

IA-32架构（始于Intel386处理器系列）为操作系统和系统开发软件提供了广泛的支持。这种支持提供了多种操作模式，包括：

- 实模式、保护模式、虚拟8086模式和系统管理模式。这些有时被称为传统模式。

Intel 64架构支持IA-32架构中几乎所有的系统编程设施，并将其扩展至支持64位编程环境的新操作模式（IA-32e模式）。IA-32e模式允许软件在以下两种子模式之一中运行：

- 64位模式支持64位操作系统和64位应用程序
- 兼容模式允许大多数传统软件运行；它与64位应用程序共存于64位操作系统之下

IA-32系统级架构包含协助以下操作的功能：

- 内存管理
- 软件模块保护机制
- 多任务处理
- 异常与中断处理
- 多处理
- 缓存管理
- 硬件资源与电源管理
- 调试与性能监控

本章详细描述了该架构的各个组成部分，同时介绍了用于在系统级设置和控制处理器的系统寄存器，并简要概述了处理器的系统级（操作系统）指令。

系统级架构的许多特性仅由系统程序员使用。然而，应用程序员可能需要阅读本章及后续章节，以便为应用程序创建可靠安全的环境。

本概述及本书后续大部分章节主要关注IA-32架构的保护模式操作。同时也会描述Intel 64架构的IA-32e模式操作，因其与保护模式操作存在差异。

所有Intel 64和IA-32处理器在上电或重置后都会进入实地址模式（参见第9章“处理器管理与初始化”）。随后软件会启动从实地址模式到保护模式的切换。若需启用IA-32e模式操作，软件还需启动从保护模式到IA-32e模式的切换。

## 2.1 系统级架构概述

系统级架构包含一组寄存器、数据结构和指令，旨在支持基本的系统级操作，如内存管理、中断与异常处理、任务管理以及多处理器控制。

图2-1汇总了适用于32位模式的系统寄存器与数据结构。适用于IA-32e模式的系统寄存器与数据结构如图2-2所示。

IA-32 architecture (beginning with the Intel386 processor family) provides extensive support for operating-system and system-development software. This support offers multiple modes of operation, which include:

- Real mode, protected mode, virtual 8086 mode, and system management mode. These are sometimes referred to as legacy modes.

Intel 64 architecture supports almost all the system programming facilities available in IA-32 architecture and extends them to a new operating mode (IA-32e mode) that supports a 64-bit programming environment. IA-32e mode allows software to operate in one of two sub-modes:

- 64-bit mode supports 64-bit OS and 64-bit applications
- Compatibility mode allows most legacy software to run; it co-exists with 64-bit applications under a 64-bit OS.

The IA-32 system-level architecture and includes features to assist in the following operations:

- Memory management
- Protection of software modules
- Multitasking
- Exception and interrupt handling
- Multiprocessing
- Cache management
- Hardware resource and power management
- Debugging and performance monitoring

This chapter provides a description of each part of this architecture. It also describes the system registers that are used to set up and control the processor at the system level and gives a brief overview of the processor's system-level (operating system) instructions.

Many features of the system-level architectural are used only by system programmers. However, application programmers may need to read this chapter and the following chapters in order to create a reliable and secure environment for application programs.

This overview and most subsequent chapters of this book focus on protected-mode operation of the IA-32 architecture. IA-32e mode operation of the Intel 64 architecture, as it differs from protected mode operation, is also described.

All Intel 64 and IA-32 processors enter real-address mode following a power-up or reset (see Chapter 9, “Processor Management and Initialization”). Software then initiates the switch from real-address mode to protected mode. If IA-32e mode operation is desired, software also initiates a switch from protected mode to IA-32e mode.

## 2.1 OVERVIEW OF THE SYSTEM-LEVEL ARCHITECTURE

System-level architecture consists of a set of registers, data structures, and instructions designed to support basic system-level operations such as memory management, interrupt and exception handling, task management, and control of multiple processors.

Figure 2-1 provides a summary of system registers and data structures that applies to 32-bit modes. System registers and data structures that apply to IA-32e mode are shown in Figure 2-2.

