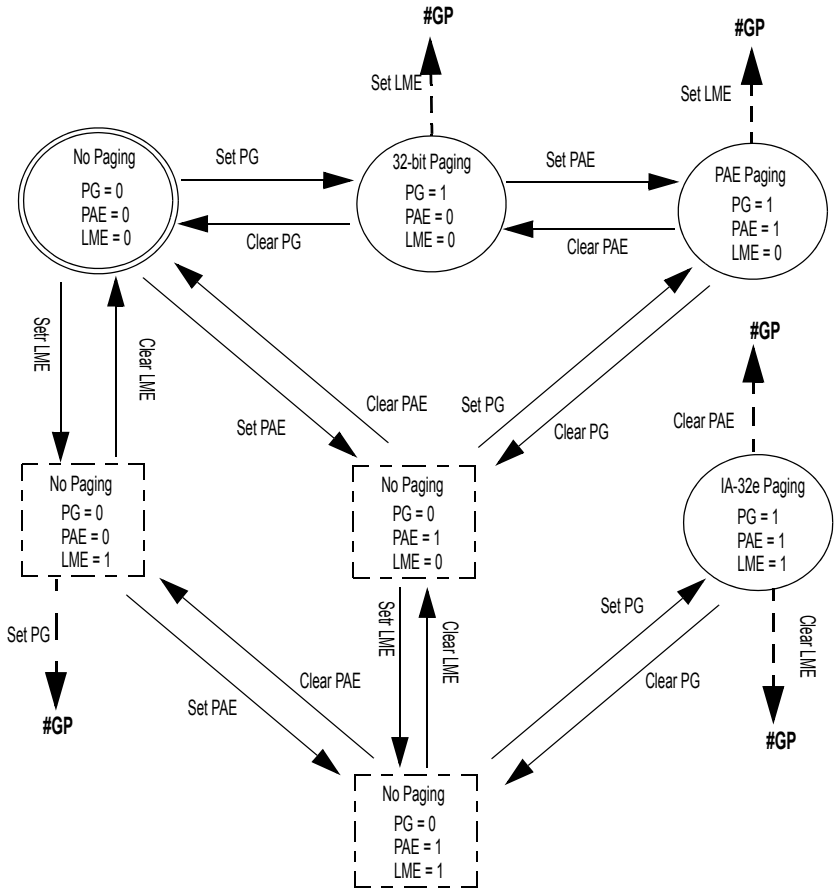

Figure 4-1 Enabling and Changing Paging Modes

- The PSE and PGE flags in CR4 (bit 4 and bit 7, respectively).
- The NXE flag in the IA32_EFER MSR (bit 11).

CR0.WP allows pages to be protected from supervisor-mode writes. If CR0.WP = 0, software operating with CPL < 3 (supervisor mode) can write to linear addresses with read-only access rights; if CR0.WP = 1, it cannot. (Software operating with CPL = 3 — user mode — cannot write to linear addresses with read-only access rights, regardless of the value of CR0.WP.) Section 4.6 explains how access rights are determined.

CR4.PSE enables 4-MByte pages for 32-bit paging. If CR4.PSE = 0, 32-bit paging can use only 4-KByte pages; if CR4.PSE = 1, 32-bit paging can use both 4-KByte pages and 4-MByte pages. See Section 4.3 for more information. (PAE paging and IA-32e paging can use multiple page sizes regardless of the value of CR4.PSE.)

CR4.PGE enables global pages. If CR4.PGE = 0, no translations are shared across address spaces; if CR4.PGE = 1, specified translations may be shared across address spaces. See Section 4.10.1.4 for more information.

IA32_EFER.NXE enables execute-disable access rights for PAE paging and IA-32e paging. If IA32_EFER.NXE = 0, software may fetch instructions from any linear address

that paging allows the software to read; if IA32_EFER.NXE = 1, instructions fetches can be prevented from specified linear addresses (even if data reads from the addresses are allowed). Section 4.6 explains how access rights are determined. (32-bit paging always allows software to fetch instructions from any linear address that may be read; IA32_EFER.NXE has no effect with 32-bit paging. Software that wants to limit instruction fetches from readable pages must use either PAE paging or IA-32e paging.)

4.1.4 Enumeration of Paging Features by CPUID

Software can discover support for different paging features using the CPUID instruction:

- PSE: page-size extensions for 32-bit paging.
  If CPUID.01H:EDX.PSE [bit 3] = 1, CR4.PSE may be set to 1, enabling support for 4-MByte pages with 32-bit paging (see Section 4.3).
- PAE: physical-address extension.
  If CPUID.01H:EDX.PAE [bit 6] = 1, CR4.PAE may be set to 1, enabling PAE paging (this setting is also required for IA-32e paging).
- PGE: global-page support.
  If CPUID.01H:EDX.PGE [bit 13] = 1, CR4.PGE may be set to 1, enabling the global-page feature (see Section 4.10.1.4).
- PAT: page-attribute table.
  If CPUID.01H:EDX.PAT [bit 16] = 1, the 8-entry page-attribute table (PAT) is supported. When the PAT is supported, three bits in certain paging-structure entries select a memory type (used to determine type of caching used) from the PAT (see Section 4.9).
- PSE-36: 36-Bit page size extension.
  If CPUID.01H:EDX.PSE-36 [bit 17] = 1, the PSE-36 mechanism is supported, indicating that translations using 4-MByte pages with 32-bit paging may produce physical addresses with more than 32 bits (see Section 4.3).
- NX: execute disable.
  If CPUID.80000001H:EDX.NX [bit 20] = 1, IA32_EFER.NXE may be set to 1, allowing PAE paging and IA-32e paging to disable execute access to selected pages (see Section 4.6). (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.NXE to be set to 1.)
- LM: IA-32e mode support.
  If CPUID.80000001H:EDX.LM [bit 29] = 1, IA32_EFER.LME may be set to 1, enabling IA-32e paging. (Processors that do not support CPUID function 80000001H do not allow IA32_EFER.LME to be set to 1.)
- CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. (For processors that do not support CPUID function 80000008H, the width is generally 36 if CPUID.01H:EDX.PAE [bit 6] = 1 and 32 otherwise.) This width is referred to as MAXPHYADDR. MAXPHYADDR is at most 52.
- CPUID.80000008H:EAX[15:8] reports the linear-address width supported by the processor. Generally, this value is 48 if CPUID.80000001H:EDX.LM [bit 29] = 1 and 32 otherwise. (Processors that do not support CPUID function 80000008H, support a linear-address width of 32.)

4.2 HIERARCHICAL PAGING STRUCTURES: AN OVERVIEW

All three paging modes translate linear addresses use **hierarchical paging structures**. This section provides an overview of their operation. Section 4.3, Section 4.4, and Section 4.5 provide details for the three paging modes.

Every paging structure is 4096 Bytes in size and comprises a number of individual **entries**. With 32-bit paging, each entry is 32 bits (4 bytes); there are thus 1024 entries in each structure. With PAE paging and IA-32e paging, each entry is 64 bits (8 bytes); there are thus 512 entries in each structure. (PAE paging includes one exception, a paging structure that is 32 bytes in size, containing 4 64-bit entries.)

The processor uses the upper portion of a linear address to identify a series of paging-structure entries. The last of these entries identifies the physical address of the region to which the linear address translates (called the **page frame**). The lower portion of the linear address (called the **page offset**) identifies the specific address within that region to which the linear address translates.

Each paging-structure entry contains a physical address, which is either the address of another paging structure or the address of a page frame. In the first case, the entry is said to **reference** the other paging structure; in the latter, the entry is said to **map a page**.

The first paging structure used for any translation is located at the physical address in CR3. A linear address is translated using the following iterative procedure. A portion of the linear address (initially the uppermost bits) select an entry in a paging structure (initially the one located using CR3). If that entry references another paging structure, the process continues with that paging structure and with the portion of the linear address immediately below that just used. If instead the entry maps a page, the process completes: the physical address in the entry is that of the page frame and the remaining lower portion of the linear address is the page offset.

The following items give an example for each of the three paging modes (each example locates a 4-KByte page frame):

- With 32-bit paging, each paging structure comprises $1024 = 2^{10}$ entries. For this reason, the translation process uses 10 bits at a time from a 32-bit linear address. Bits 31:22 identify the first paging-structure entry and bits 21:12 identify a second. The latter identifies the page frame. Bits 11:0 of the linear address are the page offset within the 4-KByte page frame. (See Figure 4-2 for an illustration.)
- With PAE paging, the first paging structure comprises only $4 = 2^2$ entries. Translation thus begins by using bits 31:30 from a 32-bit linear address to identify the first paging-structure entry. Other paging structures comprise $512 = 2^9$ entries, so the process continues by using 9 bits at a time. Bits 29:21 identify a second paging-structure entry and bits 20:12 identify a third. This last identifies the page frame. (See Figure 4-5 for an illustration.)
- With IA-32e paging, each paging structure comprises $512 = 2^9$ entries and translation uses 9 bits at a time from a 48-bit linear address. Bits 47:39 identify the first paging-structure entry, bits 38:30 identify a second, bits 29:21 a third, and bits 20:12 identify a fourth. Again, the last identifies the page frame. (See Figure 4-8 for an illustration.)

The translation process in each of the examples above completes by identifying a page frame. However, the paging structures may be configured so that translation terminates before doing so. This occurs if process encounters a paging-structure entry that is marked "not present" (because its P flag — bit 0 — is clear) or in which a reserved bit is set. In this case, there is no translation for the linear address; an access to that address causes a page-fault exception (see Section 4.7).

In the examples above, a paging-structure entry maps a page with 4-KByte page frame when only 12 bits remain in the linear address; entries identified earlier always reference other paging structures. That may not apply in other cases. The following items identify when an entry maps a page and when it references another paging structure:

- If more than 12 bits remain in the linear address, bit 7 (PS — page size) of the current paging-structure entry is consulted. If the bit is 0, the entry references another paging structure; if the bit is 1, the entry maps a page.
- If only 12 bits remain in the linear address, the current paging-structure entry always maps a page (bit 7 is used for other purposes).

If a paging-structure entry maps a page when more than 12 bits remain in the linear address, the entry identifies a page frame larger than 4 KBytes. For example, 32-bit paging uses the upper 10 bits of a linear address to locate the first paging-structure entry; 22 bits remain. If that entry maps a page, the page frame is 2^{22} Bytes = 4 MBytes. 32-bit paging supports 4-MByte pages if CR4.PSE = 1. PAE paging and IA-32e paging support 2-MByte pages (regardless of the value of CR4.PSE).

Paging structures are given different names based their uses in the translation process. Table Table 4-2. gives the names of the different paging structures. It also provides, for each structure, the source of the physical address used to locate it (CR3 or a different paging-structure entry); the bits in the linear address used to select an entry from the structure; and details of about whether and how such an entry can map a page.

Table 4-2. Paging Structures in the Different Paging Modes

Paging Structure	Entry Name	Paging Mode	Physical Address of Structure	Bits Selecting Entry	Page Mapping
PML4 table	PML4E	32-bit, PAE	N/A		
		IA-32e	CR3	47:39	N/A (PS must be 0)
Page-directory-pointer table	PDPTE	32-bit	N/A		
		PAE	CR3	31:30	N/A (PS must be 0)
		IA-32e	PML4E	38:30	
Page directory	PDE	32-bit	CR3	31:22	4-MByte page if PS=1[1]
		PAE, IA-32e	PDPTE	29:21	2-MByte page if PS=1
Page table	PTE	32-bit	PDE	21:12	4-KByte page
		PAE, IA-32e		20:12	4-KByte page

NOTES:

1. 32-bit paging ignores the PS flag in a PDE (and uses the entry to reference a page table) unless CR4.PSE = 1. Not all processors allow CR4.PSE to be 1; see Section 4.1.4 for how to determine whether 4-MByte pages are supported with 32-bit paging.

4.3 32-BIT PAGING

A logical processor uses 32-bit paging if CR0.PG = 1 and CR4.PAE = 0. 32-bit paging translates 32-bit linear addresses to 40-bit physical addresses.[1] Although 40 bits corre-

sponds to 1 TByte, linear addresses are limited to 32 bits; at most 4 GBytes of linear-address space may be accessed at any given time.

32-bit paging uses a hierarchy of paging structures to produce a translation for a linear address. CR3 is used to locate the first paging-structure, the page directory. Table Table 4-3. illustrates how CR3 is used with 32-bit paging.

Table 4-3. Use of CR3 with 32-Bit Paging

Bit Position(s)	Contents
2:0	Ignored
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory during linear-address translation (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory during linear-address translation (see Section 4.9)
11:5	Ignored
31:12	Physical address of the 4-KByte aligned page directory used for linear-address translation
63:32	Ignored (these bits exist only on processors supporting the Intel-64 architecture)

32-bit paging may map linear addresses to either 4-KByte pages or 4-MByte pages. Figure 4-2 illustrates the translation process when it uses a 4-KByte page; Figure 4-3 covers the case of a 4-MByte page. The following items describe the 32-bit paging process in more detail as well has how the page size is determined:

- A 4-KByte naturally aligned page directory is located at the physical address specified in bits 31:12 of CR3 (see Table Table 4-3.). A page directory comprises 1024 32-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
  - Bits 39:32 are all 0.
  - Bits 31:12 are from CR3.
  - Bits 11:2 are bits 31:22 of the linear address.
  - Bits 1:0 are 0.

Because a PDE is identified using bits 31:22 of the linear address, it controls access to a 4-Mbyte region of the linear-address space. Use of the PDE depends on CR.PSE and the PDE's PS flag (bit 7):

- If CR4.PSE = 1 and the PDE's PS flag is 1, the PDE maps a 4-MByte page (see Table Table 4-4.). The final physical address is computed as follows:
  - Bits 39:32 are bits 20:13 of the PDE.

1. Bits in the range 39:32 are 0 in any physical address used by 32-bit paging except those used to map 4-MByte pages. If the processor does not support the PSE-36 mechanism, this is true also for physical addresses used to map 4-MByte pages. If the processor does support the PSE-36 mechanism and MAXPHYADDR < 40, bits in the range 39:MAXPHYADDR are 0 in any physical address used to map a 4-MByte page. (The corresponding bits are reserved in PDEs.) See Section 4.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.

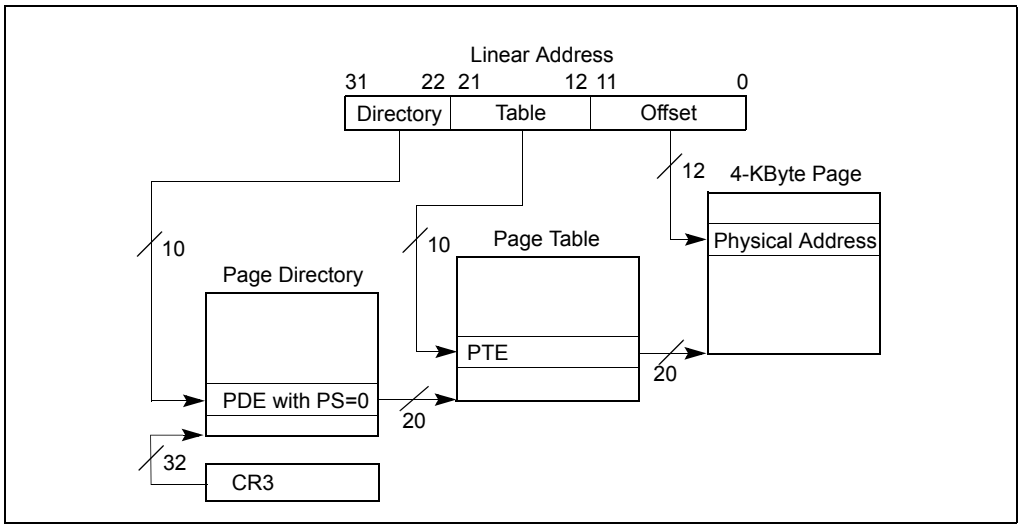

Figure 4-2 Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

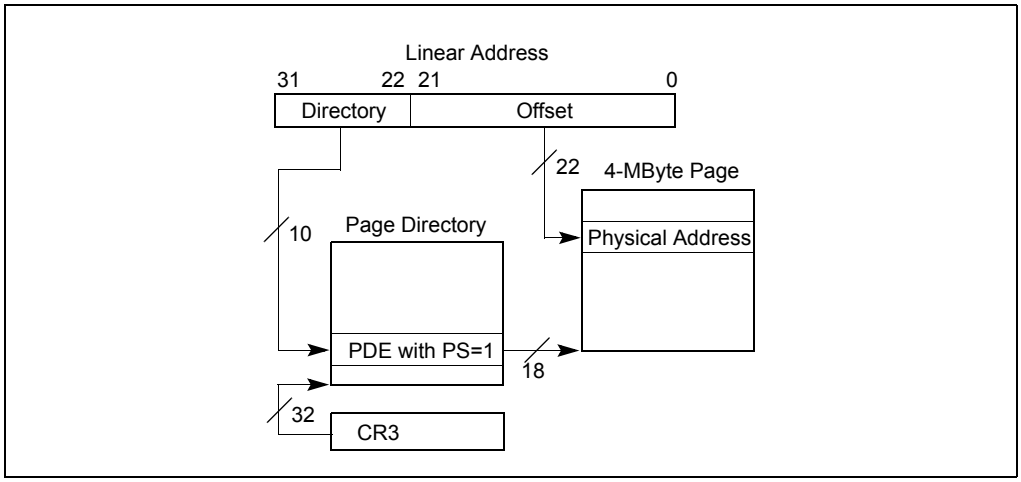

Figure 4-3 Linear-Address Translation to a 4-MByte Page using 32-Bit Paging

— Bits 31:22 are bits 31:22 of the PDE.[1]

— Bits 21:0 are from the original linear address.

- If CR4.PSE = 0 or the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 31:12 of the PDE (see Table Table 4-5.). A page table comprises 1024 32-bit entries (PTEs). A PTE is selected using the physical address defined as follows:

  — Bits 39:32 are all 0.

  — Bits 31:12 are from the PDE.

1. The upper bits in the final physical address do not all come from corresponding positions in the PDE; the physical-address bits in the PDE are not all contiguous.

Table 4-4. Format of a 32-Bit Page-Directory Entry that Maps a 4-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 4-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table Table 4-5.)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 4.9); otherwise, reserved (must be 0)[1]
M–20:13	Bits M–1:32 of physical address of the 4-MByte page referenced by this entry[2]
21:M–19	Reserved (must be 0)
31:22	Bits 31:22 of physical address of the 4-MByte page referenced by this entry

NOTES:

1. See Section 4.1.4 for how to determine whether the PAT is supported.
2. If the PSE-36 mechanism is not supported, M is 32, and this row does not apply. If the PSE-36 mechanism is supported, M is the minimum of 40 and MAXPHYADDR (this row does not apply if MAXPHYADDR = 32). See Section 4.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.

  - Bits 11:2 are bits 21:12 of the linear address.
  - Bits 1:0 are 0.

- Because a PTE is identified using bits 31:12 of the linear address, every PTE maps a 4-KByte page (see Table Table 4-6.). The final physical address is computed as follows:
  - Bits 39:32 are all 0.
  - Bits 31:12 are from the PTE.

Table 4-5. Format of a 32-Bit Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte region controlled by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 4-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	If CR4.PSE = 1, must be 0 (otherwise, this entry maps a 4-MByte page; see Table Table 4-4.); otherwise, ignored
11:8	Ignored
31:12	Physical address of 4-KByte aligned page table referenced by this entry

  — Bits 11:0 are from the original linear address.

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. A reference using a linear address whose translation would use such a paging-structure entry causes a page-fault exception (see Section 4.7).

With 32-bit paging, there are reserved bits only if CR4.PSE = 1:

- If the P flag and the PS flag (bit 7) of a PDE are both 1, the bits reserved depend on MAXPHYADDR whether the PSE-36 mechanism is supported:[1]
  - If the PSE-36 mechanism is not supported, bits 21:13 are reserved.
  - If the PSE-36 mechanism is supported, bits 21:M–19 are reserved, where M is the minimum of 40 and MAXPHYADDR.
- If the PAT is not supported:[2]
  - If the P flag of a PTE is 1, bit 7 is reserved.
  - If the P flag and the PS flag of a PDE are both 1, bit 12 is reserved.

(If CR4.PSE = 0, no bits are reserved with 32-bit paging.)

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

1. See Section 1.1.5 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.
2. See Section 4.1.4 for how to determine whether the PAT is supported.

Table 4-6. Format of a 32-Bit Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9); otherwise, reserved (must be 0)[1]
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
31:12	Physical address of the 4-KByte page referenced by this entry

NOTES:
1. See Section 4.1.4 for how to determine whether the PAT is supported.

Figure 4-4 gives a summary of the formats of CR3 and the paging-structure entries with 32-bit paging. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are "not present"; bit 0 (P) and bit 7 (PS) are highlighted because they determine how such an entry is used.

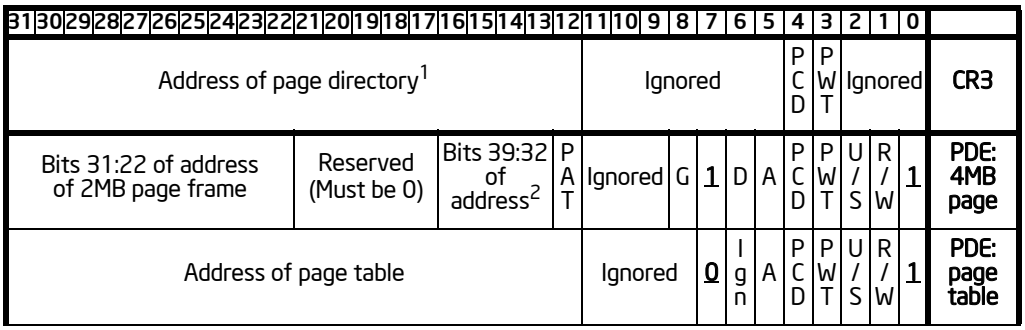

Figure 4-4 Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Ignored																															0	PDE: not present
Address of 4KB page frame																Ignored			G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page				
Ignored																															0	PTE: not present

Figure 4-4 Formats of CR3 and Paging-Structure Entries with 32-Bit Paging (Continued)

NOTES:

1. CR3 has 64 bits on processors supporting the Intel-64 architecture. These bits are ignored with 32-bit paging.
2. This example illustrates a processor in which MAXPHYADDR is 36. If this value is larger or smaller, the number of bits reserved in positions 20:13 of a PDE mapping a 4-MByte will change.

4.4 PAE PAGING

A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0. PAE paging translates 32-bit linear addresses to 52-bit physical addresses.[1] Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 32 bits; at most 4 GBytes of linear-address space may be accessed at any given time.

With PAE paging, a logical processor maintains a set of four (4) PDPTE registers, which are loaded from an address in CR3. Linear address are translated using 4 hierarchies of in-memory paging structures, each located using one of the PDPTE registers. (This is different from the other paging modes, in which there is one hierarchy referenced by CR3.)

Section 4.4.1 discusses the PDPTE registers. Section 4.4.2 describes linear-address translation with PAE paging.

4.4.1 PDPTE Registers

When PAE paging is used, CR3 references the base of a 32-Byte **page-directory-pointer table**. Table Table 4-7. illustrates how CR3 is used with PAE paging.

Table 4-7. Use of CR3 with PAE Paging

Bit Position(s)	Contents
4:0	Ignored
31:5	Physical address of the 32-Byte aligned page-directory-pointer table used for linear-address translation

1. If MAXPHYADDR < 52, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by PAE paging. (The corresponding bits are reserved in the paging-structure entries.) See Section 4.1.4 for how to determine MAXPHYADDR.

Table 4-7. Use of CR3 with PAE Paging (Continued)

Bit Position(s)	Contents
63:32	Ignored (these bits exist only on processors supporting the Intel-64 architecture)

The page-directory-pointer-table comprises four (4) 64-bit entries called PDPTEs. Each PDPTE controls access to a 1-GByte region of the linear-address space. Corresponding to the PDPTEs, the logical processor maintains a set of four (4) internal, non-architectural PDPTE registers, called PDPTE0, PDPTE1, PDPTE2, and PDPTE3. The logical processor loads these registers from the PDPTEs in memory as part of certain executions the MOV to CR instruction:

- If an execution MOV to CR0 or MOV to CR4 causes the logical processor to transition from either no paging or 32-bit paging to PAE paging (see Section 4.1.2), the PDPTEs are loaded from the address in CR3.
- If MOV to CR3 is executed while the logical processor is using PAE paging, the PDPTEs are loaded from the address being loaded into CR3.
- If PAE paging is in use and a task switch changes the value of CR3, the PDPTEs are loaded from the address in the new CR3 value.
- Certain VMX transitions load the PDPTE registers. See Section 4.11.1.

Unless the caches are disabled, the processor uses the WB memory type to load the PDPTEs from memory.[1]

1. Older IA-32 processors used the UC memory type when loading the PDPTEs. This behavior is model-specific and not architectural.

Table Table 4-8. gives the format of a PDPTE. If any of the PDPTEs sets both the P flag

Table 4-8. Format of an PAE Page-Directory-Pointer-Table Entry (PDPTE)

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
2:1	Reserved (must be 0)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9)
8:5	Reserved (must be 0)
11:9	Ignored
M–1:12	Physical address of 4-KByte aligned page directory referenced by this entry[1]
63:M	Reserved (must be 0)

NOTES:
1. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.

(bit 0) and any reserved bit, the MOV to CR instruction causes a general-protection exception (#GP(0)) and the PDPTEs are not loaded.[1] As show in Table Table 4-8., bits 2:1, 8:5, and 63:MAXPHYADDR are reserved in the PDPTEs.

4.4.2 Linear-Address Translation with PAE Paging

PAE paging may map linear addresses to either 4-KByte pages or 2-MByte pages. Figure 4-5 illustrates the translation process when it produces a 4-KByte page; Figure 4-6 covers the case of a 2-MByte page. The following items describe the PAE paging process in more detail as well has how the page size is determined:

- Bits 31:30 of the linear address select a PDPTE register (see Section 4.4.1); this is PDPTE*i*, where *i* is the value of bits 31:30.[2] Because a PDPTE register is identified using bits 31:30 of the linear address, it controls access to a 1-GByte region of the linear-address space. If the P flag (bit 0) of PDPTE*i* is 0, the processor ignores bits 63:1, and there is no mapping for the 1-GByte region controlled by PDPTE*i*. A reference using a linear address in this region causes a page-fault exception (see Section 4.7).
- If the P flag of PDPTE*i* is 1, 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of PDPTE*i* (see Table Table 4-8. in Section 4.4.1) A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:

1. On some processors, reserved bits are checked even in PDPTEs in which the P flag (bit 0) is 0.
2. With PAE paging, the processor does not use CR3 when translating a linear address (as it does the other paging modes). It does not access the PDPTEs in the page-directory-pointer table during linear-address translation.

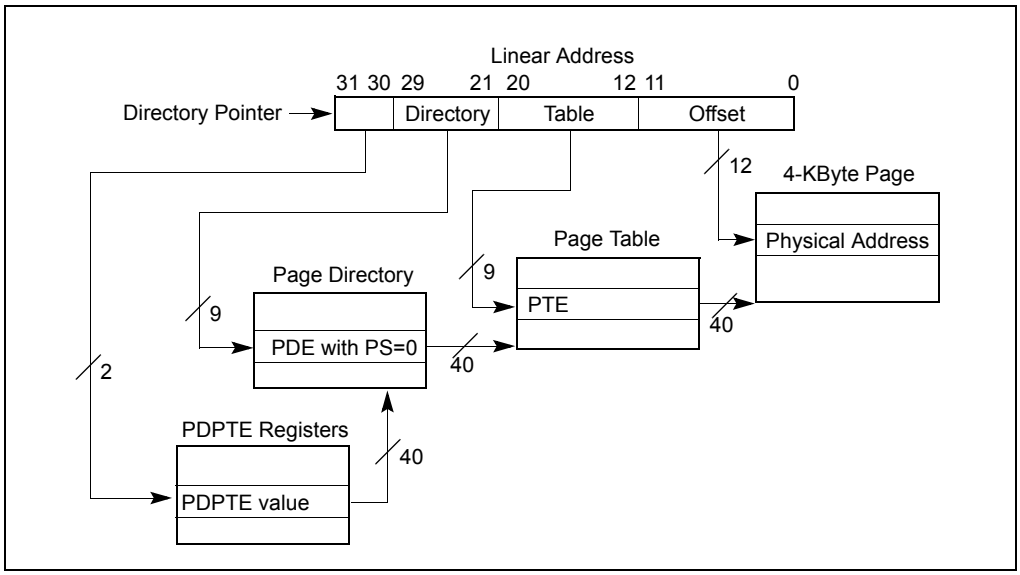

Figure 4-5 Linear-Address Translation to a 4-KByte Page using PAE Paging

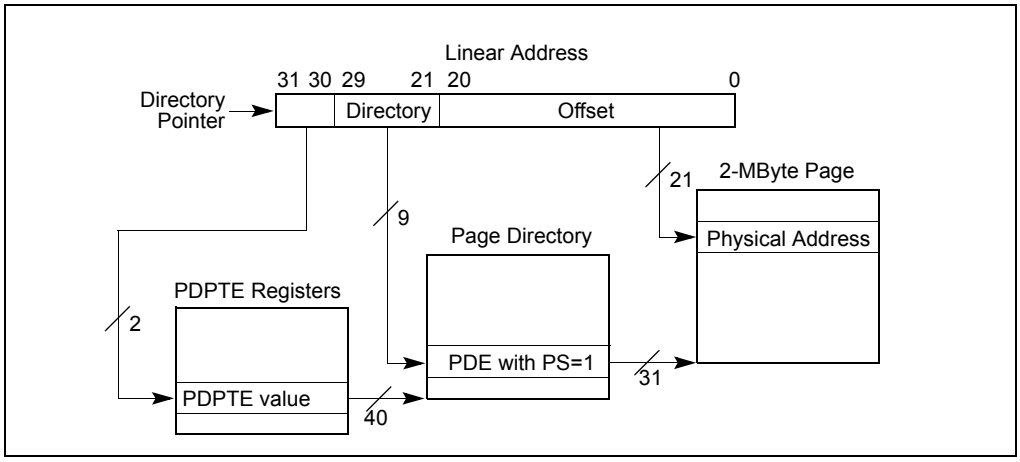

Figure 4-6 Linear-Address Translation to a 2-MByte Page using PAE Paging

— Bits 51:12 are from PDPTE*i*.

— Bits 11:3 are bits 29:21 of the linear address.

— Bits 2:0 are 0.

Because a PDE is identified using bits 31:21 of the linear address, it controls access to a 2-Mbyte region of the linear-address space. Use of the PDE depends on its PS flag (bit 7):

- If the PDE's PS flag is 1, the PDE maps a 2-MByte page (see Table Table 4-9.). The final physical address is computed as follows:

  — Bits 51:21 are from the PDE.

  — Bits 20:0 are from the original linear address.

Table 4-9. Format of a PAE Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 2-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table Table 4-10.)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9); otherwise, reserved (must be 0)[1]
20:13	Reserved (must be 0)
M–1:21	Physical address of the 2-MByte page referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:
1. See Section 4.1.4 for how to determine whether the PAT is supported.

- If the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE (see Table Table 4-10.). A page directory comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
  - Bits 51:12 are from the PDE.
  - Bits 11:3 are bits 20:12 of the linear address.
  - Bits 2:0 are 0.

Table 4-10. Format of a PAE Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 2-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table Table 4-9.)
11:8	Ignored
M–1:12	Physical address of 4-KByte aligned page table referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

- Because a PTE is identified using bits 31:12 of the linear address, every PTE maps a 4-KByte page (see Table Table 4-11.). The final physical address is computed as follows:
  - Bits 51:12 are from the PTE.
  - Bits 11:0 are from the original linear address.

If the P flag (bit 0) of a PDE or a PTE is 0 or if a PDE or a PTE sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. A reference using a linear address whose translation would use such a paging-structure entry causes a page-fault exception (see Section 4.7).

The following bits are reserved with PAE paging:

- If the P flag (bit 0) of a PDE or a PTE is 1, bits 62:MAXPHYADDR are reserved.
- If the P flag and the PS flag (bit 7) of a PDE are both 1, bits 20:13 are reserved.
- If IA32_EFER.NXE = 0 and the P flag of a PDE or a PTE is 1, the XD flag (bit 63) is reserved.
- If the PAT is not supported:[1]

1. See Section 4.1.4 for how to determine whether the PAT is supported.

Table 4-11. Format of a PAE Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9); otherwise, reserved (must be 0)[1]
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
M–1:12	Physical address of the 4-KByte page referenced by this entry
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:
1. See Section 4.1.4 for how to determine whether the PAT is supported.

— If the P flag of a PTE is 1, bit 7 is reserved.

— If the P flag and the PS flag of a PDE are both 1, bit 12 is reserved.

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

Figure 4-7 gives a summary of the formats of CR3 and the paging-structure entries with PAE paging. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither

because they are "not present"; bit 0 (P) and bit 7 (PS) are highlighted because they determine how a paging-structure entry is used.

63	62–M	M-1	32	31–12	11–9	8	7	6	5	4	3	2	1	0	
Ignored[2]				Address of page-directory-pointer table											CR3
Reserved[3]				Address of page directory		Ign.		Rsvd.		PCD	PWT	Rsvd		1	PDPTE: present
Ignored														0	PDTPE: not present
XD	Ignored		Rsvd.	Address of 2MB page frame	Reserved	PAT	Ign.	G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 2MB page
XD	Ignored		Rsvd.	Address of page table			Ign.		0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored														0	PDE: not present
XD	Ignored		Rsvd.	Address of 4KB page frame			Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored														0	PTE: not present

Figure 4-7 Formats of CR3 and Paging-Structure Entries with PAE Paging

NOTES:

1. M is an abbreviation for MAXPHYADDR.
2. CR3 has 64 bits only on processors supporting the Intel-64 architecture. These bits are ignored with PAE paging.
3. Reserved fields must be 0.

4.5 IA-32E PAGING

A logical processor uses IA-32e paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 1. With IA-32e paging, linear address are translated using a hierarchy of in-memory paging structures located using the contents of CR3. IA-32e paging translates 48-bit linear addresses to 52-bit physical addresses.[1] Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 48 bits; at most 256 TBytes of linear-address space may be accessed at any given time.

1. If MAXPHYADDR < 52, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by IA-32e paging. (The corresponding its are reserved in the paging-structure entries.) See Section 4.1.4 for how to determine MAXPHYADDR.

IA-32e paging uses a hierarchy of paging structures to produce a translation for a linear address. CR3 is used to locate the first paging-structure, the PML4 table. Table Table 4-12. illustrates how CR3 is used with IA-32e paging.

Table 4-12. Use of CR3 with IA-32e Paging

Bit Position(s)	Contents
2:0	Ignored
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table during linear-address translation (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table during linear-address translation (see Section 4.9)
11:5	Ignored
M–1:12	Physical address of the 4-KByte aligned PML4 table used for linear-address translation[1]
63:M	Reserved (Must be 0)

NOTES:
1. M is an abbreviation for MAXPHYADDR, which is at most 52; see Section 4.1.4.

IA-32e paging may map linear addresses to either 4-KByte pages or 2-MByte pages. Figure 4-8 illustrates the translation process when it produces a 4-KByte page; Figure 4-9 covers the case of a 2-MByte page. The following items describe the IA-32e paging process in more detail as well has how the page size is determined:

- A 4-KByte naturally aligned PML4 table is located at the physical address specified in bits 51:12 of CR3 (see Table Table 4-12.). A PML4 table comprises 512 64-bit entries (PML4Es). A PML4E is selected using the physical address defined as follows:
  - Bits 51:12 are from CR3.
  - Bits 11:3 are bits 47:39 of the linear address.
  - Bits 2:0 are all 0.

  Because a PML4E is identified using bits 47:39 of the linear address, it controls access to a 512-GByte region of the linear-address space.
- A 4-KByte naturally aligned page-directory-pointer table is located at the physical address specified in bits 51:12 of the PML4E (see Table Table 4-13.). A page-directory-pointer table comprises 512 64-bit entries (PDPTEs). A PDPTE is selected using the physical address defined as follows:
  - Bits 51:12 are from the PML4E.
  - Bits 11:3 are bits 38:30 of the linear address.
  - Bits 2:0 are all 0.

  Because a PDPTE is identified using bits 47:30 of the linear address, it controls access to a 1-GByte region of the linear-address space.
- A 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of the PDPTE (see Table Table 4-14.). A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:

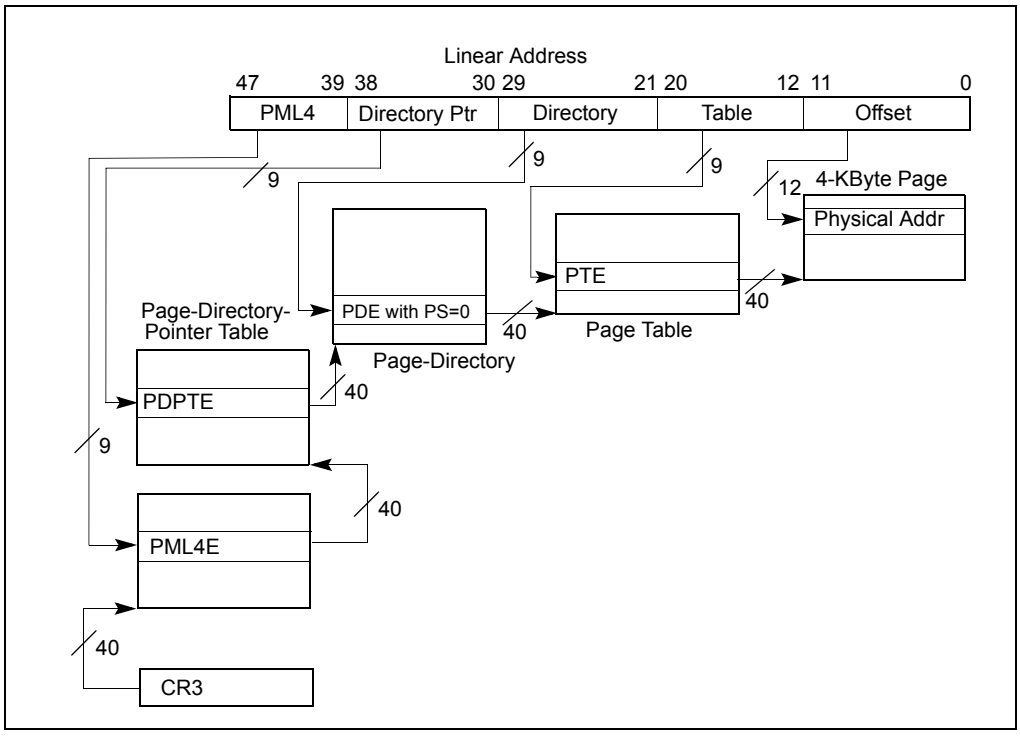

Figure 4-8 Linear-Address Translation to a 4-KByte Page using IA-32e Paging

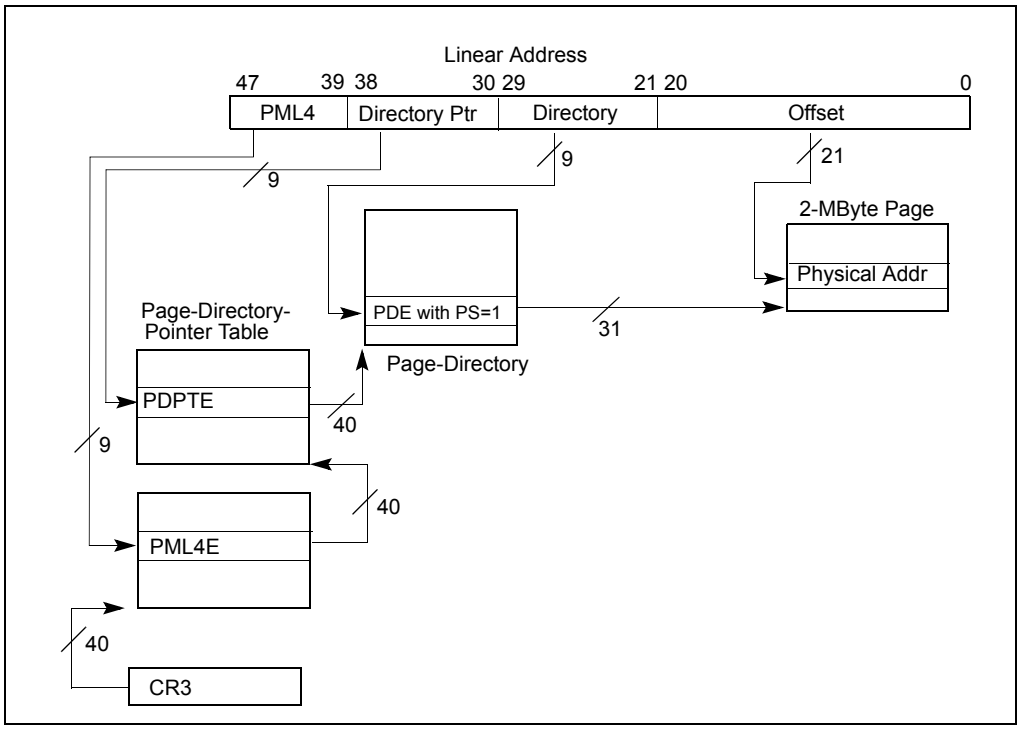

Figure 4-9 Linear-Address Translation to a 2-MByte Page using IA-32e Paging

Table 4-13. Format of an IA-32e PML4 Entry (PML4E) that References a Page-Directory-Pointer Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page-directory-pointer table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 512-GByte region controlled by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 512-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Reserved (must be 0)
11:8	Ignored
M–1:12	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 512-GByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

— Bits 51:12 are from the PDPTE.