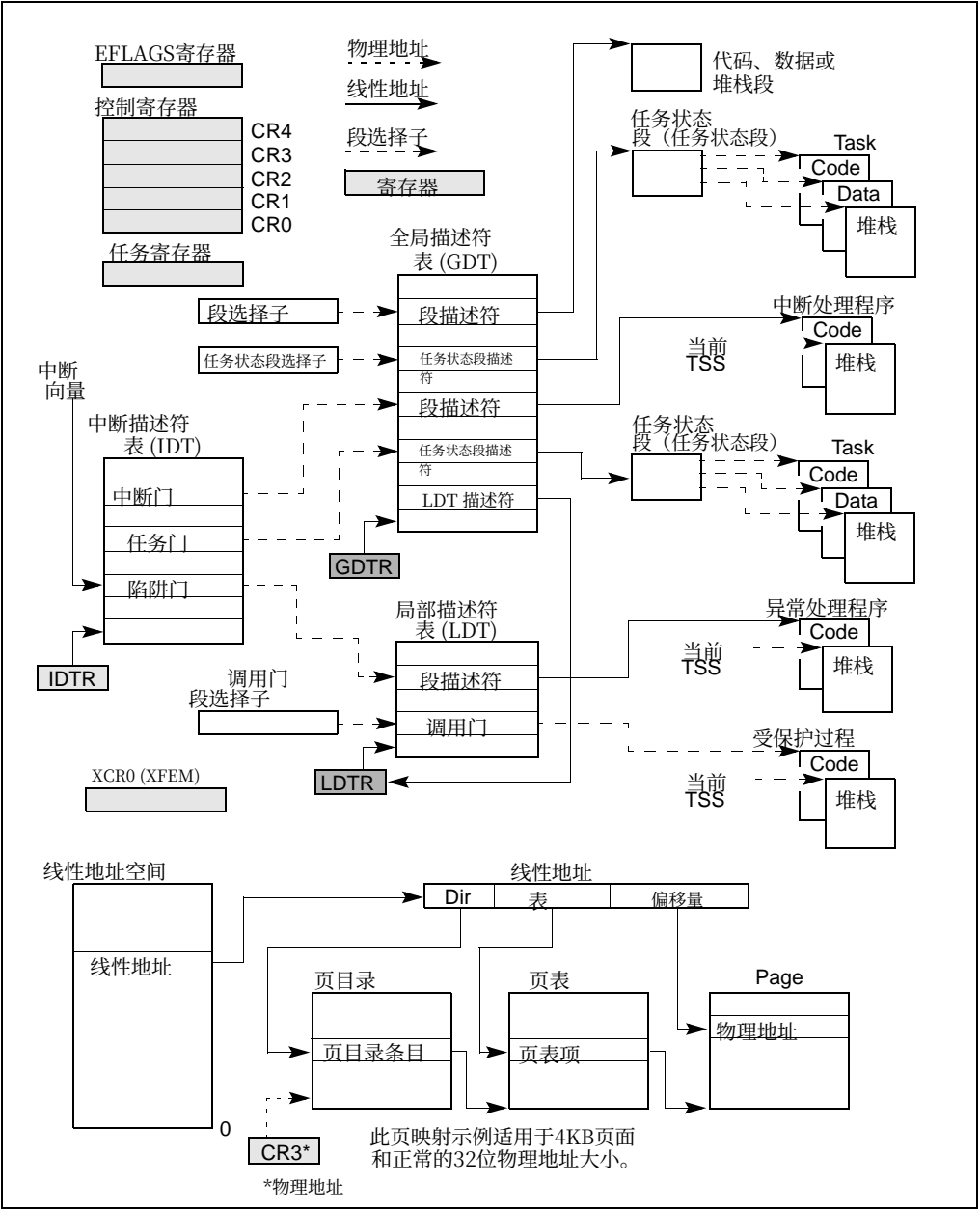


图2-1. IA-32 系统级寄存器和数据结构

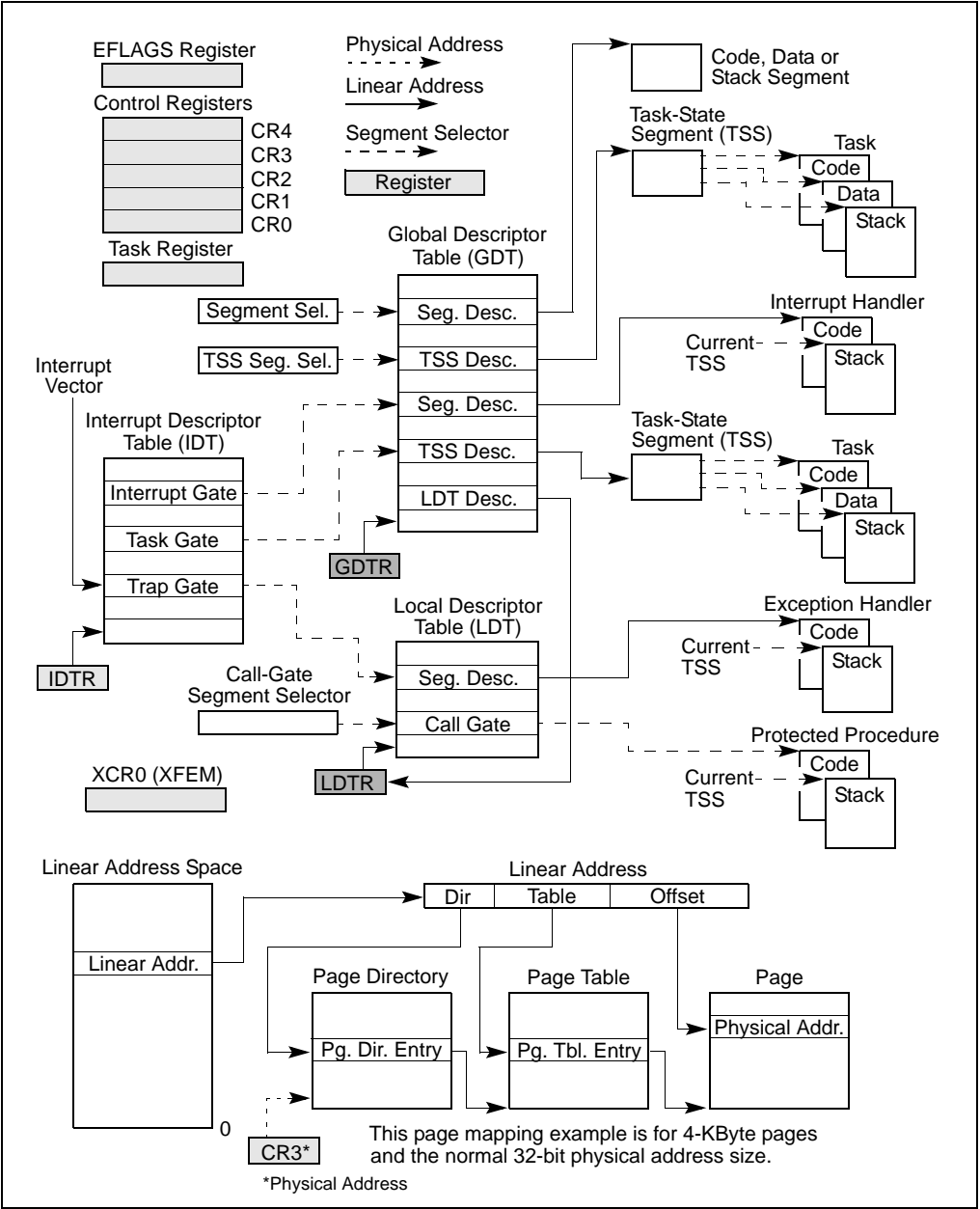


Figure 2-1. IA-32 System-Level Registers and Data Structures

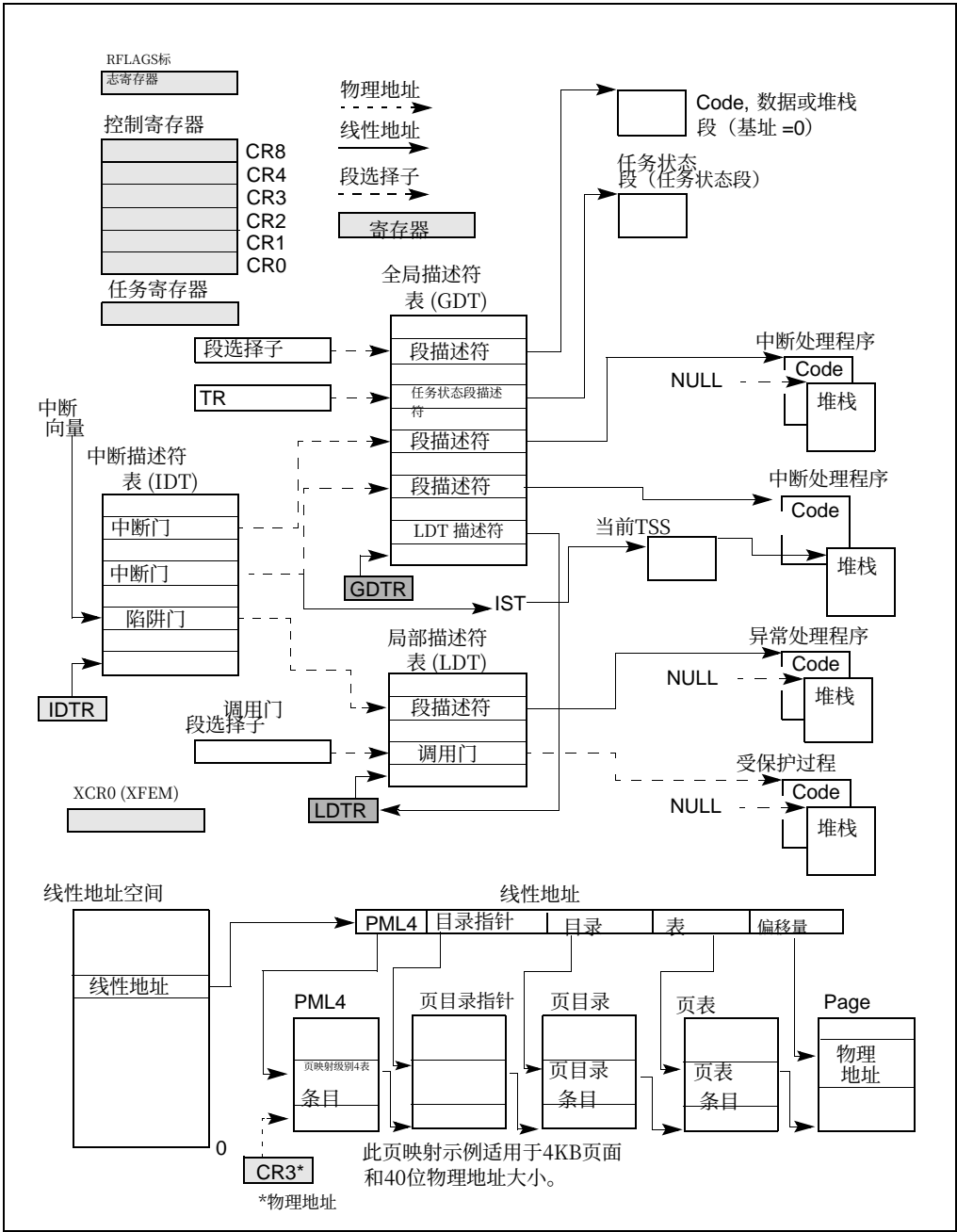


图2-2. IA-32e模式中的系统级寄存器和数据结构

2.1.1 全局和局部描述符表

在保护模式操作时，所有内存访问都会通过全局描述符表 (GDT) 或可选的局部描述符表 (LDT) 进行，如图2-1所示。这些表包含称为段描述符的条目。段描述符提供段的基地址以及访问权限、类型和使用信息。

每个段描述符都有一个关联的段选择子。段选择子为使用它的软件提供以下信息：指向GDT或LDT的索引（其关联段描述符的偏移量）、全局/局部标志（决定该选择子指向GDT还是LDT）以及访问权限信息。

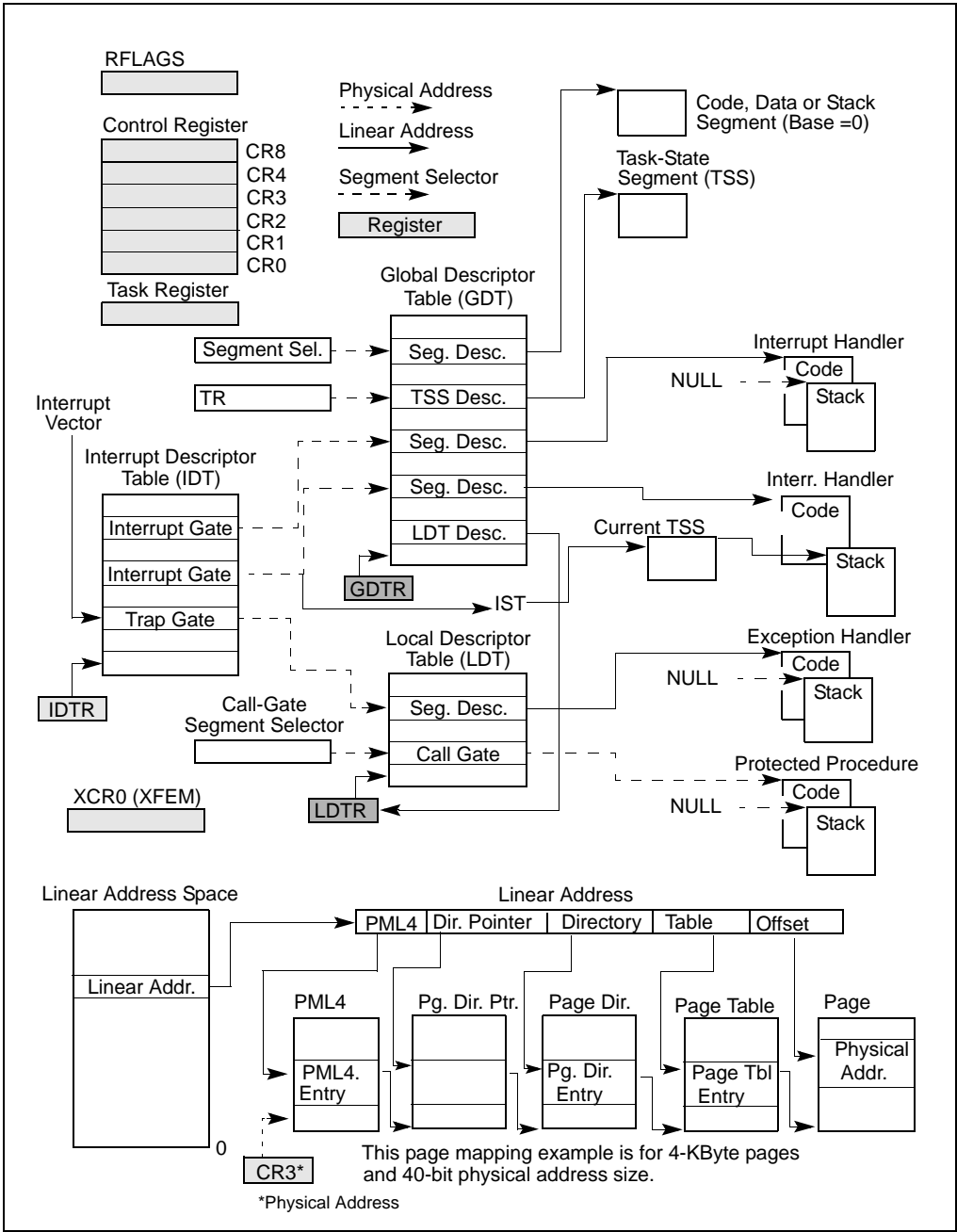


Figure 2-2. System-Level Registers and Data Structures in IA-32e Mode

2.1.1 Global and Local Descriptor Tables

When operating in protected mode, all memory accesses pass through either the global descriptor table (GDT) or an optional local descriptor table (LDT) as shown in Figure 2-1. These tables contain entries called segment descriptors. Segment descriptors provide the base address of segments well as access rights, type, and usage information.

Each segment descriptor has an associated segment selector. A segment selector provides the software that uses it with an index into the GDT or LDT (the offset of its associated segment descriptor), a global/local flag (determines whether the selector points to the GDT or the LDT), and access rights information.

要访问段中的某个字节，必须提供段选择子和偏移量。段选择子用于访问该段在GDT或LDT中的段描述符。处理器从段描述符中获取该段在线性地址空间中的基地址。随后偏移量提供该字节相对于基地址的位置。只要该段可从处理器当前运行的特权级（CPL）访问，此机制就可用于访问任何有效的代码、数据或堆栈段。CPL被定义为当前正在执行的代码段的保护级别。

参见图2-1。图中的实线箭头表示线性地址，虚线表示段选择子，点状箭头表示物理地址。为简化图示，许多段选择子被显示为直接指向段的指针。然而，从段选择子到其关联段的实际路径始终通过GDT或LDT。

GDT基址的线性地址包含在GDT寄存器（GDTR）中；LDT的线性地址包含在LDT寄存器（LDTR）中。

#### 2.1.1.1 IA-32e模式中的全局和局部描述符表

在IA-32e两种子模式（64位模式和兼容模式）下，GDTR和LDTR寄存器均扩展至64位宽度。更多信息请参阅第3.5.2节“IA-32e模式中的段描述符表”。

全局和局部描述符表在64位模式下得到扩展，以支持64位基地址（16字节的LDT描述符包含64位基地址和各种属性）。在兼容模式下，描述符不会扩展。

#### 2.1.2 系统段、段描述符和门

除了构成程序或过程执行环境的代码、数据和堆栈段之外，该架构还定义了两个系统段：任务状态段（TSS）和LDT。GDT不被视为一个段，因为它不是通过段选择子和段描述符来访问的。TSS和LDT有为它们定义的段描述符。

该架构还定义了一组特殊的描述符，称为门（调用门、中断门、陷阱门和任务门）。这些门提供了通往系统过程和处理程序的受保护网关，这些系统过程和处理程序可能运行在与应用程序和大多数过程不同的特权级上。例如，通过调用门进行CALL可以访问位于与当前代码段相同或数值上更低特权级（更高特权）的代码段中的过程。要通过调用门访问一个过程，调用过程1需提供调用门的选择子。然后处理器对调用门执行访问权限检查，比较CPL与调用门的特权级以及调用门所指向的目标代码段的特权级。

如果允许访问目标代码段，处理器将从调用门获取目标代码段的段选择子以及该代码段内的偏移量。如果调用需要改变特权级，处理器还会切换到目标特权级的堆栈。新堆栈的段选择子从当前运行任务的TSS中获取。门还促进了16位和32位代码段之间的转换，反之亦然。

##### 2.1.2.1 IA-32e模式中的门

在IA-32e模式下，以下描述符为16字节描述符（扩展后支持64位基址）：LDT描述符、64位任务状态段、调用门、中断门和陷阱门。

调用门促进64位模式与兼容模式之间的转换。IA-32e模式不支持任务门。在特权级变更时，堆栈段选择子不会从TSS中读取，而是被设置为NULL。

在本文档中，“过程”一词通常用作逻辑单元或代码块（如程序、过程、函数或例程）的通用术语。

To access a byte in a segment, a segment selector and an offset must be supplied. The segment selector provides access to the segment descriptor for the segment (in the GDT or LDT). From the segment descriptor, the processor obtains the base address of the segment in the linear address space. The offset then provides the location of the byte relative to the base address. This mechanism can be used to access any valid code, data, or stack segment, provided the segment is accessible from the current privilege level (CPL) at which the processor is operating. The CPL is defined as the protection level of the currently executing code segment.

See Figure 2-1. The solid arrows in the figure indicate a linear address, dashed lines indicate a segment selector, and the dotted arrows indicate a physical address. For simplicity, many of the segment selectors are shown as direct pointers to a segment. However, the actual path from a segment selector to its associated segment is always through a GDT or LDT.

The linear address of the base of the GDT is contained in the GDT register (GDTR); the linear address of the LDT is contained in the LDT register (LDTR).

#### 2.1.1.1 Global and Local Descriptor Tables in IA-32e Mode

GDTR and LDTR registers are expanded to 64-bits wide in both IA-32e sub-modes (64-bit mode and compatibility mode). For more information: see Section 3.5.2, “Segment Descriptor Tables in IA-32e Mode.”

Global and local descriptor tables are expanded in 64-bit mode to support 64-bit base addresses, (16-byte LDT descriptors hold a 64-bit base address and various attributes). In compatibility mode, descriptors are not expanded.

#### 2.1.2 System Segments, Segment Descriptors, and Gates

Besides code, data, and stack segments that make up the execution environment of a program or procedure, the architecture defines two system segments: the task-state segment (TSS) and the LDT. The GDT is not considered a segment because it is not accessed by means of a segment selector and segment descriptor. TSSs and LDTs have segment descriptors defined for them.

The architecture also defines a set of special descriptors called gates (call gates, interrupt gates, trap gates, and task gates). These provide protected gateways to system procedures and handlers that may operate at a different privilege level than application programs and most procedures. For example, a CALL to a call gate can provide access to a procedure in a code segment that is at the same or a numerically lower privilege level (more privileged) than the current code segment. To access a procedure through a call gate, the calling procedure<sup>1</sup> supplies the selector for the call gate. The processor then performs an access rights check on the call gate, comparing the CPL with the privilege level of the call gate and the destination code segment pointed to by the call gate.

If access to the destination code segment is allowed, the processor gets the segment selector for the destination code segment and an offset into that code segment from the call gate. If the call requires a change in privilege level, the processor also switches to the stack for the targeted privilege level. The segment selector for the new stack is obtained from the TSS for the currently running task. Gates also facilitate transitions between 16-bit and 32-bit code segments, and vice versa.

##### 2.1.2.1 Gates in IA-32e Mode

In IA-32e mode, the following descriptors are 16-byte descriptors (expanded to allow a 64-bit base): LDT descriptors, 64-bit TSSs, call gates, interrupt gates, and trap gates.

Call gates facilitate transitions between 64-bit mode and compatibility mode. Task gates are not supported in IA-32e mode. On privilege level changes, stack segment selectors are not read from the TSS. Instead, they are set to NULL.

1. The word “procedure” is commonly used in this document as a general term for a logical unit or block of code (such as a program, procedure, function, or routine).



### 2.1.3 任务状态段和任务门

TSS（见图2-1）定义了任务的执行环境状态。它包含通用寄存器、段寄存器、EFLAGS寄存器、EIP寄存器的状态，以及三个栈段（每个特权级对应一个栈）的段选择子与栈指针。TSS还包含与该任务关联的LDT段选择子以及页结构层次结构的基地址。

保护模式下的所有程序执行都在一个任务（称为当前任务）的上下文中进行。当前任务的TSS段选择子存储在任务寄存器中。切换到任务的最简单方法是调用或跳转到新任务。此处，新任务的TSS段选择子在CALL或JMP指令中给出。在切换任务时，处理器执行以下操作：

1. 将当前任务的状态存储在当前TSS中。2. 将新任务的段选择子加载到任务寄存器中。3. 通过GDT中的段描述符访问新TSS。4. 将新任务的状态从新TSS加载到通用寄存器、段寄存器、LDTR、控制寄存器CR3（页结构层次的基地址）、EFLAGS寄存器和EIP寄存器中。5. 开始执行新任务。

任务也可以通过选择任务门进行访问。任务门类似于调用门，不同之处在于它提供对TSS而非代码段的访问权限。

#### 2.1.3.1 IA-32e模式中的任务状态段

IA-32e模式不支持硬件任务切换。然而，任务状态段仍然存在。任务状态段的基地址由其描述符指定。

- 64位任务状态段保存着对64位操作至关重要的以下信息：
- 每个特权级的栈指针地址
  - 中断栈表的指针地址
  - IO权限位图的偏移地址（从TSS基地址起算）

任务寄存器在IA-32e模式下扩展为可容纳64位基地址。另请参阅：第7.7节“64位模式下的任务管理”。

### 2.1.4 中断与异常处理

外部中断、软件中断和异常通过中断描述符表 (IDT) 进行处理。IDT 存储了一系列门描述符，这些描述符提供了对中断与异常处理程序的访问。与 GDT 类似，IDT 不是一个段。IDT 基址的线性地址包含在 IDT 寄存器 (IDTR) 中。

IDT 中的门描述符可以是中断门、陷阱门或任务门描述符。要访问中断或异常处理程序，处理器首先从内部硬件、外部中断控制器或通过 INT、INTO、INT 3 或 BOUND 指令从软件接收一个中断向量（中断号）。该中断向量提供了进入 IDT 的索引。如果选定的门描述符是中断门或陷阱门，则关联的处理程序将以类似于通过调用门调用过程的方式被访问。如果描述符是任务门，则通过任务切换来访问处理程序。

#### 2.1.4.1 IA-32e 模式下的中断与异常处理

在IA-32e模式下，中断描述符被扩展为16字节以支持64位基地址。这对于64位模式和兼容模式均适用。

该IDTR寄存器已扩展为支持64位基地址。任务门不受支持。

### 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.

#### 2.1.3.1 Task-State Segments in IA-32e Mode

Hardware task switches are not supported in IA-32e mode. However, TSSs continue to exist. The base address of a TSS is specified by its descriptor.

A 64-bit TSS holds the following information that is important to 64-bit operation:

- Stack pointer addresses for each privilege level
- Pointer addresses for the interrupt stack table
- Offset address of the IO-permission bitmap (from the TSS base)

The task register is expanded to hold 64-bit base addresses in IA-32e mode. See also: Section 7.7, “Task Management in 64-bit Mode.”

### 2.1.4 Interrupt and Exception Handling

External interrupts, software interrupts and exceptions are handled through the interrupt descriptor table (IDT). The IDT stores a collection of gate descriptors that provide access to interrupt and exception handlers. Like the GDT, the IDT is not a segment. The linear address for the base of the IDT is contained in the IDT register (IDTR).

Gate descriptors in the IDT can be interrupt, trap, or task gate descriptors. To access an interrupt or exception handler, the processor first receives an interrupt vector (interrupt number) from internal hardware, an external interrupt controller, or from software by means of an INT, INTO, INT 3, or BOUND instruction. The interrupt vector provides an index into the IDT. If the selected gate descriptor is an interrupt gate or a trap gate, the associated handler procedure is accessed in a manner similar to calling a procedure through a call gate. If the descriptor is a task gate, the handler is accessed through a task switch.

#### 2.1.4.1 Interrupt and Exception Handling IA-32e Mode

In IA-32e mode, interrupt descriptors are expanded to 16 bytes to support 64-bit base addresses. This is true for 64-bit mode and compatibility mode.

The IDTR register is expanded to hold a 64-bit base address. Task gates are not supported.

### 2.1.5 内存管理

系统架构支持直接物理寻址内存或虚拟内存（通过分页）。使用物理寻址时，线性地址被视为物理地址。使用分页时：所有代码、数据、堆栈和系统段（包括GDT和IDT）均可被分页，仅最近访问的页会保留在物理内存中。

物理内存中页（有时称为页帧）的位置信息包含在分页结构中。这些结构驻留在物理内存中（关于32位分页的情况请参见图2-1）。

页结构层次的基础物理地址包含在控制寄存器CR3中。分页结构中的条目决定了页帧基址的物理地址、访问权限和内存管理信息。

要使用这种分页机制，需要将线性地址拆分为若干部分。这些部分分别提供分页结构和页帧中的独立偏移量。一个系统可以具有单一的分页结构层次或多个层次。例如，每个任务都可以拥有自己的层次结构。

#### 2.1.5.1 IA-32e模式中的内存管理

在IA-32e模式下，物理内存页由一组系统数据结构管理。在兼容模式和64位模式下，使用四级系统数据结构。这些包括：

- 页映射级别4 (PML4) — PML4表中的条目包含项目录指针表基址的物理地址、访问权限和内存管理信息。PML4的基址物理地址存储在CR3中。
- 一组项目录指针表 — 项目录指针表中的条目包含项目录表基址的物理地址、访问权限和内存管理信息。
- 多组项目录 — 项目录表中的条目包含页表基址的物理地址、访问权限和内存管理信息。
- 多组页表 — 页表中的条目包含页帧的物理地址、访问权限和内存管理信息。

### 2.1.6 系统寄存器

为协助处理器初始化和控制系统操作，系统架构在EFLAGS寄存器中提供了系统标志及若干系统寄存器：

- EFLAGS寄存器中的系统标志和IOPL字段控制任务与模式切换、中断处理、指令跟踪以及访问权限。另见：第2.3节“EFLAGS寄存器中的系统标志和字段”。
- 控制寄存器（CR0、CR2、CR3和CR4）包含用于控制系统级操作的各种标志和数据字段。这些寄存器中的其他标志用于指示操作系统或执行体内对特定处理器能力的支持。另见：第2.5节“控制寄存器”。
- 调试寄存器（未在图2-1中显示）允许设置断点，用于调试程序和系统软件。另见：第17章“调试、分支分析、TSC和服务质量”。
- GDTR、LDTR和IDTR寄存器包含各自表的线性地址和大小（界限）。另请参阅：第2.4节“内存管理寄存器”。

- 任务寄存器包含当前任务的TSS的线性地址和大小。另请参阅：第2.4节“内存管理寄存器”。

- 模型特定寄存器（未在图2-1中显示）。

模型特定寄存器（MSR）是一组主要供操作系统或执行过程（即运行在特权级别0的代码）使用的寄存器。这些寄存器控制诸如调试扩展、性能监控计数器、机器检查架构和内存类型范围（MTRR）等项目。

这些寄存器的数量和功能在Intel 64和IA-32处理器系列的不同成员中有所差异。另请参阅：第9.4节“模型特定寄存器（MSR）”和第35章“模型特定寄存器（MSR）”。

### 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 17, “Debug, Branch Profile, TSC, and Quality of Service.”
- 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 Chapter 35, “Model-Specific Registers (MSRs).”

大多数系统限制应用程序对系统寄存器（除EFLAGS寄存器外）的访问。然而，可以设计所有程序和过程都在最高特权级（特权级别0）运行的系统。在这种情况下，应用程序将被允许修改系统寄存器。

### 2.1.6.1 IA-32e模式中的系统寄存器

在IA-32e模式中，四个系统描述符表寄存器（GDTR、IDTR、LDTR和TR）在硬件层面被扩展以容纳64位基地址。EFLAGS变为64位RFLAGS寄存器。CR0–CR4扩展至64位。CR8变为可用。CR8提供对任务优先级寄存器（TPR）的读写访问，以便操作系统可以控制外部中断的优先级类别。

在64位模式中，调试寄存器DR0–DR7为64位。在兼容模式中，DR0–DR3中的地址匹配同样以64位粒度完成。

在支持IA-32e模式的系统中，扩展功能启用寄存器（IA32\_EFER）可用。此模型特定寄存器控制IA-32e模式的激活及其他IA-32e模式操作。此外，还有若干模型特定寄存器用于管理IA-32e模式指令：

- IA32\_KernelGSbase — 由SWAPGS指令使用。
- IA32\_LSTAR — 由SYSCALL指令使用。
- IA32\_SYSCALL\_FLAG\_MASK — 由SYSCALL指令使用。
- IA32\_STAR\_CS — 由SYSCALL和SYSRET指令使用。

### 2.1.7 其他系统资源

除前几节描述的系统寄存器和数据结构外，系统架构还提供以下附加资源：

- 操作系统指令（另见：第2.7节“系统指令摘要”）。
- 性能监控计数器（未在图2-1中显示）。
- 内部缓存和缓冲区（未在图2-1中显示）。

性能监控计数器是可编程的事件计数器，用于统计处理器事件，例如已解码指令的数量、接收的中断数量或缓存加载次数。另请参阅：第19章“性能监控事件”。

该处理器提供了多个内部缓存和缓冲区。缓存用于存储数据和指令。缓冲区用于存储诸如系统和应用段的已解码地址、等待执行的写操作等内容。另请参阅：第11章“内存缓存控制”。

## 2.2 操作模式

IA-32支持三种操作模式和一种准操作模式：

- 保护模式——这是处理器的原生操作模式。它提供了丰富的架构特性、灵活性、高性能以及对现有软件基础的向后兼容性。

- 实地址模式——此操作模式提供了Intel 8086处理器的编程环境，并带有若干扩展功能（例如能够切换到保护模式或系统管理模式）。
- 系统管理模式（SMM）—— SMM是所有IA-32处理器的标准架构特性，始于Intel386 SL处理器。该模式为操作系统或执行程序提供了实现电源管理和OEM差异化功能的透明机制。SMM通过激活外部系统中断引脚（SMI#）进入，从而产生系统管理中断

（SMI）。在SMM中，处理器切换到独立的地址空间，同时保存当前运行程序或任务的上下文。随后可透明地执行SMM专用代码。从SMM返回后，处理器将恢复到SMI发生前的状态。

- 虚拟8086模式——在保护模式下，处理器支持一种称为虚拟8086的准操作模式- 8086模式。该模式允许处理器在受保护的多任务处理环境中执行8086软件。

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.

### 2.1.6.1 System Registers in IA-32e Mode

In IA-32e mode, the four system-descriptor-table registers (GDTR, IDTR, LDTR, and TR) are expanded in hardware to hold 64-bit base addresses. EFLAGS becomes the 64-bit RFLAGS register. CR0–CR4 are expanded to 64 bits. CR8 becomes available. CR8 provides read-write access to the task priority register (TPR) so that the operating system can control the priority classes of external interrupts.

In 64-bit mode, debug registers DR0–DR7 are 64 bits. In compatibility mode, address-matching in DR0–DR3 is also done at 64-bit granularity.

On systems that support IA-32e mode, the extended feature enable register (IA32\_EFER) is available. This model-specific register controls activation of IA-32e mode and other IA-32e mode operations. In addition, there are several model-specific registers that govern IA-32e mode instructions:

- IA32\_KernelGSbase** — Used by SWAPGS instruction.
- IA32\_LSTAR** — Used by SYSCALL instruction.
- IA32\_SYSCALL\_FLAG\_MASK** — Used by SYSCALL instruction.
- IA32\_STAR\_CS** — Used by SYSCALL and SYSRET instruction.

### 2.1.7 Other System Resources

Besides the system registers and data structures described in the previous sections, system architecture provides the following additional resources:

- Operating system instructions (see also: Section 2.7, “System Instruction Summary”).
- Performance-monitoring counters (not shown in Figure 2-1).
- Internal caches and buffers (not shown in Figure 2-1).

Performance-monitoring counters are event counters that can be programmed to count processor events such as the number of instructions decoded, the number of interrupts received, or the number of cache loads. See also: Chapter 19, “Performance Monitoring Events.”

The processor provides several internal caches and buffers. The caches are used to store both data and instructions. The buffers are used to store things like decoded addresses to system and application segments and write operations waiting to be performed. See also: Chapter 11, “Memory Cache Control.”

## 2.2 MODES OF OPERATION

The IA-32 supports three operating modes and one quasi-operating mode:

- Protected mode** — This is the native operating mode of the processor. It provides a rich set of architectural features, flexibility, high performance and backward compatibility to existing software base.
- Real-address mode** — This operating mode provides the programming environment of the Intel 8086 processor, with a few extensions (such as the ability to switch to protected or system management mode).
- System management mode (SMM)** — SMM is a standard architectural feature in all IA-32 processors, beginning with the Intel386 SL processor. This mode provides an operating system or executive with a transparent mechanism for implementing power management and OEM differentiation features. SMM is entered through activation of an external system interrupt pin (SMI#), which generates a system management interrupt (SMI). In SMM, the processor switches to a separate address space while saving the context of the currently running program or task. SMM-specific code may then be executed transparently. Upon returning from SMM, the processor is placed back into its state prior to the SMI.
- Virtual-8086 mode** — In protected mode, the processor supports a quasi-operating mode known as virtual-8086 mode. This mode allows the processor execute 8086 software in a protected, multitasking environment.