— Bits 11:3 are bits 29:21 of the linear address.

— Bits 2:0 are all 0.

Because a PDE is identified using bits 47:21 of the linear address, it controls access to a 2-MByte region of the linear-address space. Use of the PDE depends on its PS flag (bit 7):

Table 4-14. Format of an IA-32e Page-Directory-Pointer-Table Entry (PDPTE) that References a Page Directory

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GByte region controlled by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 1-GByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Reserved (must be 0)
11:8	Ignored
M–1:12	Physical address of 4-KByte aligned page directory referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

- If the PDE's PS flag is 1, the PDE maps a 2-MByte page (see Table Table 4-15.). The final physical address is computed as follows:

Table 4-15. Format of an IA-32e Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 2-MByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)

Table 4-15. Format of an IA-32e Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 4.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table Table 4-16.)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
12 (PAT)	Indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)[1]
20:13	Reserved (must be 0)
M–1:21	Physical address of the 2-MByte page referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

NOTES:
1. The PAT is supported on all processors that support IA-32e paging.

  — Bits 51:21 are from the PDE.
  — Bits 20:0 are from the original linear address.

- If the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE (see Table Table 4-16.). A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:

Table 4-16. Format of an IA-32e Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (depends on CPL and CR0.WP; see Section 4.6)

Table 4-16. Format of an IA-32e Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 2-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table Table 4-15.)
11:8	Ignored
M–1:12	Physical address of 4-KByte aligned page table referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

  - Bits 51:12 are from the PDE.
  - Bits 11:3 are bits 20:12 of the linear address.
  - Bits 2:0 are all 0.
- Because a PTE is identified using bits 47:12 of the linear address, every PTE maps a 4-KByte page (see Table Table 4-17.). The final physical address is computed as follows:
  - Bits 51:12 are from the PTE.
  - Bits 11:0 are from the original linear address.

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. A reference using a linear address whose translation would use such a paging-structure entry causes a page-fault exception (see Section 4.7).

The following bits are reserved with IA-32e paging:

- If the P flag of a paging-structure entry is 1, bits 51:MAXPHYADDR are reserved.
- If the P flag of a PML4E or a PDPTE is 1, the PS flag is reserved.
- If the P flag and the PS flag of a PDE are both 1, bits 20:13 are reserved.
- If IA32_EFER.NXE = 0 and the P flag of a paging-structure entry is 1, the XD flag (bit 63) is reserved.

Table 4-17. Format of an IA-32e Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (depends on CPL and CR0.WP; see Section 4.6)
2 (U/S)	User/supervisor; if 0, accesses with CPL=3 are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 4.9)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
M–1:12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 4.6.

Figure 4-10 gives a summary of the formats of CR3 and the IA-32e paging-structure entries. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are "not present"; bit 0 (P) and bit 7 (PS) are highlighted because they determine how a paging-structure entry is used.

4.6 ACCESS RIGHTS

There is a translation for a linear address if the processes described in Section 4.3, Section 4.4.2, and Section 4.5 (depending upon the paging mode) completes and produces a physical address. The accesses permitted by a translation is determined by

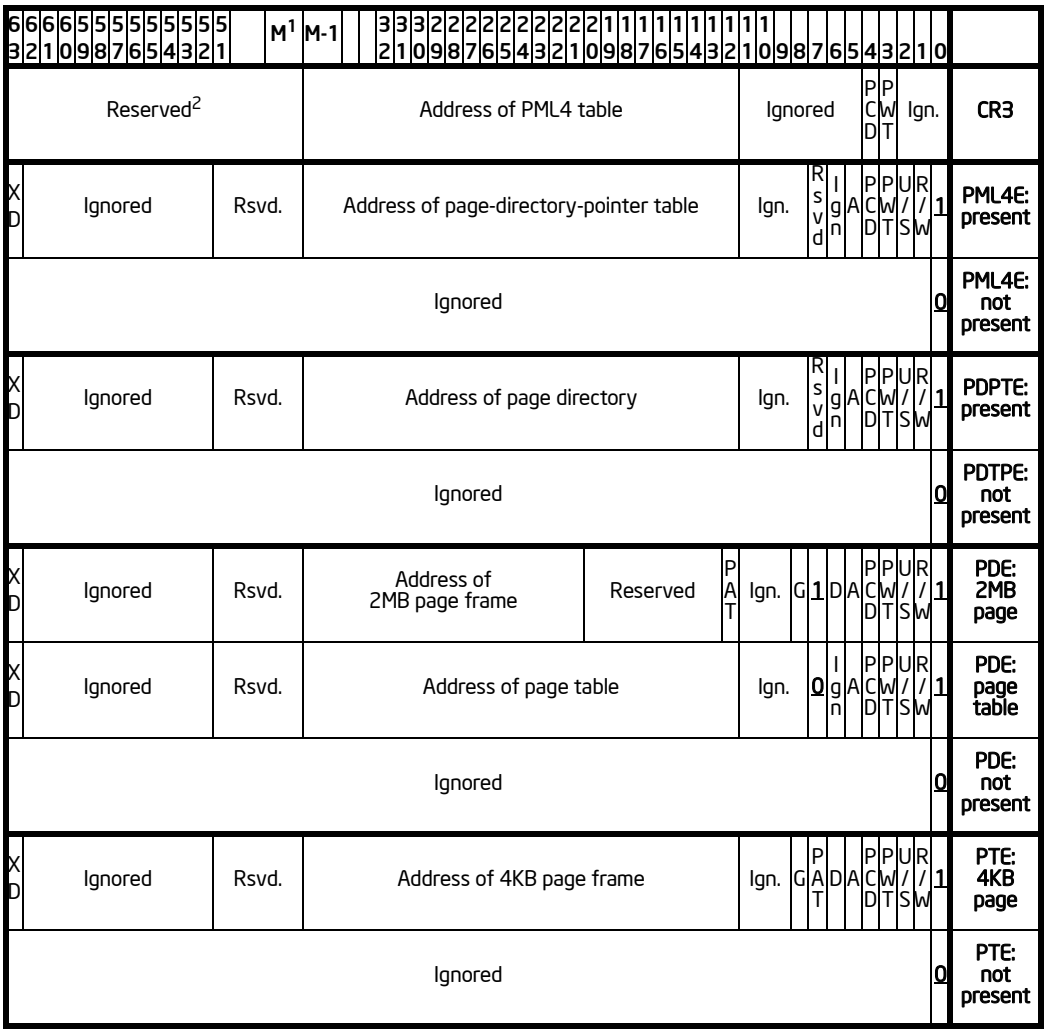

Figure 4-10 Formats of CR3 and Paging-Structure Entries with IA-32e Paging

NOTES:
1. M is an abbreviation for MAXPHYADDR.
2. Reserved fields must be 0.

the access rights specified by the paging-structure entries controlling the translation.[1] The following items detail how paging determines access rights:

- For accesses in supervisor mode (CPL < 3):
  - — Data reads.
    Data may be read from any linear address with a valid translation.
  - — Data writes.
    - If CR0.WP = 0, data may be written to any linear address with a valid translation.

1. With PAE paging, the PDPTEs do not determine access rights.

- If CR0.WP = 1, data may be written to any linear address with a valid translation for which the R/W flag (bit 1) is 1 in every paging-structure entry controlling the translation.

— Instruction fetches.

- For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any linear address with a valid translation.
- For PAE paging or IA-32e paging with IA32_EFER.NXE = 1, instructions may be fetched from any linear address with a valid translation for which the XD flag (bit 63) is 0 in every paging-structure entry controlling the translation.

- For accesses in user mode (CPL = 3):

— Data reads.
Data may be read from any linear address with a valid translation for which the U/S flag (bit 2) is 1 in every paging-structure entry controlling the translation.

— Data writes.
Data may be written to any linear address with a valid translation for which both the R/W flag and the U/S flag are 1 in every paging-structure entry controlling the translation.

— Instruction fetches.

- For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any linear address with a valid translation for which the U/S flag is 1 in every paging-structure entry controlling the translation.
- For PAE paging or IA-32e paging with IA32_EFER.NXE = 1, instructions may be fetched from any linear address with a valid translation for which the U/S flag is 1 and the XD flag is 0 in every paging-structure entry controlling the translation.

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). These structures may include information about access rights. The processor may enforce access rights based on the TLBs and paging-structure caches instead of on the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change access rights, the processor might not use that change for a subsequent access to an affected linear address (see Section 4.10.3.3). See Section 4.10.3.2 for how software can ensure that the processor uses the modified access rights.

4.7 PAGE-FAULT EXCEPTIONS

Accesses using linear addresses may cause **page-fault exceptions** (#PF; exception 14). An access to a linear address may cause page-fault exception for either of two reasons: (1) there is no valid translation for the linear address; or (2) there is a valid translation for the linear address, but its access rights do not permit the access.

As noted in Section 4.3, Section 4.4.2, and Section 4.5, there is no valid translation for a linear address if the translation process for that address would use a paging-structure entry in which the P flag (bit 0) is 0 or one that sets a reserved bit. If there is a valid translation for a linear address, its access rights are determined as specified in Section 4.6.

Figure 4-11 illustrates the error code that the processor provides on delivery of a page-fault exception. The following items explain how the bits in the error code describe the nature of the page-fault exception:

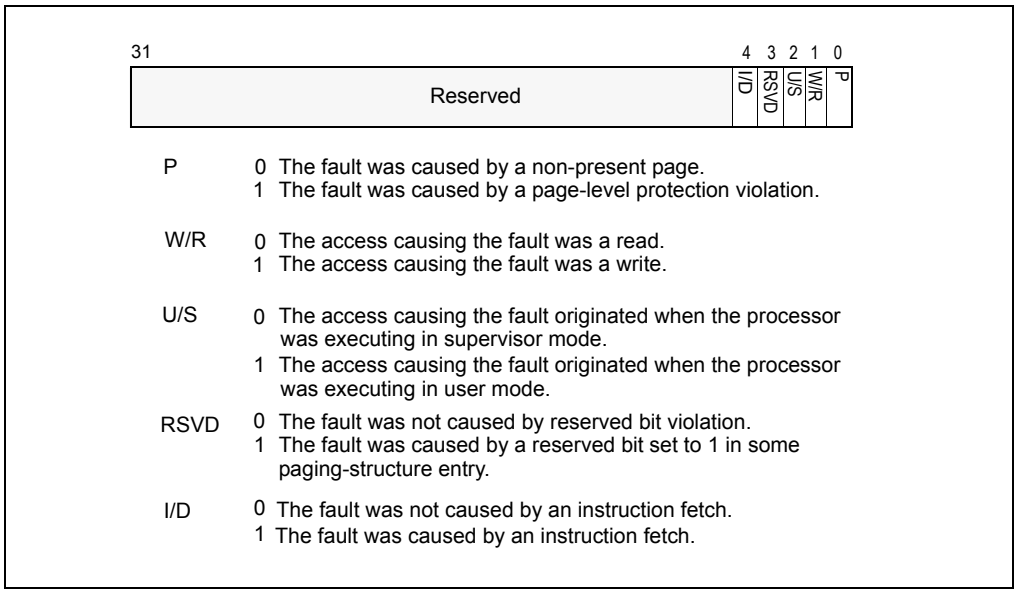

Figure 4-11 Page-Fault Error Code

- P flag (bit 0).
  This bit is 0 if there is no valid translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address.
- W/R (bit 1).
  If the access causing the page-fault exception was a write, this flag is 1; otherwise, it is 0. This bit describes the access causing the page-fault exception, not the access rights specified by paging.
- U/S (bit 2).
  If a supervisor-mode (CPL < 3) access caused the page-fault exception, this flag is 1; it is 0 if a user-mode (CPL = 3) access did so. This bit describes the access causing the page-fault exception, not the access rights specified by paging.
- RSVD flag (bit 4).
  This bit is 0 if there is no valid translation for the linear address because a reserved bit was set in one of the paging-structure entries used to translate that address. (Because reserved bits are not checked in a paging-structure entry whose P flag is 0, bit 4 of the error code can be set only if bit 0 is also set.)
- I/D flag (bit 5).
  Use of this flag depends on the setting of IA32_EFER.NXE:
  - IA32_EFER.NXE = 0. This flag is 0.
  - IA32_EFER.NXE = 1. If the access causing the page-fault exception was an instruction fetch, this flag is 1; otherwise, it is 0. This bit describes the access causing the page-fault exception, not the access rights specified by paging.

Bits reserved in the paging-structure entries are reserved for future functionality. Software developers should be aware that such settings may be used in the future and that

a paging-structure entry that causes a page-fault exception on one processor might not do so in the future.

Page-fault exceptions occur only due to an attempt to use a linear address. Failures to load the PDPTE registers with PAE paging (see Section 4.4.1) cause general-protection exceptions (#GP(0)) and not page-fault exceptions.

4.8 ACCESSED AND DIRTY FLAGS

For any paging-structure entry that is used during linear-address translation, bit 5 is the **accessed** flag.[1] For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the **dirty** flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a linear address, the processor sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a PDE in which the PS flag is 1).

Memory-management software may clear these flags when a page or a paging structure is initially loaded into physical memory. These flags are "sticky," meaning that, once set, the processor does not clear them; only software can clear them.

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). This fact implies that, if software changes an accessed flag or a dirty flag from 1 to 0, the processor might not set the corresponding bit in memory on a subsequent access using an affected linear address (see Section 4.10.3.3). See Section 4.10.3.2 for how software can ensure that these bits are updated as desired.

NOTE

The accesses used by the processor to set these flags may or may not be exposed to the processor's self-modifying code detection logic. If the processor is executing code from the same memory area that is being used for the paging structures, the setting of these flags may or may not result in an immediate change to the executing code stream.

4.9 PAGING AND MEMORY TYPING

The **memory type** of a memory access refers to the type of caching used for that access. Chapter 11, "Memory Cache Control" provides many details regarding memory typing in the Intel-64 and IA-32 architectures. This section describes how paging contributes to the determination of memory typing.

The way in which paging contributes to memory typing depends on whether the processor supports the **Page Attribute Table** (**PAT**; see Section 11.12).[2] Section 4.9.1

1. With PAE paging, the PDPTEs are not used during linear-address translation but only to load the PDPTE registers for some executions of the MOV CR instruction (see Section 4.4.1). For this reason, the PDPTEs do not contain accessed flags with PAE paging.

and Section 4.9.2 explain how paging contributes to memory typing depending on whether the PAT is supported.

4.9.1 Paging and Memory Typing When the PAT is Not Supported (Pentium Pro and Pentium II Processors)

NOTE

The PAT is supported on all processors that support IA-32e paging. Thus, this section applies only to 32-bit paging and PAE paging.

If the PAT is not supported, paging contributes to memory typing in conjunction with the memory-type range registers (MTRRs) as specified in Table 11-6 in Section 11.5.2.1.

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a PCD value and a PWT value. The latter two values are determined as follows:

- For an access to a PDE with 32-bit paging, the PCD and PWT values come from CR3.
- For an access to a PDE with PAE paging, the PCD and PWT values come from the relevant PDPTE register.
- For an access to a PTE, the PCD and PWT values come from the relevant PDE.
- For an access to the physical address that is the translation of a linear address, the PCD and PWT values come from the relevant PTE (if the translation uses a 4-KByte page) or the relevant PDE (otherwise).

4.9.2 Paging and Memory Typing When the PAT is Supported (Pentium III and More Recent Processor Families)

If the PAT is supported, paging contributes to memory typing in conjunction with the PAT and the memory-type range registers (MTRRs) as specified in Table 11-7 in Section 11.5.2.2.

The PAT is a 64-bit MSR (IA32_PAT; MSR index 277H) comprising eight (8) 8-bit entries (entry *i* comprises bits 8*i*+7:8*i* of the MSR).

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a memory type selected from the PAT. Table 11-11 in Section 11.12.3 specifies how a memory type is selected from the PAT. Specifically, it comes from entry *i* of the PAT, where *i* is defined as follows:

- For an access to an entry in a paging structure whose address is in CR3 (e.g., the PML4 table with IA-32e paging), *i* = 2*PCD+PWT, where the PCD and PWT values come from CR3.
- For an access to a PDE with PAE paging, *i* = 2*PCD+PWT, where the PCD and PWT values come from the relevant PDPTE register.
- For an access to a paging-structure entry X whose address is in another paging-structure entry Y, *i* = 2*PCD+PWT, where the PCD and PWT values come from Y.
- For an access to the physical address that is the translation of a linear address, *i* = 4*PAT+2*PCD+PWT, where the PAT, PCD, and PWT values come from the relevant

2. The PAT is supported on Pentium III and more recent processor families. See Section 4.1.4 for how to determine whether the PAT is supported.

PTE (if the translation uses a 4-KByte page) or the relevant PDE (if the translation uses a 2-MByte page or a 4-MByte page).

4.9.3 Caching Paging-Related Information about Memory Typing

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). These structures may include information about memory typing. The processor may memory-typing information from the TLBs and paging-structure caches instead of from the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change the memory-typing bits, the processor might not use that change for a subsequent translation using that entry or for access to an affected linear address. See Section 4.10.3.2 for how software can ensure that the processor uses the modified memory typing.

4.10 CACHING TRANSLATION INFORMATION

The Intel-64 and IA-32 architectures may accelerate the address-translation process by caching data from the paging structures on the processor. Because the processor does not ensure that the data that it caches are always consistent with the structures in memory, it is important for software developers to understand how and when the processor may cache such data. They should also understand what actions software can take to remove cached data that may be inconsistent and when it should do so. This section provides software developers information about the relevant processor operation.

Section 4.10.1 and Section 4.10.2 describe how the processor may cache information in translation lookaside buffers (TLBs) and paging-structure caches, respectively. Section 4.10.3 explains how software can remove inconsistent cached information by invalidating portions of the TLBs and paging-structure caches. Section 4.10.4 describes special considerations for multiprocessor systems.

4.10.1 Translation Lookaside Buffers (TLBs)

A processor may cache information about the translation of linear addresses in translation lookaside buffers (TLBs). In general, TLBs contain entries that map page numbers to page frames; these terms are defined in Section 4.10.1.1. Section 4.10.1.2 describes how information may be cached in TLBs, and Section 4.10.1.3 gives details of TLB usage. Section 4.10.1.4 explains the global-page feature, which allows software to indicate that certain translations should receive special treatment when cached in the TLBs.

4.10.1.1 Page Numbers, Page Frames, and Page Offsets

Section 4.3, Section 4.4.2, and Section 4.5 give details of how the different paging modes translate linear addresses to physical addresses. Specifically, the upper bits of a linear address (called the **page number**) determine the upper bits of the physical address (called the **page frame**); the lower bits of the linear address (called the **page offset**) determine the lower bits of the physical address. The boundary between the page number and the page offset is determined by the **page size**. Specifically:

- 32-bit paging:

— If the translation does not use a PTE (because CR4.PSE = 1 and the PS flag is 1 in the PDE used), the page size is 4 MBytes and the page number comprises bits 31:22 of the linear address.

— If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 31:12 of the linear address.

- PAE paging:
  — If the translation does not use a PTE (because the PS flag is 1 in the PDE used), the page size is 2 MBytes and the page number comprises bits 31:21 of the linear address.
  — If the translation does uses a PTE, the page size is 4 KBytes and the page number comprises bits 31:12 of the linear address.
- IA-32e paging:
  — If the translation does not uses a PTE (because the PS flag is 1 in the PDE used), the page size is 2 MBytes and the page number comprises bits 47:21 of the linear address.
  — If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 47:12 of the linear address.

4.10.1.2 Caching Translations in TLBs

The processor may accelerate the paging process by caching individual translations in **translation lookaside buffers** (**TLBs**). Each entry in a TLB is an individual translation. Each translation is referenced by a page number. It contains the following information from the paging-structure entries used to translate linear addresses with the page number:

- The physical address corresponding to the page number (the page frame).
- The access rights from the paging-structure entries used to translate linear addresses with the page number (see Section 4.6):
  — The logical-AND of the R/W flags.
  — The logical-AND of the U/S flags.
  — The logical-OR of the XD flags (necessary only if IA32_EFER.NXE = 1).
- Attributes from a paging-structure entry that identifies the final page frame for the page number (either a PTE or a PDE in which the PS flag is 1):
  — The dirty flag (see Section 4.8).
  — The memory type (see Section 4.9).

(TLB entries may contain other information as well. A processor may implement multiple TLBs, and some of these may be for special purposes, e.g., only for instruction fetches. Such special-purpose TLBs may not contain some of this information if it is not necessary. For example, a TLB used only for instruction fetches need not contain information about the R/W and dirty flags.)

Processors need not implement any TLBs. Processors that do implement TLBs may invalidate any TLB entry at any time. Software should not rely on the existence of TLBs or on the retention of TLB entries.

4.10.1.3 Details of TLB Use

Because the TLBs cache only valid translations, there can be a TLB entry for a page number only if the P flag is 1 and the reserved bits are 0 in each of the paging-structure entries used to translate that page number. In addition, the processor does not cache a translation for a page number unless the accessed flag is 1 in each of the paging-structure entries used during translation; before caching a translation, the processor sets any of these accessed flags that is not already 1.

The processor may cache translations required for prefetches and for accesses that are a result of speculative execution that would never actually occur in the executed code path.

If the page number of a linear address corresponds to a TLB entry, the processor may use that TLB entry to determine the page frame, access rights, and other attributes for accesses to that linear address. In this case, the processor may not actually consult the paging structures in memory. The processor may retain a TLB entry unmodified even if software subsequently modifies the relevant paging-structure entries in memory. See Section 4.10.3.2 for how software can ensure that the processor uses the modified paging-structure entries.

If the paging structures specify a translation using a page larger than 4 KBytes, some processors may choose to cache multiple smaller-page TLB entries for that translation. Each such TLB entry would be associated with a page number corresponding to the smaller page size (e.g., bits 47:12 of a linear address with IA-32e paging), even though part of that page number (e.g., bits 20:12) are part of the offset with respect to the page specified by the paging structures. The upper bits of the physical address in such a TLB entry are derived from the physical address in the PDE used to create the translation, while the lower bits come from the linear address of the access for which the translation is created. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page.

If software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes, the TLBs may subsequently contain multiple translations for the address range (one for each page size). A reference to a linear address in the address range may use either translation. Which translation is used may vary from one execution to another, and the choice may be implementation-specific.

4.10.1.4 Global Pages

The Intel-64 and IA-32 architectures also allow for **global pages** when the PGE flag (bit 7) is 1 in CR4. If the G flag (bit 8) is 1 in a paging-structure entry that maps a page (either a PTE or a PDE in which the PS flag is 1), any TLB entry cached for a linear address using that paging-structure entry is considered to be **global**. Because the G flag is used only in paging-structure entries that map a page, and because information from such entries are note cached in the paging-structure caches, the global-page feature does not affect the behavior of the paging-structure caches.

4.10.2 Paging-Structure Caches

In addition to the TLBs, a processor may cache other information about the paging structures in memory.

4.10.2.1 Caches for Paging Structures

A processor may support any or of all the following paging-structure caches:

- **PML4 cache** (IA-32e paging only). Each PML4-cache entry is referenced by a 9-bit value and is used for linear addresses for which bits 47:39 have that value. The entry contains information from the PML4E used to translate such linear addresses:
  - The physical address from the PML4E (the address of the page-directory-pointer table).
  - The value of the R/W flag of the PML4E.
  - The value of the U/S flag of the PML4E.
  - The value of the XD flag of the PML4E.
  - The values of the PCD and PWT flags of the PML4E.

  The following items detail how a processor may use the PML4 cache:
  - If the processor has a PML4-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML4E in memory).
  - The processor does not create a PML4-cache entry unless the P flag is 1 and all reserved bits are 0 in the PML4E in memory.
  - The processor does not create a PML4-cache entry unless the accessed flag is 1 in the PML4E in memory; before caching a translation, the processor sets the accessed flag if it is not already 1.
  - The processor may create a PML4-cache entry even if there are no translations for any linear address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced page-directory-pointer table).
  - If the processor creates a PML4-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML4E in memory.
- **PDPTE cache** (IA-32e paging only).[1] Each PDPTE-cache entry is referenced by an 18-bit value and is used for linear addresses for which bits 47:30 have that value. The entry contains information from the PML4E and PDPTE used to translate such linear addresses:
  - The physical address from the PDPTE (the address of the page directory).
  - The logical-AND of the R/W flags in the PML4E and the PDPTE.
  - The logical-AND of the U/S flags in the PML4E and the PDPTE.
  - The logical-OR of the XD flags in the PML4E and the PDPTE.
  - The values of the PCD and PWT flags of the PDPTE.

  The following items detail how a processor may use the PDPTE cache:
  - If the processor has a PDPTE-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML4E and the PDPTE in memory).
  - The processor does not create a PDPTE-cache entry unless the P flag is 1 and the reserved bits are 0 in the PML4E and the PDPTE in memory.

1. With PAE paging, the PDPTEs are stored in internal, non-architectural registers. The operation of these registers is described in Section 4.4.1 and differs from that described here.

— The processor does not create a PDPTE-cache entry unless the accessed flags are 1 in the PML4E and the PDPTE in memory; before caching a translation, the processor sets any accessed flags that are not already 1.

— The processor may create a PDPTE-cache entry even if there are no translations for any linear address that might use that entry.

— If the processor creates a PDPTE-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML4E or PDPTE in memory.

- **PDE cache**. The use of the PDE cache depends on the paging mode:

  — For 32-bit paging, each PDE-cache entry is referenced by a 10-bit value and is used for linear addresses for which bits 31:22 have that value.

  — For PAE paging, each PDE-cache entry is referenced by an 11-bit value and is used for linear addresses for which bits 31:21 have that value.

  — For IA-32e paging, each PDE-cache entry is referenced by a 27-bit value and is used for linear addresses for which bits 47:21 have that value.

  A PDE-cache entry contains information from the PML4E, PDPTE, and PDE used to translate the relevant linear addresses (for 32-bit paging and PAE paging, only the PDE applies):

  — The physical address from the PDE (the address of the page table). (No PDE-cache entry is created for a PDE that maps a page.)

  — The logical-AND of the R/W flags in the PML4E, PDPTE, and PDE.

  — The logical-AND of the U/S flags in the PML4E, PDPTE, and PDE.

  — The logical-OR of the XD flags in the PML4E, PDPTE, and PDE.

  — The values of the PCD and PWT flags of the PDE.

  The following items detail how a processor may use the PDE cache (references below to PML4Es and PDPTEs apply on to IA-32e paging):

  — If the processor has a PDE-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML4E, the PDPTE, and the PDE in memory).

  — The processor does not create a PDE-cache entry unless the P flag is 1, the PS flag is 0, and the reserved bits are 0 in the PML4E, the PDPTE, and the PDE in memory.

  — The processor does not create a PDE-cache entry unless the accessed flag is 1 in the PML4E, the PDPTE, and the PDE in memory; before caching a translation, the processor sets any accessed flags that are not already 1.

  — The processor may create a PDE-cache entry even if there are no translations for any linear address that might use that entry.

  — If the processor creates a PDE-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML4E, the PDPTE, or the PDE in memory.

Information from a paging-structure entry can be included in entries in the paging-structure caches for other paging-structure entries referenced by the original entry. For example, if the R/W flag is 0 in a PML4E, then the R/W flag will be 0 in any PDPTE-cache entry for a PDPTE from the page-directory-pointer table referenced by that PML4E. This is because the R/W flag of each such PDPTE-cache entry is the logical-AND of the R/W flags in the appropriate PML4E and PDPTE.

The paging-structure caches contain information only from paging-structure entries that reference other paging structures (and not those that map pages). Because the G flag is not used in such paging-structure entries, the global-page feature does not affect the behavior of the paging-structure caches.

The processor may create entries in paging-structure caches for translations required for prefetches and for accesses that are a result of speculative execution that would never actually occur in the executed code path.

A processor may or may not implement any of the paging-structure caches. Software should rely on neither their presence nor their absence. The processor may invalidate entries in these caches at any time. Because the processor may create the cache entries at the time of translation and not update them following subsequent modifications to the paging structures in memory, software should take care to invalidate the cache entries appropriately when causing such modifications. The invalidation of TLBs and the paging-structure caches is described in Section 4.10.3.

4.10.2.2 Using the Paging-Structure Caches to Translate Linear Addresses

When a linear address is accessed, the processor uses a procedure such as the following to determine the physical address to which it translates and whether the access should be allowed:

- If the processor finds a TLB entry for the page number of the linear address, it may use the physical address, access rights, and other attributes from that entry.
- If the processor does not find a TLB entry, it may use the upper bits of the linear address to select an entry from the PDE cache (Section 4.10.2.1 indicates which bits are used in each paging mode). It can then use that entry to complete the translation process (locating a PTE, etc.) as if it had traversed the PDE (and, for IA-32e paging, the PDPTE and PML4) corresponding to the PDE-cache entry.
- The following items apply when IA-32e paging is used:
  - If the processor does not find a TLB entry or a PDE-cache entry, it may use bits 47:30 of the linear address to select an entry from the PDPTE cache. It can then use that entry to complete the translation process (locating a PDE, etc.) as if it had traversed the PDPTE and the PML4 corresponding to the PDPTE-cache entry.
  - If the processor does not find a TLB entry, a PDE-cache entry, or a PDPTE-cache entry, it may use bits 47:39 of the linear address to select an entry from the PML4 cache. It can then use that entry to complete the translation process (locating a PDPTE, etc.) as if it had traversed the corresponding PML4.

(Any of the above steps would be skipped if the processor does not support the cache in question.)

If the processor does not find a TLB or paging-structure-cache entry for the linear address, it uses the linear address to traverse the entire paging-structure hierarchy, as described in Section 4.3, Section 4.4.2, and Section 4.5.

4.10.2.3 Multiple Cached Entries for a Single Paging-Structure Entry

Note that multiple cached entries (in the paging-structure caches or TLBs) may contain information derived from a single paging-structure entry. The following items give some examples for IA-32e paging:

- Suppose that two PML4Es contain the same physical address and thus reference the same page-directory-pointer table. Any PDPTE in that table may result in two PDPTE-

cache entries, each associated with a different set of linear addresses. Specifically, suppose that the n_1^{th} and n_2^{th} entries in the PML4 table contain the same physical address. This implies that the physical address in the m^{th} PDPTE in the page-directory-pointer table would appear in the PDPTE-cache entries associated with both p_1 and p_2, where $(p_1 » 9) = n_1$, $(p_2 » 9) = n_2$, and $(p_1 \& 1FFH) = (p_2 \& 1FFH) = m$. This is because both PDPTE-cache entries use the same PDPTE, one resulting from a reference from the n_1^{th} PML4E and one from the n_2^{th} PML4E.

- Suppose that the first PML4E (i.e., the one in position 0) contains the physical address X in CR3 (the physical address of the PML4 table). This implies the following:
  - Any PML4-cache entry associated with linear addresses with 0 in bits 47:39 contains address X.
  - Any PDPTE-cache entry associated with linear addresses with 0 in bits 47:30 contains address X. This is because the translation for a linear address for which the value of bits 47:30 is 0 uses the value of bits 47:39 (0) to locate a page-directory-pointer table at address X (the address of the PML4 table). It then uses the value of bits 38:30 (also 0) to find address X again and to store that address in the PDPTE-cache entry.
  - Any PDE-cache entry associated with linear addresses with 0 in bits 47:21 contains address X for similar reasons.
  - Any TLB entry for page number 0 (associated with linear addresses with 0 in bits 47:12) translates to page frame X » 12 for similar reasons.

  The same PML4E contributes its address X to all these cache entries because the self-referencing nature of the entry causes it to be used as a PML4E, a PDPTE, a PDE, and a PTE.

4.10.3 Invalidation of TLBs and Paging-Structure Caches

As noted in Section 4.10.1 and Section 4.10.2, the processor may create entries in the TLBs and the paging-structure caches when linear addresses are translated, and it may retain these entries even after the paging structures used to create them have been modified. To ensure that linear-address translation uses the modified paging structures, software should take action to invalidate any cached entries that may contain information that has since been modified.

4.10.3.1 Operations that Invalidate TLBs and Paging-Structure Caches

It is recommended that software use the following instructions to invalidate entries in the TLBs and the paging-structure caches:

- INVLPG. This instruction takes a single operand, which is a linear address. The instruction invalidates any TLB entries with a page number corresponding to the linear address, including those for global pages (see Section 4.10.1.4).[1] INVLPG also invalidates all entries in all paging-structure caches regardless of the linear addresses to which they correspond.
- MOV to CR3. This instruction invalidates all TLB entries except those for global pages. It also invalidates all entries in all paging-structure caches.

1. Even if the paging structures map the linear address using a page larger than 4 KBytes and there are multiple TLB entries for that page (see Section 4.10.1), the instruction invalidates all of them.

- MOV to CR4. If this instruction changes value of the PGE flag (bit 7) of CR4, it invalidates all TLB entries and all entries in all paging-structure caches. This includes global TLB entries because (1) if CR4.PGE is changing from 0 to 1, there were no global TLB entries before the execution; and (2) if CR4.PGE is changing from 1 to 0, there will be no global TLB entries after the execution.
- Task switch. If a task switch changes the value of CR3, it invalidates all TLB entries except those for global pages. It also invalidates all entries in all paging-structure caches.
- VMX transitions. See Section 4.11.1.

The processor is always free to invalidate additional entries in the TLBs and paging-structure caches. The following are some examples:

- INVLPG may invalidate TLB entries for pages other than the one corresponding to its linear-address operand.
- MOV to CR3 may invalidate TLB entries for global pages.
- On a processor supporting Hyper-Threading Technology, invalidations performed on one logical processor may invalidate entries in the TLBs and paging-structure caches used by other logical processors.

(Other instructions and operations may invalidate entries in the TLBs and the paging-structure caches, but the instructions identified above are recommended.)

In addition to the instructions identified above, page faults invalidate entries in the TLBs and paging-structure caches. In particular, a page-fault exception resulting from an attempt to use a linear address will invalidate any PML4-cache, PDPTE-cache, and PDE-cache entries that would be used for that linear address as well as any TLB entry for that address's page number.[1] These invalidations ensure that the page-fault exception will not recur (if the faulting instruction is re-executed) if it would not be caused by the contents of the paging structures in memory (and if, therefore, it resulted from cached entries that were not invalidated after the paging structures were modified in memory).

As noted in Section 4.10.1, some processors may choose to cache multiple smaller-page TLB entries for a translation specified by the paging structures to use a page larger than 4 KBytes. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. The INVLPG instruction and page faults provide the same assurances that they provide when a single TLB entry is used: they invalidate all TLB entries corresponding to the translation specified by the paging structures.

4.10.3.2 Recommended Invalidation

The following items provide some recommendations regarding when software should perform invalidations:

- If software modifies a paging-structure entry that identifies the final page frame for a page number (either a PTE or a PDE in which the PS flag is 1), it should execute INVLPG for any linear address with a page number whose translation uses that PTE.[2] (If the paging-structure entry may be used in the translation of different page numbers — see Section 4.10.2.3 — software should execute INVLPG for linear addresses with each of those page numbers; alternatively, it could use MOV to CR3 or MOV to CR4.)

1. Unlike INVLPG, page faults need not invalidate **all** entries in the paging-structure caches, only those that would be used to translate the faulting linear address.
2. One execution of INVLPG is sufficient even for a page with size greater than 4 KBytes.

- If software modifies a paging-structure entry that references another paging structure, it may use one of the following approaches depending upon the types and number of translations controlled by the modified entry:
  - Execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry. However, if no page numbers that would use the entry have translations (e.g., because the P flags are 0 in all entries in the paging structure referenced by the modified entry), it remains necessary to execute INVLPG at least once.
  - Execute MOV to CR3 if the modified entry controls no global pages.
  - Execute MOV to CR4 to modify CR4.PGE.
- If software using PAE paging modifies a PDPTE, it should reload CR3 with the register's current value to ensure that the modified PDPTE is loaded into the corresponding PDPTE register (see Section 4.4.1).
- If the nature of the paging structures is such that a single entry may be used for multiple purposes (see Section 4.10.2.3), software should perform invalidations for all of these purposes. For example, if a single entry might serve as both a PDE and PTE, it may be necessary to execute INVLPG with two (or more) linear addresses, one that uses the entry as a PDE and one that uses it as a PTE. (Alternatively, software could use MOV to CR3 or MOV to CR4.)
- As noted in Section 4.10.1, the TLBs may subsequently contain multiple translations for the address range if software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes. A reference to a linear address in the address range may use either translation.

  Software wishing to prevent this uncertainty should not write to a paging-structure entry in a way that would change, for any linear address, both the page size and either the page frame, access rights, or other attributes. It can instead use the following algorithm: first clear the P flag in the relevant paging-structure entry (e.g., PDE); then invalidate any translations for the affected linear addresses (see Section 4.10.3.2); and then modify the relevant paging-structure entry to set the P flag and establish modified translation(s) for the new page size.