Intel 6 4架构支持IA-32架构的所有操作模式和IA-32e模式；

- IA-32e模式——在IA-32e模式下，处理器支持两种子模式：兼容模式和64位模式。64位模式提供64位线性寻址并支持大于64GB的物理地址空间。兼容模式允许大多数传统的保护模式应用程序无需修改即可运行。

图2-3展示了处理器如何在操作模式之间切换。

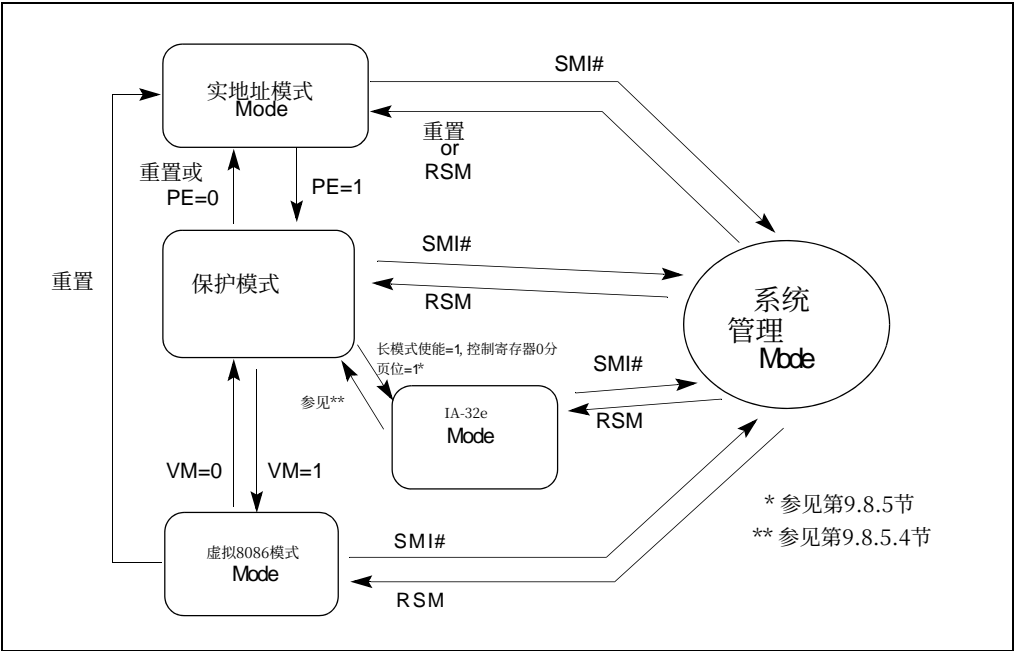


图2-3. 处理器操作模式之间的转换

处理器在上电或重置后进入实地址模式。控制寄存器CR0中的保护使能标志（PE）随后控制处理器是运行在实地址模式还是保护模式下。另见：第9.9节“模式切换”和第4.1.2节“分页模式启用”。

EFLAGS寄存器中的VM标志决定处理器是运行在保护模式还是虚拟8086模式。保护模式与虚拟8086模式之间的转换通常作为任务切换或从中断或异常处理程序返回的一部分来执行。另请参阅：第20.2.5节“进入虚拟8086模式”。

LMA位（IA32\_EFER.LMA[位10]）决定处理器是否运行在IA-32e模式。当在IA-32e模式下运行时，64位或兼容子模式的操作由代码段的CS.L位决定。处理器通过启用分页并设置LME位（IA32\_EFER.LME[位8]）从保护模式进入IA-32e模式。另请参阅：第9章“处理器管理与初始化”。

当处理器在实地址模式、保护模式、虚拟8086模式或IA-32e模式下接收到SMI时，处理器会切换到系统管理模式。执行RSM指令后，处理器始终会返回到发生SMI时所在的模式。

Intel 64 architecture supports all operating modes of IA-32 architecture and IA-32e modes:

- IA-32e mode** — In IA-32e mode, the processor supports two sub-modes: compatibility mode and 64-bit mode. 64-bit mode provides 64-bit linear addressing and support for physical address space larger than 64 GBytes. Compatibility mode allows most legacy protected-mode applications to run unchanged.

Figure 2-3 shows how the processor moves between operating modes.

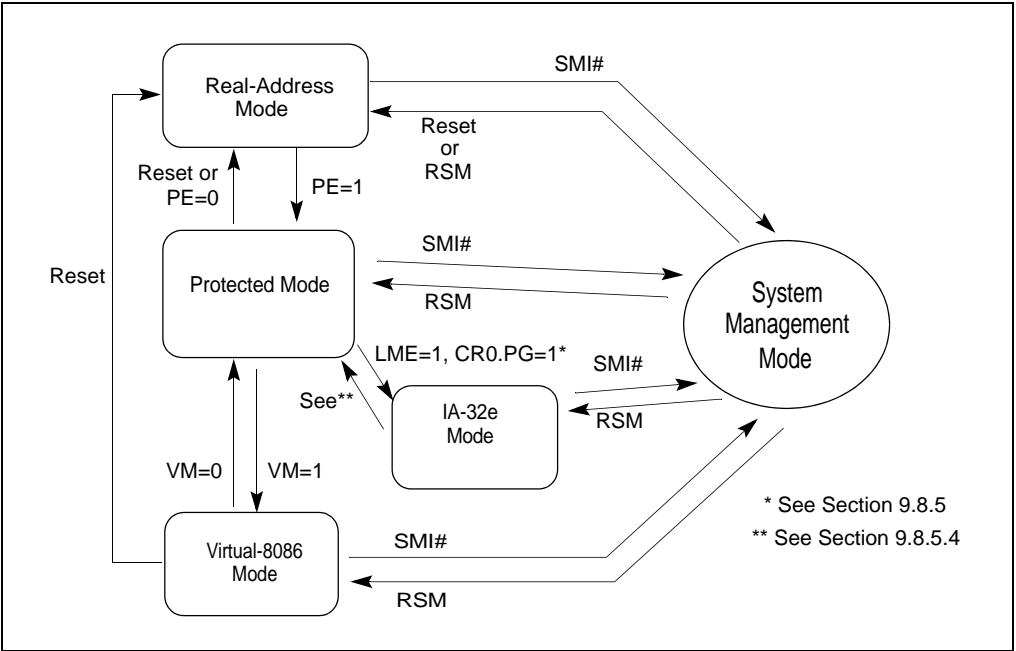


Figure 2-3. Transitions Among the Processor's Operating Modes

The processor is placed in real-address mode following power-up or a reset. The PE flag in control register CR0 then controls whether the processor is operating in real-address or protected mode. See also: Section 9.9, “Mode Switching.” and Section 4.1.2, “Paging-Mode Enabling.”

The VM flag in the EFLAGS register determines whether the processor is operating in protected mode or virtual-8086 mode. Transitions between protected mode and virtual-8086 mode are generally carried out as part of a task switch or a return from an interrupt or exception handler. See also: Section 20.2.5, “Entering Virtual-8086 Mode.”

The LMA bit (IA32\_EFER.LMA[bit 10]) determines whether the processor is operating in IA-32e mode. When running in IA-32e mode, 64-bit or compatibility sub-mode operation is determined by CS.L bit of the code segment. The processor enters into IA-32e mode from protected mode by enabling paging and setting the LME bit (IA32\_EFER.LME[bit 8]). See also: Chapter 9, “Processor Management and Initialization.”

The processor switches to SMM whenever it receives an SMI while the processor is in real-address, protected, virtual-8086, or IA-32e modes. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.



### 2.2.1 扩展功能启用寄存器

IA32\_EFER 模型特定寄存器提供了若干与IA-32e模式启用及操作相关的字段，同时还包含一个与页面访问权限修改相关的字段（参见第4.6节“访问权限”）。IA32\_EFER 模型特定寄存器的布局如图2-4所示。

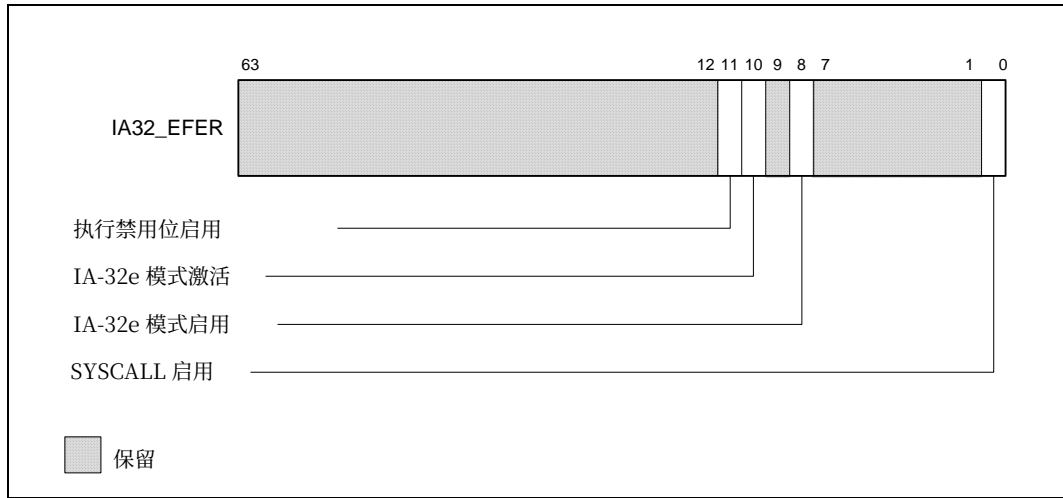


图2-4. IA32\_EFER MSR布局

表2-1. IA32_EFER MSR信息	
Bit	描述
0	SYSCALL 启用 (R/W) 在64位模式下启用SYSCALL/SYSRET指令。
7:1	保留。
8	IA-32e 模式启用 (R/W) 启用 IA-32e模式操作。
9	保留。
10	IA-32e 模式激活 (R) 置位时表示 IA-32e 模式处于激活状态。
11	执行禁用位启用 (R/W) 通过阻止从设置了XD位的PAE页面进行指令提取，启用页面访问限制（参见第4.6节）。
63:12	保留。

## 2.3 EFLAGS寄存器中的系统标志和字段

EFLAGS寄存器中的系统标志和IOPL字段控制I/O、可屏蔽硬件中断、调试、任务切换以及虚拟8086模式（参见图2-5）。只有特权代码（通常是操作系统或执行代码）才被允许修改这些位。

系统标志和IOPL包括：

TF 陷阱（第8位）——设置为启用用于调试的单步模式；清除则禁用单步模式。在单步模式下，处理器会在每条指令执行后生成调试异常。这使得程序可以在每条指令执行后检查其执行状态。如果应用程序使用

### 2.2.1 Extended Feature Enable Register

The IA32\_EFER MSR provides several fields related to IA-32e mode enabling and operation. It also provides one field that relates to page-access right modification (see Section 4.6, “Access Rights”). The layout of the IA32\_EFER MSR is shown in Figure 2-4.

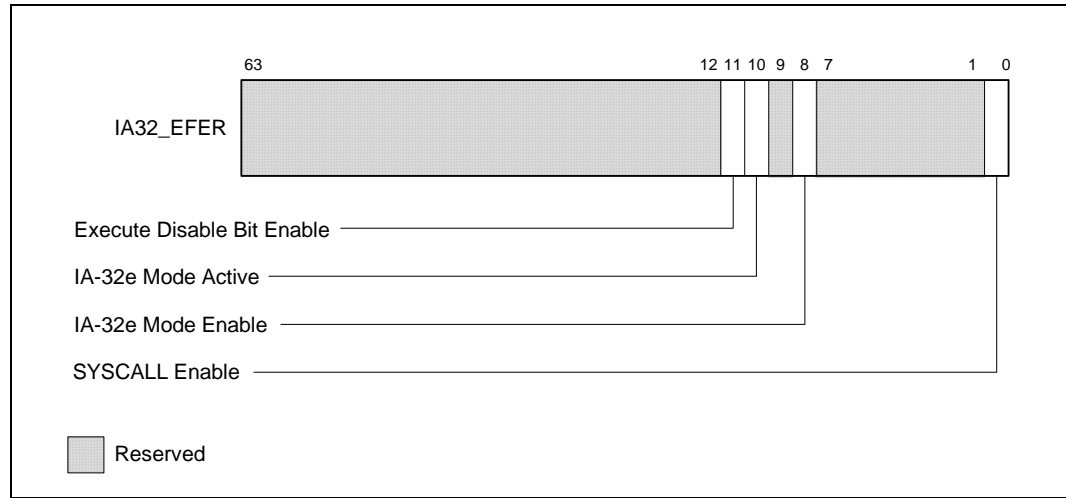


Figure 2-4. IA32\_EFER MSR Layout

Table 2-1. IA32_EFER MSR Information	
Bit	Description
0	<b>SYSCALL Enable (R/W)</b> Enables SYSCALL/SYSRET instructions in 64-bit mode.
7:1	Reserved.
8	<b>IA-32e Mode Enable (R/W)</b> Enables IA-32e mode operation.
9	Reserved.
10	<b>IA-32e Mode Active (R)</b> Indicates IA-32e mode is active when set.
11	<b>Execute Disable Bit Enable (R/W)</b> Enables page access restriction by preventing instruction fetches from PAE pages with the XD bit set (See Section 4.6).
63:12	Reserved.

## 2.3 SYSTEM FLAGS AND FIELDS IN THE EFLAGS REGISTER

The system flags and IOPL field of the EFLAGS register control I/O, maskable hardware interrupts, debugging, task switching, and the virtual-8086 mode (see Figure 2-5). Only privileged code (typically operating system or executive code) should be allowed to modify these bits.

The system flags and IOPL are:

TF     **Trap (bit 8)** — Set to enable single-step mode for debugging; clear to disable single-step mode. In single-step mode, the processor generates a debug exception after each instruction. This allows the execution state of a program to be inspected after each instruction. If an application program sets the TF flag using

POPF、POPFD 或 IRET 指令设置 TF 标志位，则会在 POPF、POPFD 或 IRET 之后的指令执行后生成调试异常。

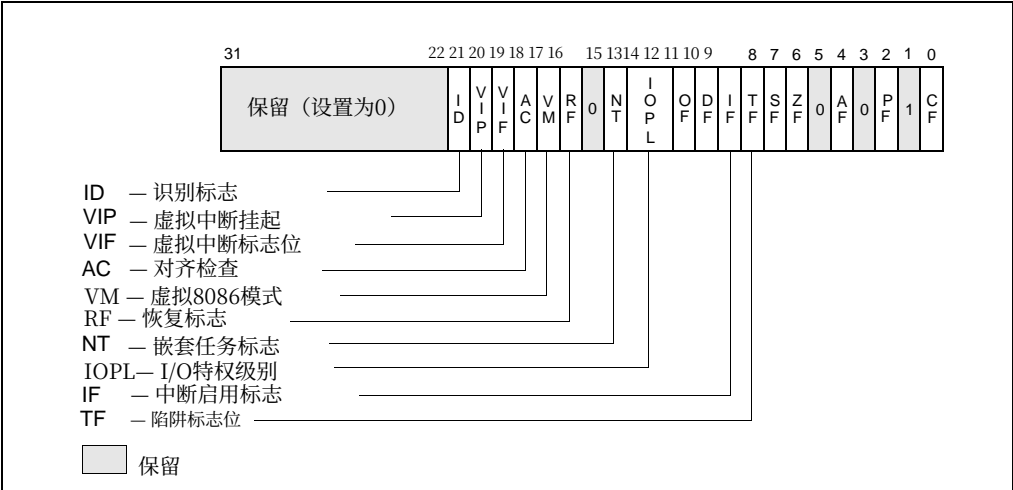


图2-5. EFLAGS寄存器中的系统标志

IF 中断使能（第9位）——控制处理器对可屏蔽硬件中断请求的响应（另见：第6.3.2节“可屏蔽硬件中断”）。该标志置位时响应可屏蔽硬件中断；清零时禁止可屏蔽硬件中断。IF标志不影响异常或不可屏蔽中断（NMI中断）的产生。当前特权级（CPL）、I/O特权级别（IOPL）以及控制寄存器CR4中的虚拟模式扩展标志（VME flag）状态决定了CLI、STI、POPF、POPFD和IRET指令是否能够修改IF标志。

IOPL I/O特权级别字段（第12和13位）——指示当前运行程序或任务的I/O特权级别（IOPL）。当前运行程序或任务的CPL必须小于或等于IOPL才能访问I/O地址空间。仅当在CPL为0运行时，POPF和IRET指令才能修改此字段。

IOPL同样是控制IF标志修改以及在虚拟模式扩展生效时（当CR4.VME = 1）处理虚拟8086模式下中断的机制之一。另请参阅：《英特尔® 64和IA-32架构软件开发人员手册》第1卷第16章“输入/输出”。

NT 嵌套任务标志（位14）——控制被中断任务和被调用任务的链式关联。处理器在通过CALL指令、中断或异常启动任务调用时设置该标志，在通过IRET指令启动任务返回时检查并修改该标志。该标志可通过POPF/POPFD指令显式设置或清除；但改变该标志状态可能导致应用程序产生意外异常。

另请参阅：第7.4节“任务链接”。

RF 恢复标志（位16）——控制处理器对指令断点条件的响应。当设置时，该标志会暂时禁止因指令断点产生调试异常（尽管其他异常条件仍可能引发异常）。当清除时，指令断点将产生调试异常。

RF标志的主要功能是允许在由指令断点条件引起的调试异常之后重新启动指令。此时，调试软件必须在通过IRETD指令返回到被中断程序之前，在堆栈上的EFLAGS寄存器映像中设置此标志（以防止指令断点引发另一个调试异常）。随后，处理器在成功执行返回的指令后会自动清除该标志，从而重新启用指令断点故障检测。

另请参阅：第17.3.1.1节“指令断点异常条件”。

VM 虚拟8086模式（位17）——置位以启用虚拟8086模式；清零则返回保护模式。

a POPF, POPFD, or IRET instruction, a debug exception is generated after the instruction that follows the POPF, POPFD, or IRET.

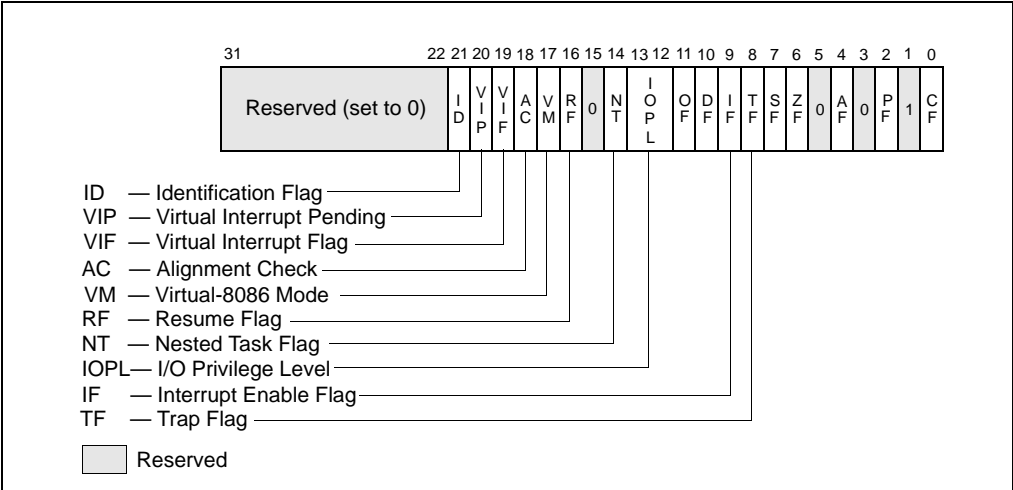


Figure 2-5. System Flags in the EFLAGS Register

IF **Interrupt enable (bit 9)** — Controls the response of the processor to maskable hardware interrupt requests (see also: Section 6.3.2, “Maskable Hardware Interrupts”). The flag is set to respond to maskable hardware interrupts; cleared to inhibit maskable hardware interrupts. The IF flag does not affect the generation of exceptions or nonmaskable interrupts (NMI interrupts). The CPL, IOPL, and the state of the VME flag in control register CR4 determine whether the IF flag can be modified by the CLI, STI, POPF, POPFD, and IRET.

IOPL **I/O privilege level field (bits 12 and 13)** — Indicates the I/O privilege level (IOPL) of the currently running program or task. The CPL of the currently running program or task must be less than or equal to the IOPL to access the I/O address space. The POPF and IRET instructions can modify this field only when operating at a CPL of 0.

The IOPL is also one of the mechanisms that controls the modification of the IF flag and the handling of interrupts in virtual-8086 mode when virtual mode extensions are in effect (when CR4.VME = 1). See also: Chapter 16, “Input/Output,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*.

NT **Nested task (bit 14)** — Controls the chaining of interrupted and called tasks. The processor sets this flag on calls to a task initiated with a CALL instruction, an interrupt, or an exception. It examines and modifies this flag on returns from a task initiated with the IRET instruction. The flag can be explicitly set or cleared with the POPF/POPFD instructions; however, changing to the state of this flag can generate unexpected exceptions in application programs.

See also: Section 7.4, “Task Linking.”

RF **Resume (bit 16)** — Controls the processor’s response to instruction-breakpoint conditions. When set, this flag temporarily disables debug exceptions (#DB) from being generated for instruction breakpoints (although other exception conditions can cause an exception to be generated). When clear, instruction breakpoints will generate debug exceptions.

The primary function of the RF flag is to allow the restarting of an instruction following a debug exception that was caused by an instruction breakpoint condition. Here, debug software must set this flag in the EFLAGS image on the stack just prior to returning to the interrupted program with IRETD (to prevent the instruction breakpoint from causing another debug exception). The processor then automatically clears this flag after the instruction returned to has been successfully executed, enabling instruction breakpoint faults again.

See also: Section 17.3.1.1, “Instruction-Breakpoint Exception Condition.”

VM **Virtual-8086 mode (bit 17)** — Set to enable virtual-8086 mode; clear to return to protected mode.

系统架构概述	
另请参阅：第20.2.1节“启用虚拟8086模式”。	
AC	<b>对齐检查（位18）</b> ——设置该标志和控制寄存器CR0中的AM标志以启用内存引用的对齐检查；清除AC标志和/或AM标志以禁用对齐检查。当引用未对齐操作数（如奇数字节地址处的字或非四的整数倍地址处的双字）时，会生成对齐检查异常。对齐检查异常仅在用户模式（特权级别3）下生成。默认特权级别0的内存引用（如段描述符加载）即使由用户模式下执行的指令引起，也不会生成此异常。
	对齐检查异常可用于检查数据的对齐情况。这在与要求所有数据对齐的处理器交换数据时非常有用。解释器也可通过对齐指针来标记某些指针为特殊指针，从而利用对齐检查异常。这消除了检查每个指针的开销，仅在特殊指针被使用时进行处理。
VIF	<b>虚拟中断（位19）</b> ——包含IF标志的虚拟映像。该标志与VIP标志配合使用。仅当控制寄存器CR4中的VME标志或PVI标志被设置且IOPL小于3时，处理器才会识别VIF标志。（VME标志启用虚拟8086模式扩展；PVI标志启用保护模式虚拟中断。）
	另请参阅：第20.3.3.5节“方法6：软件中断处理”以及第20.4节“保护模式虚拟中断”。
VIP	<b>虚拟中断挂起（位20）</b> ——由软件设置以指示有中断挂起；清零则表示无中断挂起。该标志与VIF标志配合使用。处理器会读取该标志但从不修改它。仅当控制寄存器CR4中的VME标志或PVI标志被设置且IOPL小于3时，处理器才会识别VIP标志。VME标志启用虚拟8086模式扩展；PVI标志启用保护模式虚拟中断。
	参见第20.3.3.5节“方法6：软件中断处理”和第20.4节“保护模式虚拟中断”。
ID	<b>ID标识（位21）</b> 。——程序或过程设置或清除此标志的能力表明支持CPUID指令。

### 2.3.1 IA-32e模式中的系统标志和字段

在64位模式下，RFLAGS寄存器扩展至64位，其中高32位为保留位。RFLAGS（64位模式）或EFLAGS（兼容模式）中的系统标志如图2-5所示。

在IA-32e模式下，处理器不允许设置VM位，因为不支持虚拟8086模式（尝试设置该位将被忽略）。同时，处理器也不会设置NT位。但处理器允许软件设置NT位（请注意，在IA-32e模式下如果NT位被设置，IRET会导致常规保护错误）。

在IA-32e模式下，SYSCALL/SYSRET指令具有可编程的方法来指定RFLAGS/EFLAGS寄存器中哪些位被清除。这些指令保存/恢复EFLAGS/RFLAGS寄存器。

## 2.4 内存管理寄存器

处理器提供四个内存管理寄存器（GDTR、LDTR、IDTR和TR），用于指定控制分段内存管理的数据结构的位置（参见图2-6）。系统提供了专用指令用于加载和存储这些寄存器。

SYSTEM ARCHITECTURE OVERVIEW	
See also: Section 20.2.1, “Enabling Virtual-8086 Mode.”	
AC	<b>Alignment check (bit 18)</b> — Set this flag and the AM flag in control register CR0 to enable alignment checking of memory references; clear the AC flag and/or the AM flag to disable alignment checking. An alignment-check exception is generated when reference is made to an unaligned operand, such as a word at an odd byte address or a doubleword at an address which is not an integral multiple of four. Alignment-check exceptions are generated only in user mode (privilege level 3). Memory references that default to privilege level 0, such as segment descriptor loads, do not generate this exception even when caused by instructions executed in user-mode.
	The alignment-check exception can be used to check alignment of data. This is useful when exchanging data with processors which require all data to be aligned. The alignment-check exception can also be used by interpreters to flag some pointers as special by misaligning the pointer. This eliminates overhead of checking each pointer and only handles the special pointer when used.
VIF	<b>Virtual Interrupt (bit 19)</b> — Contains a virtual image of the IF flag. This flag is used in conjunction with the VIP flag. The processor only recognizes the VIF flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. (The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.)
	See also: Section 20.3.3.5, “Method 6: Software Interrupt Handling,” and Section 20.4, “Protected-Mode Virtual Interrupts.”
VIP	<b>Virtual interrupt pending (bit 20)</b> — Set by software to indicate that an interrupt is pending; cleared to indicate that no interrupt is pending. This flag is used in conjunction with the VIF flag. The processor reads this flag but never modifies it. The processor only recognizes the VIP flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.
	See Section 20.3.3.5, “Method 6: Software Interrupt Handling,” and Section 20.4, “Protected-Mode Virtual Interrupts.”
ID	<b>Identification (bit 21)</b> . — The ability of a program or procedure to set or clear this flag indicates support for the CPUID instruction.