4.10.3.3 Optional Invalidation

The following items describe cases in which software may choose not to invalidate and the potential consequences of that choice:

- If a paging-structure entry is modified to change the P flag from 0 to 1, no invalidation is necessary. This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the P flag is 0.[1]
- If a paging-structure entry is modified to change the accessed flag from 0 to 1, no invalidation is necessary (assuming that an invalidation was performed the last time the accessed flag was changed from 1 to 0). This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the accessed flag is 0.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted write access) but no other adverse behavior. Such an

1. If it is also the case that no invalidation was performed the last time the P flag was changed from 1 to 0, the processor may use a TLB entry or paging-structure cache entry that was created when the P flag had earlier been 1.

exception will occur at most once for each affected linear address (see Section 4.10.3.1).

- If a paging-structure entry is modified to change the U/S flag from 0 to 1, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted user-mode access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.3.1).
- If a paging-structure entry is modified to change the XD flag from 1 to 0, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted instruction fetch) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 4.10.3.1).
- If a paging-structure entry is modified to change the accessed flag from 1 to 0, failure to perform an invalidation may result in the processor not setting that bit in response to a subsequent access to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such an access has not occurred.
- If software modifies a paging-structure entry that identifies the final physical address for a linear address (either a PTE or a PDE in which the PS flag is 1) to change the dirty flag from 1 to 0, failure to perform an invalidation may result in the processor not setting that bit in response to a subsequent write to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such a write has not occurred.
- The read of a paging-structure entry in translating an address being used to fetch an instruction may appear to execute before an earlier write to that paging-structure entry if there is no serializing instruction between the write and the instruction fetch. Note that the invalidating instructions identified in Section 4.10.3.1 are all serializing instructions.
- Section 4.10.2.3 describes situations in which a single paging-structure entry may contain information cached in multiple entries in the paging-structure caches. Because all entries in these caches are invalidated by any execution of INVLPG, it is not necessary to follow the modification of such a paging-structure entry by executing INVLPG multiple times solely for the purpose of invalidating these multiple cached entries. (It may be necessary to do so to invalidate multiple TLB entries.)

4.10.4 Propagation of Paging-Structure Changes to Multiple Processors

As noted in Section 4.10.3, software that modifies a paging-structure entry may need to invalidate entries in the TLBs and paging-structure caches that were derived from the modified entry before it was modified. In a system containing more than one logical processor, software must account for the fact that there may be entries in the TLBs and paging-structure caches of logical processors other than the one used to modify the paging-structure entry. The process of propagating the changes to a paging-structure entry is commonly referred to as "TLB shootdown."

TLB shootdown can be done using memory-based semaphores and/or interprocessor interrupts (IPI). The following items describe a simple but inefficient example of a TLB shootdown algorithm for processors supporting the Intel-64 and IA-32 architectures:

1. Begin barrier: Stop all but one logical processor; that is, cause all but one to execute the HLT instruction or to enter a spin loop.

2. Allow the active logical processor to change the necessary paging-structure entries.
3. Allow all logical processors to perform invalidations appropriate to the modifications to the paging-structure entries.
4. Allow all logical processors to resume normal operation.

Alternative, performance-optimized, TLB shootdown algorithms may be developed; however, software developers must take care to ensure that the following conditions are met:

- All logical processors that are using the paging structures that are being modified must participate and perform appropriate invalidations after the modifications are made.
- If the modifications to the paging-structure entries are made before the barrier or if there is no barrier, the operating system must ensure one of the following: (1) that the affected linear-address range is not used between the time of modification and the time of invalidation; or (2) that it is prepared to deal with the consequences of the affected linear-address range being used during that period. For example, if the operating system does not allow pages being freed to be reallocated for another purpose until after the required invalidations, writes to those pages by errant software will not unexpectedly modify memory that is in use.
- Software must be prepared to deal with reads, instruction fetches, and prefetch requests to the affected linear-address range that are a result of speculative execution that would never actually occur in the executed code path.

When multiple logical processors are using the same linear-address space at the same time, they must coordinate before any request to modify the paging-structure entries that control that linear-address space. In these cases, the barrier in the TLB shootdown routine may not be required. For example, when freeing a range of linear addresses, some other mechanism can assure no logical processor is using that range before the request to free it is made. In this case, a logical processor freeing the range can clear the P flags in the PTEs associated with the range, free the physical page frames associated with the range, and then signal the other logical processors using that linear-address space to perform the necessary invalidations. All the affected logical processors must complete their invalidations before the linear-address range and the physical page frames previously associated with that range can be reallocated.

4.11 INTERACTIONS WITH VIRTUAL-MACHINE EXTENSIONS (VMX)

The architecture for virtual-machine extensions (VMX) includes features that interact with paging. Section 4.11.1 discusses ways in which VMX-specific control transfers, called VMX transitions specially affect paging. Section 4.11.2 gives an overview of VMX features specifically designed to support address translation.

4.11.1 VMX Transitions

The VMX architecture defines two control transfers called **VM entries** and **VM exits**; collectively, these are called **VMX transitions**. VM entries and VM exits are described in detail in Chapter 23 and Chapter 24, respectively, in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*. The following items identify paging-related details:

- VMX transitions modify the CR0 and CR4 registers and the IA32_EFER MSR concurrently. For this reason, they allow transitions between paging modes that would not otherwise be possible:
  - VM entries allow transitions from IA-32e paging directly to either 32-bit paging or PAE paging.
  - VM exits allow transitions from either 32-bit paging or PAE paging directly to IA-32e paging.
- VMX transitions that result in PAE paging load the PDPTE registers (see Section 4.4.1) as follows:
  - VM entries load the PDPTE registers either from the physical address being loaded into CR3 or from the virtual-machine control structure (VMCS); see Section 23.3.2.4.
  - VM exits load the PDPTE registers from the physical address being loaded into CR3; see Section 24.5.4.
- VMX transitions invalidate the TLBs and paging-structure caches based on certain control settings. See Section 23.3.2.5 and Section 24.5.5.

4.11.2 VMX Support for Address Translation

Chapter 25, "VMX Support for Address Translation," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B* describe two features of the virtual-machine extensions (VMX) that interact directly with paging. These are **virtual-processor identifiers** (**VPIDs**) and the **extended page table** mechanism (**EPT**).

VPIDs provide a way for software to identify to the processor the address spaces for different "virtual processors." The processor may use this identification to maintain concurrently information for multiple address spaces in its TLBs and paging-structure caches. See Section 25.1 for details.

When EPT is in use, the addresses in the paging-structures are not used as physical addresses to access memory and memory-mapped I/O. Instead, they are treated as **guest-physical** addresses and are translated through a set of EPT paging structures to produce physical addresses. EPT can also specify its own access rights and memory typing; these are used on conjunction with those specified in this chapter. See Section 25.2 for more information.

Both VPIDs and EPT may change the way that a processor maintains information in TLBs and paging structure caches and the ways in which software can manage that information. Some of the behaviors documented in Section 4.10 may change. See Section 25.3 for details.

4.12 USING PAGING FOR VIRTUAL MEMORY

With paging, portions of the linear-address space need not be mapped to the physical-address space; data for the unmapped addresses can be stored externally (e.g., on disk). This method of mapping the linear-address space is referred to as virtual memory or demand-paged virtual memory.

Paging divides the linear address space into fixed-size pages that can be mapped into the physical-address space and/or external storage. When a program (or task) references a linear address, the processor uses paging to translate the linear address into a corresponding physical address if such an address is defined.

If the page containing the linear address is not currently mapped into the physical-address space, the processor generates a page-fault exception as described in Section 4.7. The handler for page-fault exceptions typically directs the operating system or executive to load data for the unmapped page from external storage into physical memory (perhaps writing a different page from physical memory out to external storage in the process) and to map it using paging (by updating the paging structures). When the page has been loaded into physical memory, a return from the exception handler causes the instruction that generated the exception to be restarted.

Paging differs from segmentation through its use of fixed-size pages. Unlike segments, which usually are the same size as the code or data structures they hold, pages have a fixed size. If segmentation is the only form of address translation used, a data structure present in physical memory will have all of its parts in memory. If paging is used, a data structure can be partly in memory and partly in disk storage.

4.13 MAPPING SEGMENTS TO PAGES

The segmentation and paging mechanisms provide in the support a wide variety of approaches to memory management. When segmentation and paging are combined, segments can be mapped to pages in several ways. To implement a flat (unsegmented) addressing environment, for example, all the code, data, and stack modules can be mapped to one or more large segments (up to 4-GBytes) that share same range of linear addresses (see Figure 3-2 in Section 3.2.2). Here, segments are essentially invisible to applications and the operating-system or executive. If paging is used, the paging mechanism can map a single linear-address space (contained in a single segment) into virtual memory. Alternatively, each program (or task) can have its own large linear-address space (contained in its own segment), which is mapped into virtual memory through its own paging structures.

Segments can be smaller than the size of a page. If one of these segments is placed in a page which is not shared with another segment, the extra memory is wasted. For example, a small data structure, such as a 1-Byte semaphore, occupies 4 KBytes if it is placed in a page by itself. If many semaphores are used, it is more efficient to pack them into a single page.

The Intel-64 and IA-32 architectures do not enforce correspondence between the boundaries of pages and segments. A page can contain the end of one segment and the beginning of another. Similarly, a segment can contain the end of one page and the beginning of another.

Memory-management software may be simpler and more efficient if it enforces some alignment between page and segment boundaries. For example, if a segment which can fit in one page is placed in two pages, there may be twice as much paging overhead to support access to that segment.

One approach to combining paging and segmentation that simplifies memory-management software is to give each segment its own page table, as shown in Figure 4-12. This convention gives the segment a single entry in the page directory, and this entry provides the access control information for paging the entire segment.

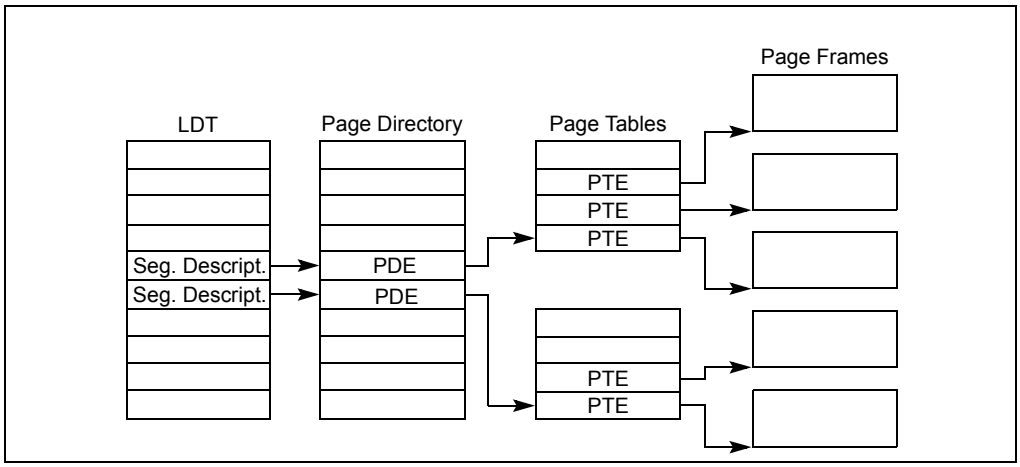

Figure 4-12 Memory Management Convention That Assigns a Page Table to Each Segment

6. Updates to Chapter 5, Volume 3A

Change bars show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

5.2 FIELDS AND FLAGS USED FOR SEGMENT-LEVEL AND PAGE-LEVEL PROTECTION

The processor's protection mechanism uses the following fields and flags in the system data structures to control access to segments and pages:

- **Descriptor type (S) flag —** (Bit 12 in the second doubleword of a segment descriptor.) Determines if the segment descriptor is for a system segment or a code or data segment.
- **Type field** — (Bits 8 through 11 in the second doubleword of a segment descriptor.) Determines the type of code, data, or system segment.
- **Limit field** — (Bits 0 through 15 of the first doubleword and bits 16 through 19 of the second doubleword of a segment descriptor.) Determines the size of the segment, along with the G flag and E flag (for data segments).
- **G flag —** (Bit 23 in the second doubleword of a segment descriptor.) Determines the size of the segment, along with the limit field and E flag (for data segments).
- **E flag —** (Bit 10 in the second doubleword of a data-segment descriptor.) Determines the size of the segment, along with the limit field and G flag.
- **Descriptor privilege level (DPL) field** — (Bits 13 and 14 in the second doubleword of a segment descriptor.) Determines the privilege level of the segment.

- **Requested privilege level (RPL) field** — (Bits 0 and 1 of any segment selector.) Specifies the requested privilege level of a segment selector.
- **Current privilege level (CPL) field** — (Bits 0 and 1 of the CS segment register.) Indicates the privilege level of the currently executing program or procedure. The term current privilege level (CPL) refers to the setting of this field.
- **User/supervisor (U/S) flag** — (Bit 2 of paging-structure entries.) Determines the type of page: user or supervisor.
- **Read/write (R/W) flag** — (Bit 1 of paging-structure entries.) Determines the type of access allowed to a page: read-only or read/write.
- **Execute-disable (XD) flag** — (Bit 63 of certain paging-structure entries.) Determines the type of access allowed to a page: executable or not-executable.

Figure 5-1 shows the location of the various fields and flags in the data, code, and system- segment descriptors; Figure 3-6 shows the location of the RPL (or CPL) field in a segment selector (or the CS register); and Chapter 4 identifies the locations of the U/S, R/W, and XD flags in the paging-structure entries.

...

5.11.1 Page-Protection Flags

Protection information for pages is contained in two flags in a paging-structure entry (see Chapter 4): the read/write flag (bit 1) and the user/supervisor flag (bit 2). The protection checks use the flags in all paging structures.

...

5.13 PAGE-LEVEL PROTECTION AND EXECUTE-DISABLE BIT

In addition to page-level protection offered by the U/S and R/W flags, paging structures used with PAE paging and IA-32e paging (see Chapter 4) provide the execute-disable bit. This bit offers additional protection for data pages.

An Intel 64 or IA-32 processor with the execute-disable bit capability can prevent data pages from being used by malicious software to execute code. This capability is provided in:

- 32-bit protected mode with PAE enabled.
- IA-32e mode.

While the execute-disable bit capability does not introduce new instructions, it does require operating systems to use a PAE-enabled environment and establish a page-granular protection policy for memory pages.

If the execute-disable bit of a memory page is set, that page can be used only as data. An attempt to execute code from a memory page with the execute-disable bit set causes a page-fault exception.

The execute-disable capability is supported only with PAE paging and IA-32e paging. It is not supported with 32-bit paging. Existing page-level protection mechanisms (see Section 5.11, “Page-Level Protection”) continue to apply to memory pages independent of the execute-disable setting.

5.13.1 Detecting and Enabling the Execute-Disable Capability

Software can detect the presence of the execute-disable capability using the CPUID instruction. CPUID.80000001H:EDX.NX [bit 20] = 1 indicates the capability is available.

If the capability is available, software can enable it by setting IA32_EFER.NXE[bit 11] to 1. IA32_EFER is available if CPUID.80000001H.EDX[bit 20 or 29] = 1.

If the execute-disable capability is not available, a write to set IA32_EFER.NXE produces a #GP exception. See Table 5-4.

...

5.13.2 Execute-Disable Page Protection

The execute-disable bit in the paging structures enhances page protection for data pages. Instructions cannot be fetched from a memory page if IA32_EFER.NXE =1 and the execute-disable bit is set in any of the paging-structure entries used to map the page. Table 5-5 lists the valid usage of a page in relation to the value of execute-disable bit (bit 63) of the corresponding entry in each level of the paging structures. Execute-disable protection can be activated using the execute-disable bit at any level of the paging structure, irrespective of the corresponding entry in other levels. When execute-disable protection is not activated, the page can be used as code or data.

...

7. Updates to Chapter 6, Volume 3A

Change bars show changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

Interrupt 14—Page-Fault Exception (#PF)

Exception Class **Fault.**

Description

Indicates that, with paging enabled (the PG flag in the CR0 register is set), the processor detected one of the following conditions while using the page-translation mechanism to translate a linear address to a physical address:

- The P (present) flag in a page-directory or page-table entry needed for the address translation is clear, indicating that a page table or the page containing the operand is not present in physical memory.
- The procedure does not have sufficient privilege to access the indicated page (that is, a procedure running in user mode attempts to access a supervisor-mode page).
- Code running in user mode attempts to write to a read-only page. In the Intel486 and later processors, if the WP flag is set in CR0, the page fault will also be triggered by code running in supervisor mode that tries to write to a read-only user-mode page.

- An instruction fetch to a linear address that translates to a physical address in a memory page with the execute-disable bit set (for information about the execute-disable bit, see Chapter 4, “Paging”).
- One or more reserved bits in page directory entry are set to 1. See description below of RSVD error code flag.

The exception handler can recover from page-not-present conditions and restart the program or task without any loss of program continuity. It can also restart the program or task after a privilege violation, but the problem that caused the privilege violation may be uncorrectable.

See also: Section 4.7, “Page-Fault Exceptions.”

Exception Error Code

Yes (special format). The processor provides the page-fault handler with two items of information to aid in diagnosing the exception and recovering from it:

- An error code on the stack. The error code for a page fault has a format different from that for other exceptions (see Figure 6-9). The error code tells the exception handler four things:
  - The P flag indicates whether the exception was due to a not-present page (0) or to either an access rights violation or the use of a reserved bit (1).
  - The W/R flag indicates whether the memory access that caused the exception was a read (0) or write (1).
  - The U/S flag indicates whether the processor was executing at user mode (1) or supervisor mode (0) at the time of the exception.
  - The RSVD flag indicates that the processor detected 1s in reserved bits of the page directory, when the PSE or PAE flags in control register CR4 are set to 1. Note:
    - The PSE flag is only available in recent Intel 64 and IA-32 processors including the Pentium 4, Intel Xeon, P6 family, and Pentium processors.
    - The PAE flag is only available on recent Intel 64 and IA-32 processors including the Pentium 4, Intel Xeon, and P6 family processors.
    - In earlier IA-32 processor, the bit position of the RSVD flag is reserved.
  - The I/D flag indicates whether the exception was caused by an instruction fetch. This flag is reserved if the processor does not support execute-disable bit or execute-disable bit feature is not enabled (see Section 4.7).

8. Updates to Chapter 9, Volume 3A

Change bars show changes to Chapter 9 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A:* System Programming Guide, Part 1.

...

9.8.5 Initializing IA-32e Mode

On Intel 64 processors, the IA32_EFER MSR is cleared on system reset. The operating system must be in protected mode with paging enabled before attempting to initialize

IA-32e mode. IA-32e mode operation also requires physical-address extensions with four levels of enhanced paging structures (see Section 4.5, "IA-32e Paging").

...

9. Updates to Chapter 11, Volume 3A

Change bars show changes to Chapter 11 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

11.5 CACHE CONTROL

...

- **PCD flag in the page-directory and page-table entries** — Controls caching for individual page tables and pages, respectively (see Section 4.9, "Paging and Memory Typing"). This flag only has effect when paging is enabled and the CD flag in control register CR0 is clear. The PCD flag enables caching of the page table or page when clear and prevents caching when set.
- **PWT flag in the page-directory and page-table entries** — Controls the write policy for individual page tables and pages, respectively (see Section 4.9, "Paging and Memory Typing"). This flag only has effect when paging is enabled and the NW flag in control register CR0 is clear. The PWT flag enables write-back caching of the page table or page when clear and write-through caching when set.
- **PCD and PWT flags in control register CR3** — Control the global caching and write policy for the page directory (see Section 2.5, "Control Registers"). The PCD flag enables caching of the page directory when clear and prevents caching when set. The PWT flag enables write-back caching of the page directory when clear and write-through caching when set. These flags do not affect the caching and write policy for individual page tables. These flags only have effect when paging is enabled and the CD flag in control register CR0 is clear.
- **G (global) flag in the page-directory and page-table entries (introduced to the IA-32 architecture in the P6 family processors)** — Controls the flushing of TLB entries for individual pages. See Section 4.10, "Caching Translation Information," for more information about this flag.
- **PGE (page global enable) flag in control register CR4** — Enables the establishment of global pages with the G flag. See Section 4.10, "Caching Translation Information," for more information about this flag.