### 2.3.1 System Flags and Fields in IA-32e Mode

In 64-bit mode, the RFLAGS register expands to 64 bits with the upper 32 bits reserved. System flags in RFLAGS (64-bit mode) or EFLAGS (compatibility mode) are shown in Figure 2-5.

In IA-32e mode, the processor does not allow the VM bit to be set because virtual-8086 mode is not supported (attempts to set the bit are ignored). Also, the processor will not set the NT bit. The processor does, however, allow software to set the NT bit (note that an IRET causes a general protection fault in IA-32e mode if the NT bit is set).

In IA-32e mode, the SYSCALL/SYSRET instructions have a programmable method of specifying which bits are cleared in RFLAGS/EFLAGS. These instructions save/restore EFLAGS/RFLAGS.

## 2.4 MEMORY-MANAGEMENT REGISTERS

The processor provides four memory-management registers (GDTR, LDTR, IDTR, and TR) that specify the locations of the data structures which control segmented memory management (see Figure 2-6). Special instructions are provided for loading and storing these registers.

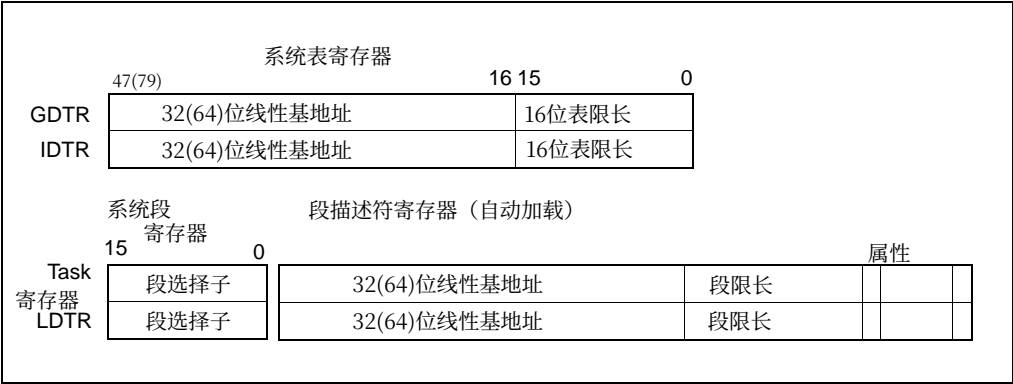


图2-6. 内存管理寄存器

2.4.1 全局描述符表寄存器 (GDTR)

GDTR寄存器存储GDT的基地址（保护模式下为32位；IA-32e模式下为64位）和16位表限长。基地址指定GDT字节0的线性地址；表限长指定表中的字节数。

LGDT和SGDT指令分别用于加载和存储GDTR寄存器。在处理器上电或重置时，基地址被设置为默认值0，界限被设置为0FFFFH。作为保护模式操作处理器初始化过程的一部分，必须将新的基地址加载到GDTR中。

另请参阅：第3.5.1节“段描述符表”。

2.4.2 局部描述符表寄存器 (LDTR)

LDTR寄存器保存着LDT的16位段选择子、基地址（保护模式下为32位；IA-32e模式下为64位）、段限长以及描述符属性。基地址指定了LDT段字节0的线性地址；段限长指定了段中的字节数。另请参阅：第3.5.1节“段描述符表”。

LLDT和SLDT指令分别加载和存储LDTR寄存器的段选择子部分。包含LDT的段必须在GDT中具有段描述符。当LLDT指令将段选择子加载到LDTR时：LDT描述符中的基地址、限长和描述符属性会被自动加载到LDTR中。

当发生任务切换时，LDTR会自动加载新任务的LDT段选择子和描述符。在将新的LDT信息写入寄存器之前，LDTR的内容不会自动保存。

在处理器上电或重置时，段选择子和基地址被设置为默认值0，表限长被设置为0FFFFH。

2.4.3 IDTR 中断描述符表寄存器

IDTR寄存器保存着IDT的基地址（保护模式下为32位；IA-32e模式下为64位）和16位表限长。基地址指定了IDT字节0的线性地址；表限长指定了表中的字节数。LIDT和SIDT指令分别用于加载和存储IDTR寄存器。在处理器上电或重置时，基地址被设置为默认值0，表限长被设置为0FFFFH。随后，寄存器中的基地址和限长可以作为处理器初始化过程的一部分进行更改。

另请参阅：第6.10节，“中断描述符表 (IDT)”。

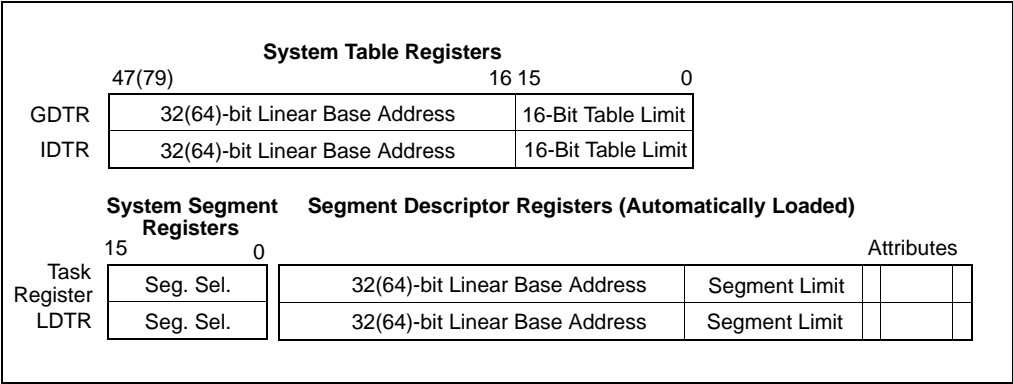


Figure 2-6. Memory Management Registers

2.4.1 Global Descriptor Table Register (GDTR)

The GDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and the 16-bit table limit for the GDT. The base address specifies the linear address of byte 0 of the GDT; the table limit specifies the number of bytes in the table.

The LGDT and SGDT instructions load and store the GDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. A new base address must be loaded into the GDTR as part of the processor initialization process for protected-mode operation.

See also: Section 3.5.1, “Segment Descriptor Tables.”

2.4.2 Local Descriptor Table Register (LDTR)

The LDTR register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the LDT. The base address specifies the linear address of byte 0 of the LDT segment; the segment limit specifies the number of bytes in the segment. See also: Section 3.5.1, “Segment Descriptor Tables.”

The LLDT and SLDT instructions load and store the segment selector part of the LDTR register, respectively. The segment that contains the LDT must have a segment descriptor in the GDT. When the LLDT instruction loads a segment selector in the LDTR: the base address, limit, and descriptor attributes from the LDT descriptor are automatically loaded in the LDTR.

When a task switch occurs, the LDTR is automatically loaded with the segment selector and descriptor for the LDT for the new task. The contents of the LDTR are not automatically saved prior to writing the new LDT information into the register.

On power up or reset of the processor, the segment selector and base address are set to the default value of 0 and the limit is set to 0FFFFH.

2.4.3 IDTR Interrupt Descriptor Table Register

The IDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and 16-bit table limit for the IDT. The base address specifies the linear address of byte 0 of the IDT; the table limit specifies the number of bytes in the table. The LIDT and SIDT instructions load and store the IDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. The base address and limit in the register can then be changed as part of the processor initialization process.

See also: Section 6.10, “Interrupt Descriptor Table (IDT).”



### 2.4.4 任务寄存器(TR)

任务寄存器保存当前任务的TSS的16位段选择子、基地址（保护模式下为32位；IA-32e模式下为64位）、段限长和描述符属性。该选择子引用GDT中的TSS描述符。基地址指定TSS字节0的线性地址；段限长指定TSS中的字节数。另请参阅：第7.2.4节，“任务寄存器”。

LTR和STR指令分别加载和存储任务寄存器的段选择子部分。当LTR指令将段选择子加载到任务寄存器时，TSS描述符中的基地址、界限和描述符属性会自动加载到任务寄存器中。在处理器上电或重置时，基地址被设置为默认值0，界限被设置为0FFFFH。

当任务切换发生时，任务寄存器会自动加载新任务的TSS段选择子和描述符。在将新的TSS信息写入寄存器之前，任务寄存器的内容不会自动保存。

## 2.5 控制寄存器

控制寄存器（CR0、CR1、CR2、CR3和CR4；参见图2-7）决定了处理器的操作模式及当前执行任务的特性。这些寄存器在所有32位模式和兼容模式下均为32位。

在64位模式下，控制寄存器扩展至64位。MOV CRn指令用于操作寄存器位，这些指令的操作数大小前缀将被忽略。同时存在以下特性：

- CR0和CR4的63:32位为保留位且必须写入零。向高32位中的任意位写入非零值将触发通用保护异常，#GP(0)。

- CR2的所有64位均可由软件写入。
- CR3的51:40位为保留位且必须为0。
- MOV CRn指令不会检查写入CR2和CR3的地址是否在实现的线性地址或物理地址限制范围内。

- 控制寄存器CR8仅在64位模式下可用。

控制寄存器总结如下，这些控制寄存器中每个架构定义的控制字段都将单独说明。在图2-7中，括号内标示了该寄存器在64位模式下的宽度（CR0除外）。

- CR0 — 包含系统控制标志，用于控制处理器的操作模式和状态。
- CR1 — 保留。
- CR2 — 包含页故障线性地址（引发页故障的线性地址）。
- CR3 — 包含页结构层次基地址的物理地址和两个标志（PCD和PWT）。仅指定基地址的最高有效位（低12位除外）；地址的低12位假定为0。因此，第一个页表结构必须按页（4KB）边界对齐。PCD和PWT标志控制该页表结构在处理器内部数据缓存中的缓存（它们不控制页目录信息的TLB缓存）。当使用物理地址扩展时，CR3寄存器包含页目录指针表的基地址。在IA-32e模式下，CR3寄存器包含PML4表的基地址。另请参阅：第4章“分页”。

- CR4——包含一组标志，这些标志启用若干架构扩展，并表明操作系统或执行程序对特定处理器能力的支持。控制寄存器可使用MOV指令的移至或移自控制寄存器形式进行读取和加载（或修改）。在保护模式下，MOV指令允许读取或加载控制寄存器（仅限特权级别0）。此限制意味着应用程序或操作系统过程（在特权级别1、2或3下运行）被禁止读取或加载控制寄存器。

### 2.4.4 Task Register (TR)

The task register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the TSS of the current task. The selector references the TSS descriptor in the GDT. The base address specifies the linear address of byte 0 of the TSS; the segment limit specifies the number of bytes in the TSS. See also: Section 7.2.4, “Task Register.”

The LTR and STR instructions load and store the segment selector part of the task register, respectively. When the LTR instruction loads a segment selector in the task register, the base address, limit, and descriptor attributes from the TSS descriptor are automatically loaded into the task register. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH.

When a task switch occurs, the task register is automatically loaded with the segment selector and descriptor for the TSS for the new task. The contents of the task register are not automatically saved prior to writing the new TSS information into the register.

## 2.5 CONTROL REGISTERS

Control registers (CR0, CR1, CR2, CR3, and CR4; see Figure 2-7) 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 2-7, 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——提供对任务优先级寄存器的读写访问。它指定了优先级阈值，操作系统使用该阈值来控制允许中断处理器的外部中断的优先级类别。此寄存器仅在64位模式下可用。然而，在兼容模式下，中断过滤仍然适用。