...

11.5.2.2 Selecting Memory Types for Pentium III and More Recent Processor Families

The Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M, Pentium 4, Intel Xeon, and Pentium III processors use the PAT to select effective page-level memory types. Here, a memory type for a page is selected by the MTRRs and the value in a PAT entry that is selected with the PAT, PCD and PWT bits in a page-table or page-directory entry (see Section 11.12.3, "Selecting a Memory Type from the PAT"). Table

11-7 describes the mapping of MTRR memory types and PAT entry types to effective memory types, when normal caching is in effect (the CD and NW flags in control register CR0 are clear).
...

11.9 INVALIDATING THE TRANSLATION LOOKASIDE BUFFERS (TLBS)

The processor updates its address translation caches (TLBs) transparently to software. Several mechanisms are available, however, that allow software and hardware to invalidate the TLBs either explicitly or as a side effect of another operation. Most details are given in Section 4.10.3, "Invalidation of TLBs and Paging-Structure Caches." In addition, the following operations invalidate all TLB entries, irrespective of the setting of the G flag:

- Asserting or de-asserting the FLUSH# pin.
- (Pentium 4, Intel Xeon, and later processors only.) Writing to an MTRR (with a WRMSR instruction).
- Writing to control register CR0 to modify the PG or PE flag.
- (Pentium 4, Intel Xeon, and later processors only.) Writing to control register CR4 to modify the PSE, PGE, or PAE flag.

See Section 4.10, "Caching Translation Information," for additional information about the TLBs.

...

11.12.1 Detecting Support for the PAT Feature

An operating system or executive can detect the availability of the PAT by executing the CPUID instruction with a value of 1 in the EAX register. Support for the PAT is indicated by the PAT flag (bit 16 of the values returned to EDX register). If the PAT is supported, the operating system or executive can use the IA32_PAT MSR to program the PAT. When memory types have been assigned to entries in the PAT, software can then use of the PAT-index bit (PAT) in the page-table and page-directory entries along with the PCD and PWT bits to assign memory types from the PAT to individual pages.

Note that there is no separate flag or control bit in any of the control registers that enables the PAT. The PAT is always enabled on all processors that support it, and the table lookup always occurs whenever paging is enabled, in all paging modes.

11.12.2 IA32_PAT MSR

The IA32_PAT MSR is located at MSR address 277H (see to Appendix B, "Model-Specific Registers (MSRs)," and this address will remain at the same address on future IA-32 processors that support the PAT feature. Figure 11-9 shows the format of the 64-bit IA32_PAT MSR.

The IA32_PAT MSR contains eight page attribute fields: PA0 through PA7. The three low-order bits of each field are used to specify a memory type. The five high-order bits of each field are reserved, and must be set to all 0s. Each of the eight page attribute fields can contain any of the memory type encodings specified in Table 11-10.

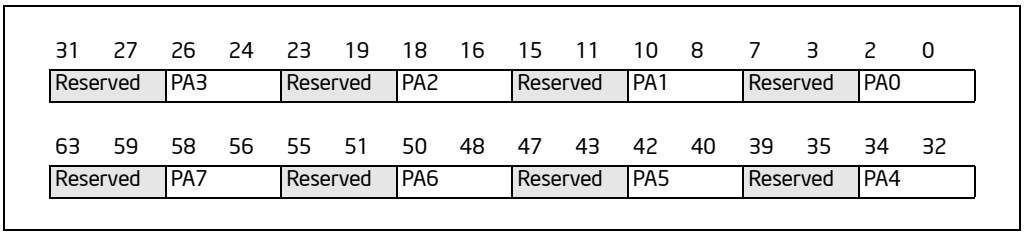

Figure 11-9 IA32_PAT MSR

Note that for the P6 family processors, the IA32_PAT MSR is named the PAT MSR.

...

11.12.3 Selecting a Memory Type from the PAT

To select a memory type for a page from the PAT, a 3-bit index made up of the PAT, PCD, and PWT bits must be encoded in the page-table or page-directory entry for the page. Table 11-11 shows the possible encodings of the PAT, PCD, and PWT bits and the PAT entry selected with each encoding. The PAT bit is bit 7 in page-table entries that point to 4-KByte pages and bit 12 in paging-structure entries that point to larger pages. The PCD and PWT bits are bits 4 and 3, respectively, in paging-structure entries that point to pages of any size.

...

11.12.4Programming the PAT

...

The values in all the entries of the PAT can be changed by writing to the IA32_PAT MSR using the WRMSR instruction. The IA32_PAT MSR is read and write accessible (use of the RDMSR and WRMSR instructions, respectively) to software operating at a CPL of 0. Table 11-10 shows the allowable encoding of the entries in the PAT. Attempting to write an undefined memory type encoding into the PAT causes a general-protection (#GP) exception to be generated.

...

11.12.5 PAT Compatibility with Earlier IA-32 Processors

For IA-32 processors that support the PAT, the IA32_PAT MSR is always active. That is, the PCD and PWT bits in page-table entries and in page-directory entries (that point to pages) are always select a memory type for a page indirectly by selecting an entry in the PAT. They never select the memory type for a page directly as they do in earlier IA-32 processors that do not implement the PAT (see Table 11-6).

...

10. Updates to Chapter 15, Volume 3A

Change bars show changes to Chapter 15 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

15.1 MACHINE-CHECK ARCHITECTURE

...

Starting with 45nm Intel 64 processor with CPUID signature DisplayFamily_DisplayModel encoding of 06H_1AH (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-M" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*), the processor can report information on corrected machine-check errors and deliver a programmable interrupt for software to respond to MC errors, referred to as corrected machine-check error interrupt (CMCI). See Section 15.5 for detail.

Intel 64 processors supporting machine-check architecture and CMCI may also support an additional enhancement, namely, support for software recovery from certain uncorrected recoverable machine check errors. See Section 15.6 for detail.

...

15.3.1.1 IA32_MCG_CAP MSR

The IA32_MCG_CAP MSR is a read-only register that provides information about the machine-check architecture of the processor. Figure 15-2 shows the structure of the register in Pentium 4, Intel Xeon, and P6 family processors.

...

- **MCG_SER_P (software error recovery support present) flag, bit 24**— Indicates (when set) that the processor supports software error recovery (see Section 15.6), and IA32_MCi_STATUS MSR bits 56:55 are used to report the signaling of uncorrected recoverable errors and whether software must take recovery actions for uncorrected errors. Note that when MCG_TES_P is not set, bits 56:53 of the IA32_MCi_STATUS MSR are model-specific. If MCG_TES_P is set but MCG_SER_P is not set, bits 56:55 are reserved.

The effect of writing to the IA32_MCG_CAP MSR is undefined.

15.3.2.2 IA32_MCi_STATUS MSRS

Each IA32_MCi_STATUS MSR contains information related to a machine-check error if its VAL (valid) flag is set (see Figure 15-5). Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.

NOTE

Figure 15-5 depicts the IA32_MCi_STATUS MSR when IA32_MCG_CAP[24] = 1, IA32_MCG_CAP[11] = 1 and IA32_MCG_CAP[10] = 1. When IA32_MCG_CAP[24] = 0 and IA32_MCG_CAP[11] = 1, bits 56:55 is reserved and bits 54:53 for threshold-based error reporting. When IA32_MCG_CAP[11] = 0, bits 56:53 are part of the "Other Information" field. The use of bits 54:53 for threshold-based error reporting began with Intel Core Duo processors, and is currently used for cache memory. See Section 15.4, "Enhanced Cache Error reporting," for more information. When IA32_MCG_CAP[10] = 0, bits 52:38 are part of the "Other Information" field. The use of bits 52:38 for corrected MC error count is introduced with Intel 64 processor having CPUID DisplayFamily_DisplayModel encoding of 06H_1AH.

...

- If IA32_MCG_CAP[11] is 1, bits **56:53** are architectural (not model-specific). In this case, bits 56:53 have the following functionality:
  - If IA32_MCG_CAP[24] is 0, bits 56:55 are reserved.
  - If IA32_MCG_CAP[24] is 1, bits 56:55 are defined as follows:
  - S (Signaling) flag, bit 56 - Signals the reporting of UCR errors in this MC bank. See Section 15.6.2 for additional detail.
  - AR (Action Required) flag, bit 55 - Indicates (when set) that MCA error code specific recovery action must be performed by system software at the time this error was signaled. See Section 15.6.2 for additional detail.
  - If the UC bit (Figure 15-5) is 1, bits 54:53 are undefined.

...

15.3.2.4 IA32_MCi_MISC MSRs

The IA32_MC*i*_MISC MSR contains additional information describing the machine-check error if the MISCV flag in the IA32_MC*i*_STATUS register is set. The IA32_MCi_MISC_MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MCi_STATUS register is clear.

When not implemented in the processor, all reads and writes to this MSR will cause a general protection exception. When implemented in a processor, these registers can be cleared by explicitly writing all 0s to them; writing 1s to them causes a general-protection exception to be generated. This register is not implemented in any of the error-reporting register banks for the P6 family processors.

If both MISCV and IA32_MCG_CAP[24] are set, the IA32_MCi_MISC_MSR is defined according to Figure 15-7 to support software recovery of uncorrected errors (see Section 15.6):

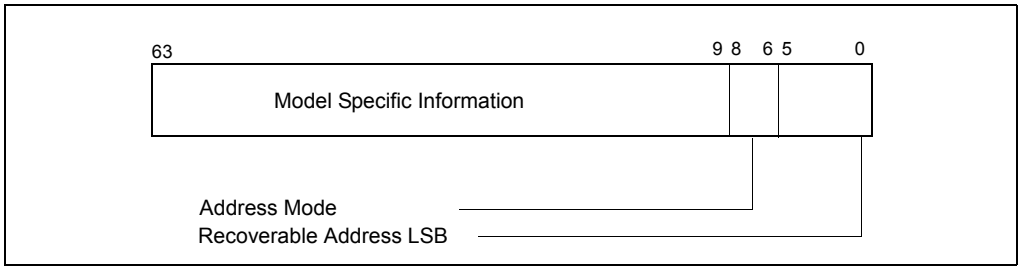

Figure 15-7 UCR Support in IA32_MCi_MISC Register

- Recoverable Address LSB (bits 5:0): The lowest valid recoverable address bit. Indicates the position of the least significant bit (LSB) of the recoverable error address. For example, if the processor logs bits [43:9] of the address, the LSB sub-field in IA32_MCi_MISC is 01001b (9 decimal). For this example, bits [8:0] of the recoverable error address in IA32_MCi_ADDR should be ignored.
- Address Mode (bits 8:6): Address mode for the address logged in IA32_MCi_ADDR. The supported address modes are given in Table Table 15-3..

Table 15-3. Address Mode in IA32_MCi_MISC[8:6]

IA32_MCi_MISC[8:6] Encoding	Definition
000	Segment Offset
001	Linear Address
010	Physical Address
011	Memory Address
100 to 110	Reserved
111	Generic

- Model Specific Information (bits 63:9): Not architecturally defined.

…

15.3.2.5 IA32_MCi_CTL2 MSRs

…

- **Corrected error count threshold, bits 14:0** — Software must initialize this field. The value is compared with the corrected error count field in IA32_MCi_STATUS, bits 38 through 52. An overflow event is signaled to the CMCI LVT entry (see Table 10-1) in the APIC when the count value equals the threshold value. The new LVT entry in the APIC is at 02F0H offset from the APIC_BASE. If CMCI interface is not supported for a particular bank (but IA32_MCG_CAP[10] = 1), this field will always read 0.

…

15.6 RECOVERY OF UNCORRECTED RECOVERABLE (UCR) ERRORS

Recovery of uncorrected recoverable machine check errors is an enhancement in machine-check architecture. The first processor that supports this feature is 45nm Intel 64 processor with CPUID signature DisplayFamily_DisplayModel encoding of 06H_2EH. This allow system software to perform recovery action on certain class of uncorrected errors and continue execution.

15.6.1 Detection of Software Error Recovery Support

Software must use bit 24 of IA32_MCG_CAP (MCG_SER_P) to detect the presence of software error recovery support (see Figure 15-2). When IA32_MCG_CAP[24] is set, this indicates that the processor supports software error recovery. When this bit is clear, this indicates that there is no support for error recovery from the processor and the primary responsibility of the machine check handler is logging the machine check error information and shutting down the system.

The new class of architectural MCA errors from which system software can attempt recovery is called Uncorrected Recoverable (UCR) Errors. UCR errors are uncorrected errors that have been detected and signaled but have not corrupted the processor context. For certain UCR errors, this means that once system software has performed a certain recovery action, it is possible to continue execution on this processor. UCR error reporting provides an error containment mechanism for data poisoning. The machine check handler will use the error log information from the error reporting registers to analyze and implement specific error recovery actions for UCR errors.

15.6.2 UCR Error Reporting and Logging

IA32_MCi_STATUS MSR is used for reporting UCR errors and existing corrected or uncorrected errors. The definitions of IA32_MCi_STATUS, including bit fields to identify UCR errors, is shown in Figure 15-5. UCR errors can be signaled through either the corrected machine check interrupt (CMCI) or machine check exception (MCE) path depending on the type of the UCR error.

When IA32_MCG_CAP[24] is set, a UCR error is indicated by the following bit settings in the IA32_MCi_STATUS register:

- Valid (bit 63) = 1
- UC (bit 61) = 1
- PCC (bit 57) = 0

Additional information from the IA32_MCi_MISC and the IA32_MCi_ADDR registers for the UCR error are available when the ADDRV and the MISCV flags in the IA32_MCi_STATUS register are set (see Section 15.3.2.4). The MCA error code field of the IA32_MCi_STATUS register indicates the type of UCR error. System software can interpret the MCA error code field to analyze and identify the necessary recovery action for the given UCR error.

In addition, the IA32_MCi_STATUS register bit fields, bits 56:55, are defined (see Figure 15-5) to provide additional information to help system software to properly identify the necessary recovery action for the UCR error:

- S (Signaling) flag, bit 56 - Indicates (when set) that a machine check exception was generated for the UCR error reported in this MC bank and system software needs to check the AR flag and the MCA error code fields in the IA32_MCi_STATUS register to identify the necessary recovery action for this error. When the S flag in the IA32_MCi_STATUS register is clear, this UCR error was not signaled via a machine check exception and instead was reported as a corrected machine check (CMC). System software is not required to take any recovery action when the S flag in the IA32_MCi_STATUS register is clear.
- AR (Action Required) flag, bit 55 - Indicates (when set) that MCA error code specific recovery action must be performed by system software at the time this error was signaled. This recovery action must be completed successfully before any additional work is scheduled for this processor When the RIPV flag in the IA32_MCG_STATUS is clear, an alternative execution stream needs to be provided; when the MCA error code specific recovery specific recovery action cannot be successfully completed, system software must shut down the system. When the AR flag in the IA32_MCi_STATUS register is clear, system software may still take MCA error code specific recovery action but this is optional; system software can safely resume program execution at the instruction pointer saved on the stack from the machine check exception when the RIPV flag in the IA32_MCG_STATUS register is set.

Both the S and the AR flags in the IA32_MCi_STATUS register are defined to be sticky bits, which mean that once set, the processor does not clear them. Only software and good power-on reset can clear the S and the AR-flags. Both the S and the AR flags are only set when the processor reports the UCR errors (MCG_CAP[24] is set).

15.6.3 UCR Error Classification

With the S and AR flag encoding in the IA32_MCi_STATUS register, UCR errors can be classified as:

- Uncorrected no action required (UCNA) - is a UCR error that is not signaled via a machine check exception and, instead, is reported to system software as a corrected machine check error. UCNA errors indicate that some data in the system is corrupted, but the data has not been consumed and the processor state is valid and you may continue execution on this processor. UCNA errors require no action from system software to continue execution. A UNCA error is indicated with UC=1, PCC=0, S=0 and AR=0 in the IA32_MCi_STATUS register.
- software recoverable action optional (SRAO) - a UCR error is signaled via a machine check exception and a system software recovery action is optional and not required to continue execution from this machine check exception. SRAO errors indicate that some data in the system is corrupt, but the data has not been consumed and the processor state is valid. SRAO errors provide the additional error information for system software to perform a recovery action. An SRAO error is indicated with UC=1, PCC=0, S=1, EN=1 and AR=0 in the IA32_MCi_STATUS register. Recovery actions for SRAO errors are MCA error code specific. The MISCV and the ADDRV flags in the IA32_MCi_STATUS register are set when the additional error information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. System software needs to inspect the MCA error code fields in the IA32_MCi_STATUS register to identify the specific recovery action for a given SRAO error. If MISCV and ADDRV are

not set, it is recommended that no system software error recovery be performed however, you can resume execution.

- software recoverable action required (SRAR) - a UCR error that requires system software to take a recovery action on this processor before scheduling another stream of execution on this processor. SRAR errors indicate that the error was detected and raised at the point of the consumption in the execution flow. An SRAR error is indicated with UC=1, PCC=0, S=1, EN=1 and AR=1 in the IA32_MCi_STATUS register. Recovery actions are MCA error code specific. The MISCV and the ADDRV flags in the IA32_MCi_STATUS register are set when the additional error information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. System software needs to inspect the MCA error code fields in the IA32_MCi_STATUS register to identify the specific recovery action for a given SRAR error. If MISCV and ADDRV are not set, it is recommended that system software shutdown the system.

Table Table 15-6. summarizes UCR, corrected, and uncorrected errors.

Table 15-6. MC Error Classifications

Type of Error[1]	UC	PCC	S	AR	Signaling	Software Action	Example
Uncorrected Error (UC)	1	1	x	x	MCE	Reset the system	
SRAR	1	0	1	1	MCE	For known MCACOD, take specific recovery action; For unknown MCACOD, must bugcheck	Cache to processor load error
SRAO	1	0	1	0	MCE	For known MCACOD, take specific recovery action; For unknown MCACOD, OK to keep the system running	Patrol scrub and explicit writeback poison errors
UCNA	1	0	0	0	CMC	Log the error and Ok to keep the system running	Poison detection error
Corrected Error (CE)	0	0	x	x	CMC	Log the error and no corrective action required	ECC in caches and memory

NOTES:
1. VAL=1, EN=1 for UC=1 errors; OVER=0 for UC=1 and PCC=0 errors SRAR, SRAO and UCNA errors are supported by the processor only when IA32_MCG_CAP[24] (MCG_SER_P) is set.

15.6.4 UCR Error Overwrite Rules

In general, the overwrite rules are as follows:

- UCR errors will overwrite corrected errors.
- Uncorrected (PCC=1) errors overwrite UCR (PCC=0) errors.
- UCR errors are not written over previous UCR errors.
- Corrected errors do not write over previous UCR errors.

Regardless of whether the 1st error is retained or the 2nd error is overwritten over the 1st error, the OVER flag in the IA32_MCi_STATUS register will be set to indicate an overflow condition. As the S flag and AR flag in the

IA32_MCi_STATUS register are defined to be sticky flags, a second event cannot clear these 2 flags once set, however the MC bank information may be filled in for the 2nd error. The table below shows the overwrite rules and how to treat a second error if the first event is already logged in a MC bank along with the resulting bit setting of the UC, PCC, and AR flags in the IA32_MCi_STATUS register. As UCNA and SRA0 errors do not require recovery action from system software to continue program execution, a system reset by system software is not required unless the AR flag or PCC flag is set for the UCR overflow case (OVER=1, VAL=1, UC=1, PCC=0).

Table Table 15-7. lists overwrite rules for uncorrected errors, corrected errors, and UCR errors.