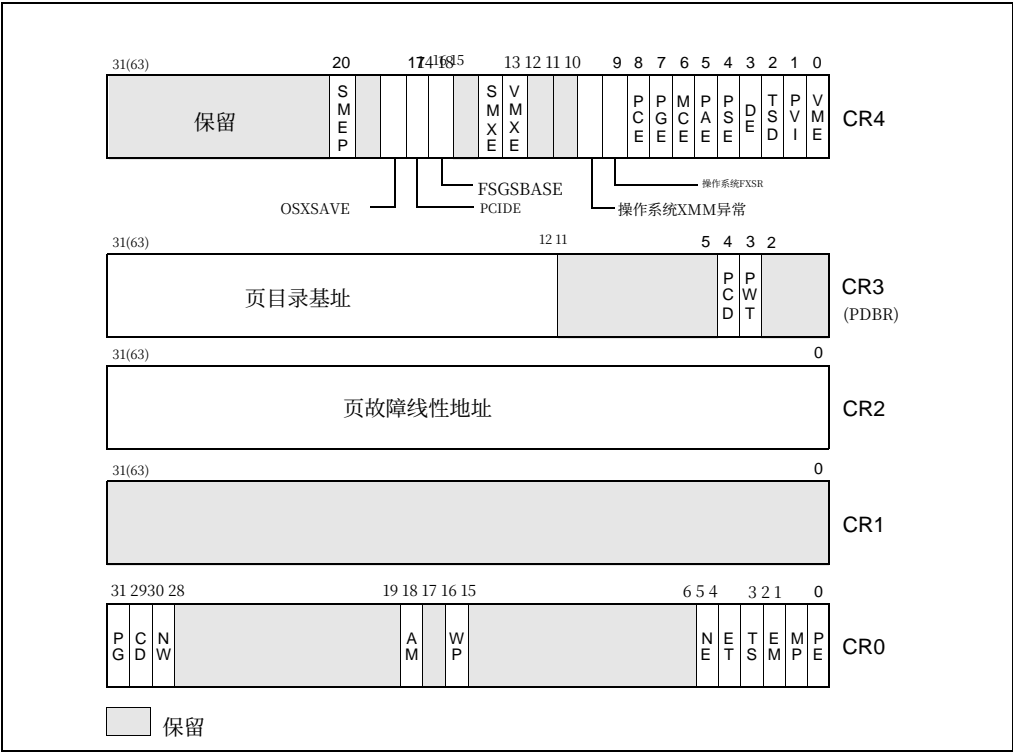


图2-7. 控制寄存器

加载控制寄存器时，保留位应始终设置为先前读取的值。控制寄存器中的标志包括：

PG 分页（CR0的位31）——置位时启用分页；清零时禁用分页。当分页被禁用时，所有线性地址均被视为物理地址。如果PE标志（寄存器CR0的位0）未同时置位，则PG标志不起作用；在PE标志清零时设置PG标志会导致通用保护异常（#GP）。另请参阅：第4章“分页”。在Intel 64处理器上，启用和禁用IA-32e模式操作也需要修改CR0.PG。CD 缓存禁用（CR0的位30）——当CD和NW标志清零时，处理器内部（及外部）缓存中对整个物理内存的内存位置缓存被启用。当CD标志置位时，缓存会受到限制，详见表11-5。为防止处理器访问和更新其缓存，必须设置CD标志并使缓存失效，以确保不会发生缓存命中。另请参阅：第11.5.3节“防止缓存”和第11.5节“缓存控制”。NW 非直写（CR0的位29）——当NW和CD标志清零时，对于命中缓存的写入启用写回（针对奔腾4、英特尔至强、P6系列和奔腾处理器）或直写（针对Intel486处理器），并启用失效周期。关于CD和NW标志其他设置下NW标志对缓存影响的具体信息，请参阅表11-5。AM 对齐掩码（CR0的位18）——置位时启用自动对齐检查；清零时禁用对齐检查。仅当AM标志置位、EFLAGS寄存器中的AC标志置位、CPL为3且处理器处于保护模式或虚拟8086模式操作时，才会执行对齐检查。

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

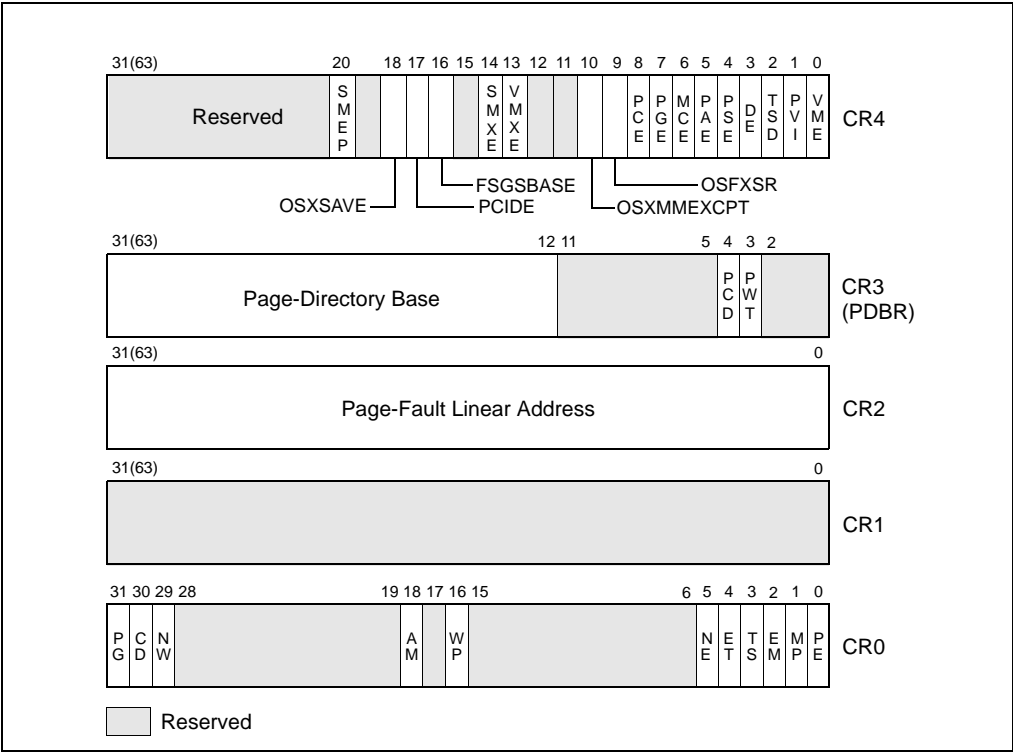


Figure 2-7. Control Registers

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

- PG **Paging (bit 31 of CR0)** — Enables paging when set; disables paging when clear. When paging is disabled, all linear addresses are treated as physical addresses. The PG flag has no effect if the PE flag (bit 0 of register CR0) is not also set; setting the PG flag when the PE flag is clear causes a general-protection exception (#GP). See also: Chapter 4, “Paging.”
- On Intel 64 processors, enabling and disabling IA-32e mode operation also requires modifying CR0.PG.
- CD **Cache Disable (bit 30 of CR0)** — When the CD and NW flags are clear, caching of memory locations for the whole of physical memory in the processor’s internal (and external) caches is enabled. When the CD flag is set, caching is restricted as described in Table 11-5. To prevent the processor from accessing and updating its caches, the CD flag must be set and the caches must be invalidated so that no cache hits can occur.
- See also: Section 11.5.3, “Preventing Caching,” and Section 11.5, “Cache Control.”
- NW **Not Write-through (bit 29 of CR0)** — When the NW and CD flags are clear, write-back (for Pentium 4, Intel Xeon, P6 family, and Pentium processors) or write-through (for Intel486 processors) is enabled for writes that hit the cache and invalidation cycles are enabled. See Table 11-5 for detailed information about the affect of the NW flag on caching for other settings of the CD and NW flags.
- AM **Alignment Mask (bit 18 of CR0)** — Enables automatic alignment checking when set; disables alignment checking when clear. Alignment checking is performed only when the AM flag is set, the AC flag in the EFLAGS register is set, CPL is 3, and the processor is operating in either protected or virtual-8086 mode.

系统架构概述		SYSTEM ARCHITECTURE OVERVIEW
WP	WP	<b>Write Protect (bit 16 of CR0)</b> — When set, inhibits supervisor-level procedures from writing into read-only pages; when clear, allows supervisor-level procedures to write into read-only pages (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.
NE	NE	<b>Numeric Error (bit 5 of CR0)</b> — Enables the native (internal) mechanism for reporting x87 FPU errors when set; enables the PC-style x87 FPU error reporting mechanism when clear. When the NE flag is clear and the IGNNE# input is asserted, x87 FPU errors are ignored. When the NE flag is clear and the IGNNE# input is deasserted, an unmasked x87 FPU error causes the processor to assert the FERR# pin to generate an external interrupt and to stop instruction execution immediately before executing the next waiting floating-point instruction or WAIT/FWAIT instruction.
		The FERR# pin is intended to drive an input to an external interrupt controller (the FERR# pin emulates the ERROR# pin of the Intel 287 and Intel 387 DX math coprocessors). The NE flag, IGNNE# pin, and FERR# pin are used with external logic to implement PC-style error reporting. Using FERR# and IGNNE# to handle floating-point exceptions is deprecated by modern operating systems; this non-native approach also limits newer processors to operate with one logical processor active.
		See also: “Software Exception Handling” in Chapter 8, “Programming with the x87 FPU,” and Appendix A, “EFLAGS Cross-Reference,” in the <i>Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1</i> .
ET	ET	<b>Extension Type (bit 4 of CR0)</b> — Reserved in the Pentium 4, Intel Xeon, P6 family, and Pentium processors. In the Pentium 4, Intel Xeon, and P6 family processors, this flag is hardcoded to 1. In the Intel386 and Intel486 processors, this flag indicates support of Intel 387 DX math coprocessor instructions when set.
TS	TS	<b>Task Switched (bit 3 of CR0)</b> — Allows the saving of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context on a task switch to be delayed until an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction is actually executed by the new task. The processor sets this flag on every task switch and tests it when executing x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.
		<ul style="list-style-type: none"><li>If the TS flag is set and the EM flag (bit 2 of CR0) is clear, a device-not-available exception (#NM) is raised prior to the execution of any x87 FPU/MMX/SSE/ SSE2/SSE3/SSSE3/SSE4 instruction; with the exception of PAUSE, PREFETCH<math>h</math>, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. See the paragraph below for the special case of the WAIT/FWAIT instructions.</li><li>If the TS flag is set and the MP flag (bit 1 of CR0) and EM flag are clear, an #NM exception is not raised prior to the execution of an x87 FPU WAIT/FWAIT instruction.</li><li>If the EM flag is set, the setting of the TS flag has no affect on the execution of x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.</li></ul>
		Table 2-2 shows the actions taken when the processor encounters an x87 FPU instruction based on the settings of the TS, EM, and MP flags. Table 12-1 and 13-1 show the actions taken when the processor encounters an MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction.
		The processor does not automatically save the context of the x87 FPU, XMM, and MXCSR registers on a task switch. Instead, it sets the TS flag, which causes the processor to raise an #NM exception whenever it encounters an x87 FPU/MMX/SSE /SSE2/SSE3/SSSE3/SSE4 instruction in the instruction stream for the new task (with the exception of the instructions listed above).
		The fault handler for the #NM exception can then be used to clear the TS flag (with the CLTS instruction) and save the context of the x87 FPU, XMM, and MXCSR registers. If the task never encounters an x87 FPU/MMX/SSE/SSE2/SSE3//SSSE3/SSE4 instruction; the x87 FPU/MMX/SSE/SSE2/ SSE3/SSSE3/SSE4 context is never saved.

WP 写保护（CR0的位16）——置位时，禁止超级用户级过程写入只读页；清零时，允许超级用户级过程写入只读页（无论U/S位设置如何；参见第4.1.3节和第4.6节）。此标志有助于操作系统（如UNIX）实现创建新进程（分叉）时使用的写时复制方法。

NE 数值错误（CR0的第5位）——当设置时启用用于报告x87浮点处理单元错误的原生（内部）机制；当清除时启用PC式x87浮点处理单元错误报告机制。当NE标志被清除且IGNNE#输入被置位时，x87浮点处理单元错误将被忽略。当NE标志被清除且IGNNE#输入被取消置位时，未屏蔽的x87浮点处理单元错误将导致处理器置位FERR#引脚以产生外部中断，并在执行下一条等待浮点指令或WAIT/FWAIT指令前立即停止指令执行。

FERR#引脚旨在驱动外部中断控制器的输入（FERR#引脚模拟了Intel 287和Intel 387 DX数学协处理器的ERROR#引脚）。NE标志、IGNNE#引脚和FERR#引脚与外部逻辑配合使用以实现PC式错误报告。现代操作系统已弃用使用FERR#和IGNNE#处理浮点异常的方法；这种非原生方法还限制了新一代处理器只能在一个活跃逻辑处理器下运行。

另请参阅：《Intel® 64和IA-32架构软件开发人员手册》第1卷第8章“使用x87浮点处理单元编程”中的“软件异常处理”以及附录A“EFLAGS寄存器交叉参考”。

ET扩展类型（CR0的第4位）——在奔腾4、英特尔至强、P6系列和奔腾处理器中保留。在奔腾4、英特尔至强和P6系列处理器中，此标志硬编码为1。在Intel386和Intel486处理器中，该标志置位时表示支持Intel 387 DX数学协处理器指令。

TS任务切换（CR0的第3位）——允许将x87浮点处理单元/MMX/SSE/SSE2/SSE3/SSSE3/SSE4上下文的保存延迟到新任务实际执行x87浮点处理单元/MMX/SSE/SSE2/SSE3/SSSE3/SSE4指令时。处理器在每次任务切换时设置此标志，并在执行x87浮点处理单元/MMX/SSE/SSE2/SSE3/SSSE3/SSE4指令时测试该标志。

- 如果设置了任务切换标志且清除了EM标志（CR0的第2位），在执行任何x87浮点处理单元/MMX/SSE/SSE2/SSE3/SSSE3/SSE4指令之前会引发设备不可用异常；但PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、MOVNTI、CLFLUSH、CRC32和POPCNT指令除外。关于WAIT/FWAIT指令的特殊情况，请参阅下文段落。

- 如果设置了任务切换标志且清除了MP标志（CR0的第1位）和EM标志，在执行x87浮点处理单元的WAIT/FWAIT指令前不会引发#NM异常。
- 如果设置了EM标志，任务切换标志的设置对x87浮点处理单元/MMX/SSE/SSE2/SSE3/SSSE3/SSE4指令的执行没有影响。

表2-2展示了当处理器遇到x87浮点处理单元指令时，根据TS、EM和MP标志的设置所采取的操作。表12-1和13-1则展示了当处理器遇到MMX/SSE/SSE2/SSE3/SSSE3/SSE4指令时所采取的操作。