Table 15-7. Overwrite Rules for UC, CE, and UCR Errors

First Event	Second Event	UC	PCC	S	AR	MCA Bank	Reset System
CE	UCR	1	0	0 if UCNA, else 1	1 if SRAR, else 0	second	yes, if AR=1
UCR	CE	1	0	0 if UCNA, else 1	1 if SRAR, else 0	first	yes, if AR=1
UCNA	UCNA	1	0	0	0	first	no
UCNA	SRAO	1	0	1	0	first	no
UCNA	SRAR	1	0	1	1	first	yes
SRAO	UCNA	1	0	1	0	first	no
SRAO	SRAO	1	0	1	0	first	no
SRAO	SRAR	1	0	1	1	first	yes
SRAR	UCNA	1	0	1	1	first	yes
SRAR	SRAO	1	0	1	1	first	yes
SRAR	SRAR	1	0	1	1	first	yes
UCR	UC	1	1	undefined	undefined	second	yes
UC	UCR	1	1	undefined	undefined	first	yes

...

15.9.3 Architecturally Defined UCR Errors

Software recoverable compound error code are defined in this section.

5.9.3.1 Architecturally Defined SRAO Errors

The following two SRAO errors are architecturally defined.

- UCR Errors detected by memory controller scrubbing; and
- UCR Errors detected during L3 cache (L3) explicit writebacks.

The MCA error code encodings for these two architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 15-9). Their values and compound encoding format are given in Table Table 15-15..

Table 15-15. MCA Compound Error Code Encoding for SRAO Errors

Type	MCACOD Value	MCA Error Code Encoding[1]
Memory Scrubbing	0xC0 - 0xCF	0000_0000_1100_CCCC 000F 0000 1MMM CCCC (Memory Controller Error), where Memory subfield MMM = 100B (memory scrubbing) Channel subfield CCCC = channel # or generic
L3 Explicit Writeback	0x17A	0000_0001_0111_1010 000F 0001 RRRR TTLL (Cache Hierarchy Error) where Request subfields RRRR = 0111B (Eviction) Transaction Type subfields TT = 10B (Generic) Level subfields LL = 10B

NOTES:
1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error is 0, indicating "normal" filtering.

Table Table 15-16. lists values of relevant bit fields of IA32_MCi_STATUS for architecturally defined SRAO errors.

Table 15-16. IA32_MCi_STATUS Values for SRAO Errors

SRAO Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR	MCACOD
Memory Scrubbing	1	0	1	1	1	1	0	1	0	0xC0-0xCF
L3 Explicit Writeback	1	0	1	1	1	1	0	1	0	0x17A

For both the memory scrubbing and L3 explicit writeback errors, the ADDRV and MISCV flags in the IA32_MCi_STATUS register are set to indicate that the offending physical address information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. For the memory scrubbing and L3 explicit writeback errors, the address mode in the IA32_MCi_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32_MCi_MISC register should indicate the lowest valid address bit in the address information provided from the IA32_MCi_ADDR register.

An MCE signal is broadcast to all logical processors on the system on which the UCR errors are supported. MCi_STATUS banks can be shared by logical processors within a core or within the same package. So several logical processors may find an SRAO error in the shared IA32_MCi_STATUS bank but other processors do not find it in any of the IA32_MCi_STATUS banks. Table Table 15-17. shows the RIPV and EIPV flag indication in the

IA32_MCG_STATUS register for the memory scrubbing and L3 explicit write-back errors on both the reporting and non-reporting logical processors.

Table 15-17. IA32_MCG_STATUS Flag Indication for SRAO Errors

SRAO Type	Reporting Logical Processors		Non-reporting Logical Processors	
	RIPV	EIPV	RIPV	EIPV
Memory Scrubbing	1	0	1	0
L3 Explicit Writeback	1	0	1	0

5.9.3.2 Architecturally Defined SRAR Errors

The following two SRAR errors are architecturally defined.

- UCR Errors detected on data load; and
- UCR Errors detected on instruction fetch.

The MCA error code encodings for these two architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 15-9). Their values and compound encoding format are given in Table Table 15.18..

Table 15.18. MCA Compound Error Code Encoding for SRAR Errors

Type	MCACOD Value	MCA Error Code Encoding[1]
Data Load	0x134	0000_0001_0011_0100 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0011B (Data Load) Transaction Type subfield TT= 01B (Data) Level subfield LL = 00B (Level 0)
Instruction Fetch	0x150	0000_0001_0101_0000 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0101B (Instruction Fetch) Transaction Type subfield TT= 00B (Instruction) Level subfield LL = 00B (Level 0)

NOTES:
1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error is 0, indicating "normal" filtering.

Table Table 15.19. lists values of relevant bit fields of IA32_MCi_STATUS for architecturally defined SRAR errors.

Table 15.19. IA32_MCi_STATUS Values for SRAR Errors

SRAR Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR	MCACOD
Data Load	1	0	1	1	1	1	0	1	1	0x134
Instruction Fetch	1	0	1	1	1	1	0	1	1	0x150

For both the data load and instruction fetch errors, the ADDRV and MISCV flags in the IA32_MCi_STATUS register are set to indicate that the offending physical address information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. For the memory scrubbing and L3 explicit writeback errors, the address mode in the IA32_MCi_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32_MCi_MISC register should indicate the lowest valid address bit in the address information provided from the IA32_MCi_ADDR register.

An MCE signal is broadcast to all logical processors on the system on which the UCR errors are supported. The IA32_MCG_STATUS MSR allows system software to distinguish the affected logical processor of an SRAR error amongst logical processors that observed SRAR via a shared MCi_STATUS bank.

Table Table 15-20. shows the RIPV and EIPV flag indication in the IA32_MCG_STATUS register for the data load and instruction fetch errors on both the reporting and non-reporting logical processors.

Table 15-20. IA32_MCG_STATUS Flag Indication for SRAR Errors

SRAR Type	Affected Logical Processors		Non-Affected Logical Processors	
	RIPV	EIPV	RIPV	EIPV
Data Load	0	1	1	0
instruction Fetch	0	0	1	0

The affected logical processor is the one that has detected and raised an SRAR error at the point of the consumption in the execution flow. The affected logical processor should find the Data Load or the Instruction Fetch error information in the IA32_MCi_STATUS register that is reporting the SRAR error.

For Data Load recoverable errors, the affected logical processor should find that the IA32_MCG_STATUS.RIPV flag is cleared and the IA32_MCG_STATUS.EIPV flag is set indicating that the error is detected at the instruction pointer saved on the stack for this machine check exception and restarting execution with the interrupted context is not possible.

For Instruction Fetch recoverable error, the affected logical processor should find that the RIPV flag and the EIPV Flag in the IA32_MCG_STATUS register are cleared, indicating that the error is detected at the instruction pointer saved on the stack may not be associated with this error and restarting the execution with the interrupted context is not possible.

The logical processors that observed but not affected by an SRAR error should find that the RIPV flag in the IA32_MCG_STATUS register is set and the EIPV flag in the IA32_MCG_STATUS register is cleared, indicating that it is safe to restart the execution at the instruction saved on the stack for the machine check exception on these processors after the recovery action is successfully taken by system software.

For the Data-Load and the Instruction-Fetch recoverable errors, system software may take the following recovery actions for the affected logical processor:

- The current executing thread cannot be continued. You must terminate the interrupted stream of execution and provide a new stream of execution on return from the machine check handler for the affected logical processor.

In addition to taking the recovery action described above, system software may also need to disable the use of the affected page from the program. This recovery action by system software may prevent the occurrence of future consumption errors from that affected page.

15.9.4 Multiple MCA Errors

When multiple MCA errors are detected within a certain detection window, the processor may aggregate the reporting of these errors together as a single event, i.e. a single machine exception condition. If this occurs, system soft-

ware may find multiple MCA errors logged in different MC banks on one logical processor or find multiple MCA errors logged across different processors for a single machine check broadcast event. In order to handle multiple UCR errors reported from a single machine check event and possibly recover from multiple errors, system software may consider the following:

- Whether it can recover from multiple errors is determined by the most severe error reported on the system. If the most severe error is found to be an unrecoverable error (VAL=1, UC=1, PCC=1 and EN=1) after system software examines the MC banks of all processors to which the MCA signal is broadcast, recovery from the multiple errors is not possible and system software needs to reset the system.
- When multiple recoverable errors are reported and no other fatal condition (e.g.. overflowed condition for SRAR error) is found for the reported recoverable errors, it is possible for system software to recover from the multiple recoverable errors by taking necessary recovery action for each individual recoverable error. However, system software can no longer expect one to one relationship with the error information recorded in the IA32_MCi_STATUS register and the states of the RIPV and EIPV flags in the IA32_MCG_STATUS register as the states of the RIPV and the EIPV flags in the IA32_MCG_STATUS register may indicate the information for the most severe error recorded on the processor. System software is required to use the RIPV flag indication in the IA32_MCG_STATUS register to make a final decision of recoverability of the errors and find the restart-ability requirement after examining each IA32_MCi_STATUS register error information in the MC banks.

...

15.10 GUIDELINES FOR WRITING MACHINE-CHECK SOFTWARE

The machine-check architecture and error logging can be used in three different ways:

- To detect machine errors during normal instruction execution, using the machine-check exception (#MC).
- To periodically check and log machine errors.
- To examine recoverable UCR errors, determine software recoverability and perform recovery actions via a machine-check exception handler or a corrected machine-check interrupt handler.

...

15.10.4 Machine-Check Software Handler Guidelines for Error Recovery

15.10.4.1 Machine-Check Exception Handler for Error Recovery

When writing a machine-check exception (MCE) handler to support software recovery from Uncorrected Recoverable (UCR) errors, consider the following:

- When IA32_MCG_CAP [24] is zero, there are no recoverable errors supported and all machine-check are fatal exceptions. The logging of status and error information is therefore a baseline implementation requirement.

- When IA32_MCG_CAP [24] is 1, certain uncorrected errors called uncorrected recoverable (UCR) errors may be software recoverable. The handler can analyze the reported error information, and in some cases attempt to recover from the uncorrected error and continue execution.
- For processors with DisplayFamily_DisplayModel encoding of 06H_EH and above, a MCA signal is broadcast to all logical processors in the system. Due to the potentially shared machine check MSR resources among the logical processors on the same package/core, the MCE handler may be required to synchronize with the other processors that received a machine check error and serialize access to the machine check registers when analyzing, logging and clearing the information in the machine check registers.
- The VAL (valid) flag in each IA32_MCi_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank do not contain valid error information and should not be checked.
- The MCE handler is primarily responsible for processing uncorrected errors. The UC flag in each IA32_MCi_Status register indicates whether the reported error was corrected (UC=0) or uncorrected (UC=1). The MCE handler can optionally log and clear the corrected errors in the MC banks if it can implement software algorithm to avoid the undesired race conditions with the CMCI or CMC polling handler.
- For uncorrectable errors, the EIPV flag in the IA32_MCG_STATUS register indicates (when set) that the instruction pointed to by the instruction pointer pushed onto the stack when the machine-check exception is generated is directly associated with the error. When this flag is cleared, the instruction pointed to may not be associated with the error.
- The MCIP flag in the IA32_MCG_STATUS register indicates whether a machine-check exception was generated. When a machine check exception is generated, it is expected that the MCIP flag in the IA32_MCG_STATUS register is set to 1. If it is not set, this machine check was generated by either an INT 18 instruction or some piece of hardware signaling an interrupt with vector 18.

When IA32_MCG_CAP [24] is 1, the following rules can apply when writing a machine check exception (MCE) handler to support software recovery:

- The PCC flag in each IA32_MCi_STATUS register indicates whether recovery from the error is possible for uncorrected errors (UC=1). If the PCC flag is set for uncorrected errors (UC=1), recovery is not possible. When recovery is not possible, the MCE handler typically records the error information and signals the operating system to reset the system.
- The RIPV flag in the IA32_MCG_STATUS register indicates whether restarting the program execution from the instruction pointer saved on the stack for the machine check exception is possible. When the RIPV is set, program execution can be restarted reliably when recovery is possible. If the RIPV flag is not set, program execution cannot be restarted reliably. In this case the recovery algorithm may involve terminating the current program execution and resuming an alternate thread of execution upon return from the machine check handler when recovery is possible. When recovery is not possible, the MCE handler signals the operating system to reset the system.
- When the EN flag is zero but the VAL and UC flags are one in the IA32_MCi_STATUS register, the reported uncorrected error in this bank is not enabled. As uncorrected errors with the EN flag = 0 are not the source of machine check exceptions, the MCE handler should log and clear non-enabled errors when the S bit is set and should continue searching for enabled errors from the other IA32_MCi_STATUS registers. Note that when IA32_MCG_CAP [24] is 0, any uncorrected error condition (VAL =1

and UC=1) including the one with the EN flag cleared are fatal and the handler must signal the operating system to reset the system. For the errors that do not generate machine check exceptions, the EN flag has no meaning. See Appendix A: Table A-4 to find the errors that do not generate machine check exceptions.

When the VAL flag is one, the UC flag is one, the EN flag is one and the PCC flag is zero in the IA32_MCi_STATUS register, the error in this bank is an uncorrected recoverable (UCR) error. The MCE handler needs to examine the S flag and the

- AR flag to find the type of the UCR error for software recovery and determine if software error recovery is possible.
- When both the S and the AR flags are clear in the IA32_MCi_STATUS register for the UCR error (VAL=1, UC=1, EN=x and PCC=0), the error in this bank is an uncorrected no-action required error (UCNA). UCNA errors are uncorrected but do not require any OS recovery action to continue execution. These errors indicate that some data in the system is corrupt, but that data has not been consumed and may not be consumed. If that data is consumed a non-UNCA machine check exception will be generated. UCNA errors are signaled in the same way as corrected machine check errors and the CMCI and CMC polling handler is primarily responsible for handling UCNA errors. Like corrected errors, the MCA handler can optionally log and clear UCNA errors as long as it can avoid the undesired race condition with the CMCI or CMC polling handler. As UCNA errors are not the source of machine check exceptions, the MCA handler should continue searching for uncorrected or software recoverable errors in all other MC banks.
- When the S flag in the IA32_MCi_STATUS register is set for the UCR error ((VAL=1, UC=1, EN=1 and PCC=0), the error in this bank is software recoverable and it was signaled through a machine-check exception. The AR flag in the IA32_MCi_STATUS register further clarifies the type of the software recoverable errors.
- When the AR flag in the IA32_MCi_STATUS register is clear for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action optional (SRAO) error. The MCE handler and the operating system can analyze the IA32_MCi_STATUS [15:0] to implement MCA error code specific optional recovery action, but this recovery action is optional. System software can resume the program execution from the instruction pointer saved on the stack for the machine check exception when the RIPV flag in the IA32_MCG_STATUS register is set.
- When the OVER flag in the IA32_MCi_STATUS register is set for the SRAO error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=0), the MCE handler cannot take recovery action as the information of the SRAO error in the IA32_MCi_STATUS register was potentially lost due to the overflow condition. Since the recovery action for SRAO errors is optional, restarting the program execution from the instruction pointer saved on the stack for the machine check exception is still possible for the overflowed SRAO error if the RIPV flag in the IA32_MCG_STATUS is set.
- When the AR flag in the IA32_MCi_STATUS register is set for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action required (SRAR) error. The MCE handler and the operating system must take recovery action in order to continue execution after the machine-check exception. The MCA handler and the operating system need to analyze the IA32_MCi_STATUS [15:0] to determine the MCA error code specific recovery action. If no recovery action can be performed, the operating system must reset the system.
- When the OVER flag in the IA32_MCi_STATUS register is set for the SRAR error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=1), the MCE handler cannot take recovery action as the information of the SRAR error in the IA32_MCi_STATUS

register was potentially lost due to the overflow condition. Since the recovery action for SRAR errors must be taken, the MCE handler must signal the operating system to reset the system.

- When the MCE handler cannot find any uncorrected (VAL=1, UC=1 and EN=1) or any software recoverable errors (VAL=1, UC=1, EN=1, PCC=0 and S=1) in any of the IA32_MCi banks of the processors, this is an unexpected condition for the MCE handler and the handler should signal the operating system to reset the system.
- Before returning from the machine-check exception handler, software must clear the MCIP flag in the IA32_MCG_STATUS register. The MCIP flag is used to detect recursion. The machine-check architecture does not support recursion. When the processor receives a machine check when MCIP is set, it automatically enters the shutdown state.

Example Example 15-4. gives pseudocode for an MC exception handler that supports recovery of UCR.

Example 15-4. Machine-Check Error Handler Pseudocode Supporting UCR

```
MACHINE CHECK HANDLER:  (* Called from INT 18 handler *)
NOERROR = TRUE;
ProcessorCount = 0;
IF CPU supports MCA
    THEN
        RESTARTABILITY = TRUE;
        IF (Processor Family = 6 AND Display_Model >= 0EH) OR (Processor Family > 6)
            THEN
                MCA_BROADCAST = TRUE;
                Acquire SpinLock;
                ProcessorCount++;  (* Allowing one logical processor at a time to examine machine check registers *)
                CALL MCA ERROR PROCESSING; (* returns RESTARTABILITY and NOERROR *)
            ELSE
                MCA_BROADCAST = FALSE;
                (* Implement a rendezvous mechanism with the other processors if necessary *)
                CALL MCA ERROR PROCESSING;
        FI;
    ELSE (* Pentium(R) processor compatible *)
        READ P5_MC_ADDR
        READ P5_MC_TYPE;
        RESTARTABILITY = FALSE;
FI;

IF NOERROR = TRUE
    THEN
        IF NOT (MCG_RIPV = 1 AND MCG_EIPV = 0)
            THEN
RESTARTABILITY = FALSE;
        FI
FI;

IF RESTARTABILITY = FALSE
    THEN
        Report RESTARTABILITY to console;
        Reset system;
FI;

IF MCA_BROADCAST = TRUE
    THEN
        IF ProcessorCount = MAX_PROCESSORS
```

```
            AND NOERROR = TRUE
              THEN
                  Report RESTARTABILITY to console;
                  Reset system;
          FI;
          Release SpinLock;
          Wait till ProcessorCount = MAX_PROCESSRS on system;
          (* implement a timeout and abort function if necessary *)
FI;
CLEAR MCIP flag in IA32_MCG_STATUS;
RESUME Execution;
(* End of MACHINE CHECK HANDLER*)


MCA ERROR PROCESSING:    (* MCA Error Processing Routine called from MCA Handler *)
IF MCIP flag in IA32_MCG_STATUS = 0
    THEN (* MCIP=0 upon MCA is unexpected *)
        RESTARTABILITY = FALSE;
FI;
FOR each bank of machine-check registers
    DO
        CLEAR_MC_BANK = FALSE;
        READ IA32_MCi_STATUS;
        IF VAL Flag in IA32_MCi_STATUS = 1
            THEN
                IF UC Flag in IA32_MCi_STATUS = 1
                    THEN
                        IF Bit 24 in IA32_MCG_CAP = 0
                            THEN (* the processor does not support software error recovery *)
                                RESTARTABILITY = FALSE;
                                NOERROR = FALSE;
                                GOTO LOG MCA REGISTER;
                        FI;
                        (* the processor supports software error recovery *)
                        IF EN Flag in IA32_MCi_STATUS = 0 AND OVER Flag in IA32_MCi_STATUS=0
                            THEN (* It is a spurious MCA Log. Log and clear the register *)
                                CLEAR_MC_BANK = TRUE;
                                GOTO LOG MCA REGISTER;
                        FI;
                        IF PCC Flag in IA32_MCi_STATUS = 1
                            THEN (* processor context might have been corrupted *)
                                RESTARTABILITY = FALSE;
                            ELSE (* It is a uncorrected recoverable (UCR) error *)
                                IF S Flag in IA32_MCi_STATUS = 0
                                    THEN
                                        IF AR Flag in IA32_MCi_STATUS = 0
                                            THEN (* It is a uncorrected no action required (UCNA) error *)
                                                GOTO CONTINUE; (* let CMCI and CMC polling handler to process *)
                                            ELSE
                                                FESTARTABILITY = FALSE; (* S=0, AR=1 is illegal *)
                                        FI
                                FI;
                                IF RESTARTABILITY = FALSE
                                    THEN (* no need to take recovery action if RESTARTABILITY is already false *)
                                        NOERROR = FALSE;
                                        GOTO LOG MCA REGISTER;
                                FI;
                                (* S in IA32_MCi_STATUS = 1 *)
                                IF AR Flag in IA32_MCi_STATUS = 1
                                    THEN (* It is a software recoverable and action required (SRAR) error *)
```

```
                                        IF OVER Flag in IA32_MCi_STATUS = 1
                                            THEN
                                                RESTARTABILITY = FALSE;
                                                NOERROR = FALSE;
                                                GOTO LOG MCA REGISTER;
                                        FI
                                        IF MCACOD Value in IA32_MCi_STATUS is recognized
                                           AND Current Processor is an Affected Processor
                                            THEN
                                                Implement MCACOD specific recovery action;
                                                CLEAR_MC_BANK = TURE;
                                            ELSE
                                                RESTARTABILITY = FALSE;
                                        FI;
                                    ELSE (* It is a software recoverable and action optional (SRAO) error *)
                                        IF OVER Flag in IA32_MCi_STATUS = 0 AND
                                         MCACOD in IA32_MCi_STATUS is recognized
                                            THEN
                                                Implement MCACOD specific recovery action;
                                        FI;
                                        CLEAR_MC_BANK = TRUE;
                                FI; AR
                            FI; PCC
                            NOERROR = FALSE;
                            GOTO LOG MCA REGISTER;
                        ELSE  (* It is a corrected error; continue to the next IA32_MCi_STATUS *)
                            GOTO CONTINUE;
                    FI; UC
            FI; VAL
LOG MCA REGISTER:
            SAVE IA32_MCi_STATUS;
            If MISCV in IA32_MCi_STATUS
                THEN
                    SAVE IA32_MCi_MISC;
            FI;
            IF ADDRV in IA32_MCi_STATUS
                THEN
                    SAVE IA32_MCi_ADDR;
            FI;
            IF CLEAR_MC_BANK = TRUE
                THEN
                    SET all 0 to IA32_MCi_STATUS;
                    If MISCV in IA32_MCi_STATUS
                        THEN
                            SET all 0 to IA32_MCi_MISC;
                    FI;
                    IF ADDRV in IA32_MCi_STATUS
                        THEN
                            SET all 0 to IA32_MCi_ADDR;
                    FI;
            FI;
            CONTINUE:
    OD;
( *END FOR *)
RETURN;
(* End of MCA ERROR PROCESSING*)
```

15.10.4.2 Corrected Machine-Check Handler for Error Recovery

When writing a corrected machine check handler, which is invoked as a result of CMCI or called from an OS CMC Polling dispatcher, consider the following:

- The VAL (valid) flag in each IA32_MCi_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank does not contain valid error information and does not need to be checked.
- The CMCI or CMC polling handler is responsible for logging and clearing corrected errors. The UC flag in each IA32_MCi_Status register indicates whether the reported error was corrected (UC=0) or not (UC=1).
- When IA32_MCG_CAP [24] is one, the CMC handler is also responsible for logging and clearing uncorrected no-action required (UCNA) errors. When the UC flag is one but the PCC, S, and AR flags are zero in the IA32_MCi_STATUS register, the reported error in this bank is an uncorrected no-action required (UCNA) error.
- In addition to corrected errors and UCNA errors, the CMC handler optionally logs uncorrected (UC=1 and PCC=1), software recoverable machine check errors (UC=1, PCC=0 and S=1), but should avoid clearing those errors from the MC banks. Clearing these errors may result in accidentally removing these errors before these errors are actually handled and processed by the MCE handler for attempted software error recovery.

Example Example 15-5. gives pseudocode for a CMCI handler with UCR support.

Example 15-5. Corrected Error Handler Pseudocode with UCR Support

```
Corrected Error HANDLER:  (* Called from CMCI handler or OS CMC Polling Dispatcher*)
IF CPU supports MCA
    THEN
        FOR each bank of machine-check registers
            DO
                READ IA32_MCi_STATUS;
                IF VAL flag in IA32_MCi_STATUS = 1
                    THEN
                        IF UC Flag in IA32_MCi_STATUS = 0 (* It is a corrected error *)
                            THEN
                                GOTO LOG CMC ERROR;
                            ELSE
                                IF Bit 24 in IA32_MCG_CAP = 0
                                    THEN
                                        GOTO CONTINUE;
                                FI;
                                IF S Flag in IA32_MCi_STATUS = 0 AND AR Flag in IA32_MCi_STATUS = 0
                                    THEN (* It is a uncorrected no action required error *)
                                        GOTO LOG CMC ERROR
                                FI
                                IF EN Flag in IA32_MCi_STATUS = 0
                                    THEN (* It is a spurious MCA error *)
                                        GOTO LOG MCM ERROR
                                FI;
                        FI;
                FI;
                GOTO CONTINUE;
            LOG CMC ERROR:
                SAVE IA32_MCi_STATUS;
                If MISCV Flag in IA32_MCi_STATUS
                    THEN
```

```
                    SAVE IA32_MCi_MISC;
                    SET all 0 to IA32_MCi_MISC;
                FI;
                IF ADDRV Flag in IA32_MCi_STATUS
                    THEN
                    SAVE IA32_MCi_ADDR;
                    SET all 0 to IA32_MCi_ADDR
                FI;
                SET all 0 to IA32_MCi_STATUS;
                CONTINUE:
            OD;
    ( *END FOR *)
FI;
```

11. Updates to Chapter 18, Volume 3A

Change bars show changes to Chapter 18 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:* System Programming Guide, Part 1.

...

18.21 CONTROL REGISTERS

...

- DE — Debugging extensions. Causes an undefined opcode (#UD) exception to be generated when debug registers DR4 and DR5 are references for improved performance (see Section 19.2.2, "Debug Registers DR4 and DR5").
- PSE — Page size extensions. Enables 4-MByte pages with 32-bit paging when set (see Section 4.3, "32-Bit Paging").
- MCE — Machine-check enable. Enables the machine-check exception, allowing exception handling for certain hardware error conditions (see Chapter 15, "Machine-Check Architecture").

...

18.22.1.1 Physical Memory Addressing Extension

The new PAE (physical address extension) flag in control register CR4, bit 5, may enable additional address lines on the processor, allowing extended physical addresses. This option can only be used when paging is enabled, using a new page-table mechanism provided to support the larger physical address range (see Section 4.1, "Paging Modes and Control Bits").

...

18.22.1.2 Global Pages

The new PGE (page global enable) flag in control register CR4, bit 7, provides a mechanism for preventing frequently used pages from being flushed from the translation lookaside buffer (TLB). When this flag is set, frequently used pages (such as pages

containing kernel procedures or common data tables) can be marked global by setting the global flag in a page-directory or page-table entry.

On a task switch or a write to control register CR3 (which normally causes the TLBs to be flushed), the entries in the TLB marked global are not flushed. Marking pages global in this manner prevents unnecessary reloading of the TLB due to TLB misses on frequently used pages. See Section 4.10, "Caching Translation Information" for a detailed description of this mechanism.

18.22.1.3 Larger Page Sizes

The P6 family processors support large page sizes. For 32-bit paging, this facility is enabled with the PSE (page size extension) flag in control register CR4, bit 4. When this flag is set, the processor supports either 4-KByte or 4-MByte page sizes. PAE paging and IA-32e paging support 2-MByte pages regardless of the value of CR4.PSE (see Section 4.4, "PAE Paging" and Section 4.5, "IA-32e Paging"). See Chapter 4, "Paging," for more information about large page sizes.

...

18.30.1 Large Pages

The Pentium processor extended the memory management/paging facilities of the IA-32 to allow large (4 MBytes) pages sizes (see Section 4.3, "32-Bit Paging"). The first P6 family processor (the Pentium Pro processor) added a 2 MByte page size to the IA-32 in conjunction with the physical address extension (PAE) feature (see Section 4.4, "PAE Paging").

The availability of large pages with 32-bit paging on any IA-32 processor can be determined via feature bit 3 (PSE) of register EDX after the CPUID instruction has been execution with an argument of 1. (Large pages are always available with PAE paging and IA-32e paging.) Intel processors that do not support the CPUID instruction support only 32-bit paging and do not support page size enhancements. (See "CPUID—CPU Identification" in Chapter 3, "Instruction Set Reference, A-M," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*, and AP-485, *Intel Processor Identification and the CPUID Instruction,* for more information on the CPUID instruction.)

...

12. Updates to Chapter 21, Volume 3B

Change bars show changes to Chapter 21 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

21.6.9 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 to 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 toC0001FFFH. 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 to 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 toC0001FFFH. 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 22.1.3 for details. If the bitmaps are used, their address must be 4-KByte aligned.

...

21.6.11 Extended-Page-Table Pointer (EPTP)

The **extended-page-table pointer** (EPTP) contains the address of the base of EPML4 table (see Chapter 24), as well as other EPT configuration information. The format of this field is shown in Table Table 21-8..

Table 21-8. Format of Extended-Page-Table Pointer

Bit Position(s)	Field
2:0	EPT paging-structure memory type (see Section 25.2.4): 0 = Uncacheable (UC) 6 = Write-back (WB) Other values are reserved.[1]
5:3	This value is 1 less than the EPT page-walk length (see Section 25.2.2)
11:6	Reserved
N–1:12	Bits N–1:12 of the physical address of the 4-KByte aligned EPT PML4 table[2]
63:N	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.

...

21.6.13 Controls for PAUSE-Loop Exiting

On processors that support the 1-setting of the "PAUSE-loop exiting" VM-execution control, the VM-execution control fields include the following 32-bit fields:

- **PLE_Gap**. Software can configure this field as an upper bound on the amount of time between two successive executions of PAUSE in a loop.
- **PLE_Window**. Software can configure this field as an upper bound on the amount of time a guest is allowed to execute in a PAUSE loop.

These fields measure time based on a counter that runs at the same rate as the time-stamp counter (TSC). See Section 22.1.3 for more details regarding PAUSE-loop exiting.

...

13. Updates to Chapter 22, Volume 3B

Change bars show changes to Chapter 22 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

22.1.3 Instructions That Cause VM Exits Conditionally

...

- **MWAIT.** The MWAIT instruction causes a VM exit if the "MWAIT exiting" VM-execution control is 1.
- **PAUSE.**The behavior of each of this instruction depends on CPL and the settings of the "PAUSE exiting" and "PAUSE-loop exiting" VM-execution controls**:**
  - CPL = 0.
    - If the "PAUSE exiting" and "PAUSE-loop exiting" VM-execution controls are both 0, the PAUSE instruction executes normally.
    - If the "PAUSE exiting" VM-execution control is 1, the PAUSE instruction causes a VM exit (the "PAUSE-loop exiting" VM-execution control is ignored if CPL = 0 and the "PAUSE exiting" VM-execution control is 1).
    - If the "PAUSE exiting" VM-execution control is 0 and the "PAUSE-loop exiting" VM-execution control is 1, the following treatment applies.

      The logical processor determines the amount of time between this execution of PAUSE and the previous execution of PAUSE at CPL 0. If this amount of time exceeds the value of the VM-execution control field PLE_Gap, the processor considers this execution to be the first execution of PAUSE in a loop. (It also does so for the first execution of PAUSE at CPL 0 after VM entry.)

      Otherwise, the logical processor determines the amount of time since the most recent execution of PAUSE that was considered to be the first in a loop. If this amount of time exceeds the value of the VM-execution control field PLE_Window, a VM exit occurs.

For purposes of these computations, time is measured based on a counter that runs at the same rate as the timestamp counter (TSC).

— CPL > 0.

- If the "PAUSE exiting" VM-execution control is 0, the PAUSE instruction executes normally.
- If the "PAUSE exiting" VM-execution control is 1, the PAUSE instruction causes a VM exit.

The "PAUSE-loop exiting" VM-execution control is ignored if CPL > 0.

...

14. Updates to Chapter 23, Volume 3B

Change bars show changes to Chapter 23 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

23.2.1.1 VM-Execution Control Fields

...

— If the VM entry fails, the any clearing of the bytes may or may not occur. This is true either if the failure causes control to pass to the instruction following the VM-entry instruction or if it cause processor state to be loaded from the host-state area of the VMCS. Behavior may be implementation-specific.

...

23.3.1.6 Checks on Guest Page-Directory-Pointer-Table Entries

If CR0.PG =1, CR4.PAE = 1, and IA32_EFER.LMA = 0, the logical processor also uses **PAE paging** (see Section 4.4 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.

...

23.3.2.2 Loading Guest Segment Registers and Descriptor-Table Registers

...

- SS, DS, ES, FS, and GS, and LDTR.
  — The selector fields are loaded.

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.

— 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.
  — SS.B: if the unusable bit for SS is 1, SS.B is set to 1.
  — The base addresses for FS and GS: always loaded. 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.

...

15. Updates to Chapter 24, Volume 3B

Change bars show changes to Chapter 24 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

Table24-7. 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]
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).
11:9	Reserved (cleared to 0).
12	NMI unblocking due to IRET
63:13	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 25.2.2).

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.

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 24.2.3).

Otherwise, bit 12 is defined as follows:

If the "virtual NMIs" VM-execution control is 0, the EPT violation was caused by a memory access as part of execution of the IRET instruction, and blocking by NMI (see Table 21-3) was in effect before execution of IRET, bit 12 is set to 1.

If the "virtual NMIs" VM-execution control is 1,the EPT violation was caused by a memory access as part of execution of 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.

...

24.5.2... Loading Host Segment and Descriptor-Table Registers

...

- 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. Set to 1.
  - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
  - TR. Set to 0.

...

24.5.4 Checking and Loading Host Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LMA = 0, the logical processor uses **PAE paging**. See Section 4.4 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

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.

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).

...

16. Updates to Chapter 29, Volume 3B

Change bars show changes to Chapter 29 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B:* System Programming Guide, Part 2.

...

29.4.2 Machine Check Considerations

The following sequence determine how machine check exceptions 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 23.7). The basic exit reason will be "VM-entry failure due to machine check."

...

29.4.3 MCA Error Handling Guidelines for VMM

Section 29.4.2 covers general requirements for VMMs to handle machine-check exceptions, when normal operation of the guest machine and/or the VMM is no longer possible. enhancements of machine check architecture in newer processors may support software recovery of uncorrected MC errors (UCR) signaled through either machine-check exceptions or corrected machine-check interrupt (CMCI). Section 15.5 and Section 15.6 describes details of these more recent enhancements of machine check architecture.

In general, Virtual Machine Monitor (VMM) error handling should follow the recommendations for OS error handling described in Section 15.3, Section 15.6, Section 15.9, and Section 15.10. This section describes additional guidelines for hosted and native hypervisor-based VMM implementations to support corrected MC errors and recoverable uncorrected MC errors.

Because a hosted VMM provides virtualization services in the context of an existing standard host OS, the host OS controls platform hardware through the host OS services such as the standard OS device drivers. In hosted VMMs. MCA errors will be handled by the host OS error handling software.

In native VMMs, the hypervisor runs on the hardware directly, and may provide only a limited set of platform services for guest VMs. Most platform services may instead be provided by a "control OS". In hypervisor-based VMMs, MCA errors will either be delivered directly to the VMM MCA handler (when the error is signaled while in the VMM context) or cause by a VM exit from a guest VM or be delivered to the MCA intercept handler. There are two general approaches the hypervisor can use to handle the MCA error: either within the hypervisor itself or by forwarding the error to the control OS.

29.4.3.1 VMM Error Handling Strategies

Broadly speaking, there are two strategies that VMMs may take for error handling:

- Basic error handling: in this approach the guest VM is treated as any other thread of execution. If the error recovery action does not support restarting the thread after handling the error, the guest VM should be terminated.
- MCA virtualization: in this approach, the VMM virtualizes the MCA events and hardware. This enables the VMM to intercept MCA events and inject an MCA into the guest VM. The guest VM then has the opportunity to attempt error recovery actions, rather than being terminated by the VMM.

Details of these approaches and implementation considerations for hosted and native VMMs are discussed below.

29.4.3.2 Basic VMM MCA error recovery handling

The simplest approach is for the VMM to treat the guest VM as any other thread of execution:

- MCE's that occur outside the stream of execution of a virtual machine guest will cause an MCE abort and may be handled by the MCA error handler following the recovery actions and guidelines described in Section 15.9, and Section 15.10. This includes logging the error and taking appropriate recovery actions when necessary. The VMM must not resume the interrupted thread of execution or another VM until it has taken the appropriate recovery action or, in the case of fatal MCAs, reset the system.
- MCE's that occur while executing in the context of a virtual machine will be intercepted by the VMM. The MCA intercept handler may follow the error handling guidelines listed in Section 15.9 and Section 15.10 for SRAO and SRAR errors. For SRAR errors, terminating the thread of execution will involve terminating the affected guest VM. For fatal errors the MCA handler should log the error and reset the system -- the VMM should not resume execution of the interrupted VM.

29.4.3.3 Implementation Considerations for the Basic Model

For hosted VMMs, the host OS MCA error handling code will perform error analysis and initiate the appropriate recovery actions. For the basic model this flow does not change when terminating a guest VM although the specific actions needed to terminate a guest VM may be different than terminating an application or user process.

For native, hypervisor-based VMMs, MCA errors will either be delivered directly to the VMM MCA handler (when the error is signaled while in the VMM context) or cause a VM exit from a guest VM or be delivered to the MCA intercept handler. There are two general approaches the hypervisor can use to handle the MCA error: either by forwarding the error to the control OS or within the hypervisor itself. These approaches are described in the following paragraphs.

The hypervisor may forward the error to the control OS for handling errors. This approach simplifies the hypervisor error handling since it relies on the control OS to implement the basic error handling model. The control OS error handling code will be similar to the error handling code in the hosted VMM. Errors can be forwarded to the control OS via an OS callback or by injecting an MCE event into the control OS. Injecting an MCE will cause the control OS MCA error handler to be invoked. The control OS is responsible for terminating the affected guest VM, if necessary, which may require cooperation from the hypervisor.

Alternatively, the error may be handled completely in the hypervisor. The hypervisor error handler is enhanced to implement the basic error handling model and the hypervisor error handler has the capability to fully analyze the error information and take recovery actions based on the guidelines. In this case error handling steps in the hypervisor are similar to those for the hosted VMM described above (where the hypervisor replaces the host OS actions). The hypervisor is responsible for terminating the affected guest VM, if necessary.

In all cases, if a fatal error is detected the VMM error handler should log the error and reset the system. The VMM error handler must ensure that guest VMs are not resumed after a fatal error is detected to ensure error containment is maintained.

29.4.3.4 MCA Virtualization

A more sophisticated approach for handling errors is to virtualize the MCA. This involves virtualizing the MCA hardware and intercepting the MCA event in the VMM when a guest VM is interrupted by an MCA. After analyzing the error, the VMM error handler may then decide to inject an MCE abort into the guest VM for attempted guest VM error recovery. This would enable the guest OS the opportunity to take recovery actions specific to that guest.

For MCA virtualization, the VMM must provide the guest physical address for memory errors instead of the system physical address when reporting the errors to the guest VM. To compute the guest physical address, the VMM needs to maintain a reverse mapping of system physical page addresses to guest physical page addresses.

When the MCE is injected into the guest VM, the guest OS MCA handler would be invoked. The guest OS implements the MCA handling guidelines and it could potentially terminate the interrupted thread of execution within the guest instead of terminating the VM. The guest OS may also disable use of the affected page by the guest. When disabling the page the VMM error handler may handle the case where a page is shared by the VMM and a guest or by two guests. In these cases the page use must be disabled in both contexts to ensure no subsequent consumption errors are generated.

29.4.3.5 Implementation Considerations for the MCA Virtualization Model

MCA virtualization may be done in either hosted VMMs or hypervisor-based VMMs. The error handling flow is similar to the flow described in the basic handling case. The major difference is that the recovery action includes injecting the MCE abort into the guest VM to enable recovery by the guest OS when the MCA interrupts the execution of a guest VM.