处理器不会在任务切换时自动保存x87浮点处理单元、XMM寄存器以及MXCSR寄存器的上下文。相反，它会设置TS标志，这将导致处理器在新任务的指令流中遇到x87浮点处理单元/MMX/SSE/SSE2/SSE3/SSSE3/SSE4指令时（除上述列出的指令外）引发#NM异常。

#NM异常的故障处理程序随后可用于清除任务切换标志（通过CLTS指令）并保存x87浮点处理单元、XMM寄存器及MXCSR寄存器的上下文。若任务从未遇到x87浮点处理单元/MMX/SSE/SSE2/SSE3/SSSE3/SSE4指令，则其上下文将永不保存。



表2-2。 x87浮点处理单元指令针对EM、MP和TS不同组合所采取的操作

CR0标志位			x87浮点处理单元指令类型	
EM	MP	TS	浮点	等待/浮点等待
0	0	0	执行	执行。
0	0	1	#NM异常	执行。
0	1	0	执行	执行。
0	1	1	#NM异常	#NM异常。
1	0	0	#NM异常	执行。
1	0	1	#NM异常	执行。
1	1	0	#NM异常	执行。
1	1	1	#NM异常	#NM异常。

EM仿真（CR0的第2位）——当置位时表示处理器无内置或外部x87浮点处理单元；清零时表示存在x87浮点处理单元。该标志同时影响MMX/SSE/SSE2/SSE3/SSSE3/SSE4指令的执行。

当EM标志被设置时，执行x87浮点处理单元指令会引发设备不可用异常。当处理器没有内部x87浮点处理单元或未连接外部数学协处理器时，必须设置此标志。设置该标志会强制所有浮点指令通过软件模拟处理。表9-2展示了根据系统中存在的IA-32处理器和x87浮点处理单元或数学协处理器，该标志的推荐设置。表2-2展示了EM、MP和TS标志的交互作用。

此外，当EM标志被设置时，执行MMX指令会导致产生无效操作码异常（参见表12-1）。因此，如果IA-32或Intel 64处理器集成了MMX技术，必须将EM标志设置为0才能启用MMX指令的执行。

同样地，对于SSE/SSE2/SSE3/SSSE3/SSE4扩展，当EM标志被设置时，执行大多数SSE/SSE2/SSE3/SSSE3/SSE4指令会导致生成无效操作码异常（#UD）（参见表13-1）。如果IA-32或Intel 64处理器集成了SSE/SSE2/SSE3/SSSE3/SSE4扩展，则必须将EM标志设置为0以启用这些扩展的执行。不受EM标志影响的SSE/SSE2/SSE3/SSSE3/SSE4指令包括：PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、MOVNTI、CLFLUSH、CRC32和POPCNT。

MP监控协处理器（CR0的第1位）。——控制WAIT（或FWAIT）指令与TS标志（CR0的第3位）的交互。如果MP标志被设置，且TS标志也被设置，则WAIT指令会生成设备不可用异常（#NM）。如果MP标志被清除，WAIT指令将忽略TS标志的设置。表9-2显示了根据系统中存在的IA-32处理器和x87浮点处理单元或数学协处理器，推荐设置的此标志。表2-2显示了MP、EM和TS标志的交互作用。

PE 保护模式使能（CR0的第0位）——置位时启用保护模式；清零时启用实地址模式。该标志不直接启用分页功能，仅启用段级保护机制。要启用分页，必须同时设置PE和PG标志。

另请参阅：第9.9节“模式切换”。

PCD 页级缓存禁用（CR3的第4位）——控制用于访问当前页结构层次中首个页表结构的内存类型。参见第4.9节“分页与内存类型”。当分页被禁用、使用PAE分页，或使用IA-32e分页且CR4.PCIDE=1时，此位无效。

PWT 页级写通（CR3的第3位）——控制用于访问当前页结构层次中首个页表结构的内存类型。参见第4.9节“分页与内存类型”。当分页被禁用、使用PAE分页，或使用IA-32e分页且CR4.PCIDE=1时，此位无效。

VME 虚拟8086模式扩展（CR4的第0位）——当置位时启用虚拟8086模式下的中断和异常处理扩展；当清零时禁用这些扩展。使用虚拟模式扩展可以通过消除调用虚拟8086监视器来处理执行8086程序时发生的中断和异常的开销，并将这些中断和异常重定向回8086程序的处理程序，从而提高虚拟8086应用程序的性能。它还提供

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

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

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

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

Also, when the EM flag is set, execution of an MMX instruction causes an invalid-opcode exception (#UD) to be generated (see Table 12-1). Thus, if an IA-32 or Intel 64 processor incorporates MMX technology, the EM flag must be set to 0 to enable execution of MMX instructions.

Similarly for SSE/SSE2/SSE3/SSSE3/SSE4 extensions, when the EM flag is set, execution of most SSE/SSE2/SSE3/SSSE3/SSE4 instructions causes an invalid opcode exception (#UD) to be generated (see Table 13-1). If an IA-32 or Intel 64 processor incorporates the SSE/SSE2/SSE3/SSSE3/SSE4 extensions, the EM flag must be set to 0 to enable execution of these extensions. SSE/SSE2/SSE3/SSSE3/SSE4 instructions not affected by the EM flag include: PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

MP **Monitor Coprocessor (bit 1 of CR0)**. — Controls the interaction of the WAIT (or FWAIT) instruction with the TS flag (bit 3 of CR0). If the MP flag is set, a WAIT instruction generates a device-not-available exception (#NM) if the TS flag is also set. If the MP flag is clear, the WAIT instruction ignores the setting of the TS flag. Table 9-2 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 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 the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 4.9, “Paging and Memory Typing”. This bit is not used if paging is disabled, with PAE paging, or with IA-32e paging if CR4.PCIDE=1.

PWT **Page-level Write-Through (bit 3 of CR3)** — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 4.9, “Paging and Memory Typing”. This bit is not used if paging is disabled, with PAE paging, or with IA-32e paging if CR4.PCIDE=1.

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



	对虚拟中断标志（VIF）的硬件支持，以提高在多任务和多处理器环境中运行8086程序的可靠性。
	另请参阅：第20.3节“虚拟8086模式下的中断与异常处理”。
PVI	保护模式虚拟中断（CR4寄存器的第1位）——设置时启用保护模式下虚拟中断标志位（VIF）的硬件支持；清除时禁用保护模式下的VIF标志位。
	另请参阅：第20.4节“保护模式虚拟中断”。TSD 时间戳禁用（CR4寄存器的第2位）——设置时将RDTSC指令的执行限制在特权级别0运行的进程中；清除时允许RDTSC指令在任何特权级执行。如果支持RDTSCP指令（如果CPUID.80000001H:EDX[27] = 1），此位同样适用于RDTSCP指令。DE 调试扩展（CR4寄存器的第3位）——设置时对调试寄存器DR4与DR5的引用会引发未定义操作码（#UD异常）；清除时，处理器将对寄存器DR4和DR5的引用进行别名处理，以兼容早期IA-32处理器上运行的软件。另请参阅：第17.2.2节“调试寄存器DR4与DR5”。PSE 页大小扩展（CR4寄存器的第4位）——设置时启用32位分页的4MB页；限制
	32位分页将页大小限制为4KB（当该位清零时）。
	另请参阅：第4.3节“32位分页”。
PAE	物理地址扩展（CR4寄存器的第5位）——当置位时，启用分页以生成超过32位的物理地址。当清零时，将物理地址限制为32位。在进入IA-32e模式前必须设置PAE。
	另请参阅：第4章“分页”。
MCE	机器检查启用（CR4寄存器的第6位）——当置位时启用机器检查异常；当清零时禁用清除时产生机器检查异常。
	另请参阅：第15章“机器检查架构”。
PGE	页全局启用（CR4的第7位）——（在P6系列处理器中引入。）设置时启用全局页功能；清除时禁用全局页功能。全局页功能允许将频繁使用或共享的页面标记为对所有用户全局（通过页目录或页表条目中的全局标志，第8位实现）。全局页面在任务切换或写入寄存器CR3时不会从转译后备缓冲器（TLB）中刷新。
	启用全局页功能时，必须在设置PGE标志之前启用分页（通过设置控制寄存器CR0中的PG标志）。颠倒此顺序可能会影响程序正确性，并且处理器性能将受到影响。
	另请参阅：第4.10节“缓存转换信息”。
PCE	性能监控计数器启用（CR4的第8位）——设置时允许在任何保护级别下运行的程序或过程执行RDPMC指令；清除时RDPMC指令只能在保护级别0执行。

操作系统FXSR O

	操作系统对FXSAVE和FXRSTOR指令的支持（CR4的第9位）——当设置时，该标志：(1)向软件表明操作系统支持使用FXSAVE和FXRSTOR指令，(2)使FXSAVE和FXRSTOR指令能够保存和恢复XMM和MXCSR寄存器的内容，以及x87浮点处理单元和MMX寄存器的内容，并且(3)使处理器能够执行SSE/SSE2/SSE3/SSSE3/SSE4指令，但PAUSE、预取、存储屏障、加载屏障、内存屏障、MOVNTI、缓存行刷新、CRC32和位计数指令除外。
	若该标志位被清除，FXSAVE和FXRSTOR指令将保存并恢复x87浮点处理单元和MMX指令的内容，但可能不会保存并恢复XMM寄存器和MXCSR寄存器的内容。此外，若处理器尝试执行任何SSE/SSE2/SSE3指令（除PAUSE、PREFETCHh、SFENCE、LFENCE、MFENCE、MOVNTI、CLFLUSH、CRC32及POPCNT外），将产生无效操作码异常（#UD异常）。操作系统或执行体必须显式设置此标志位。

	hardware support for a virtual interrupt flag (VIF) to improve reliability of running 8086 programs in multi-tasking and multiple-processor environments.
	See also: Section 20.3, “Interrupt and Exception Handling in Virtual-8086 Mode.”
PVI	<b>Protected-Mode Virtual Interrupts (bit 1 of CR4)</b> — Enables hardware support for a virtual interrupt flag (VIF) in protected mode when set; disables the VIF flag in protected mode when clear.
	See also: Section 20.4, “Protected-Mode Virtual Interrupts.”
TSD	<b>Time Stamp Disable (bit 2 of CR4)</b> — Restricts the execution of the RDTSC instruction to procedures running at privilege level 0 when set; allows RDTSC instruction to be executed at any privilege level when clear. This bit also applies to the RDTSCP instruction if supported (if CPUID.80000001H:EDX[27] = 1).
DE	<b>Debugging Extensions (bit 3 of CR4)</b> — References to debug registers DR4 and DR5 cause an undefined opcode (#UD) exception to be generated when set; when clear, processor aliases references to registers DR4 and DR5 for compatibility with software written to run on earlier IA-32 processors.
	See also: Section 17.2.2, “Debug Registers DR4 and DR5.”
PSE	<b>Page Size Extensions (bit 4 of CR4)</b> — 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	<b>Physical Address Extension (bit 5 of CR4)</b> — 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.”
MCE	<b>Machine-Check Enable (bit 6 of CR4)</b> — Enables the machine-check exception when set; disables the machine-check exception when clear.
	See also: Chapter 15, “Machine-Check Architecture.”
PGE	<b>Page Global Enable (bit 7 of CR4)</b> — (Introduced in the P6 family processors.) Enables the global page feature when set; disables the global page feature when clear. The global page feature allows frequently used or shared pages to be marked as global to all users (done with the global flag, bit 8, in a page-directory or page-table entry). Global pages are not flushed from the translation-lookaside buffer (TLB) on a task switch or a write to register CR3.
	When enabling the global page feature, paging must be enabled (by setting the PG flag in control register CR0) before the PGE flag is set. Reversing this sequence may affect program correctness, and processor performance will be impacted.
	See also: Section 4.10, “Caching Translation Information.”
PCE	<b>Performance-Monitoring Counter Enable (bit 8 of CR4)</b> — Enables execution of the RDPMC instruction for programs or procedures running at any protection level when set; RDPMC instruction can be executed only at protection level 0 when clear.
OSFXSR	<b>Operating System Support for FXSAVE and FXRSTOR instructions (bit 9 of CR4)</b> — When set, this flag: (1) indicates to software that the operating system supports the use of the FXSAVE and FXRSTOR instructions, (2) enables the FXSAVE and FXRSTOR instructions to save and restore the contents of the XMM and MXCSR registers along with the contents of the x87 FPU and MMX registers, and (3) enables the processor to execute SSE/SSE2/SSE3/SSSE3/SSE4 instructions, with the exception of the PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.
	If this flag is clear, the FXSAVE and FXRSTOR instructions will save and restore the contents of the x87 FPU and MMX instructions, but they may not save and restore the contents of the XMM and MXCSR registers. Also, the processor will generate an invalid opcode exception (#UD) if it attempts to execute any SSE/SSE2/SSE3 instruction, with the exception of PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. The operating system or executive must explicitly set this flag.

NOTE

CPUID指令功能标志FXSR表示FXSAVE/FXRSTOR指令的可用性。OSFXSR位为操作系统软件提供了一种启用FXSAVE/FXRSTOR以保存/恢复X87浮点处理单元、XMM寄存器及MXCSR寄存器内容的方法。因此，OSFXSR位表明操作系统为SSE/SSE2/SSE3/SSSE3/SSE4提供了上下文切换支持。

OSXMMEXCPT Opera

操作系统对未屏蔽SIMD浮点异常的支持（CR4的位10）——当设置时，表示操作系统支持通过异常处理程序处理未屏蔽的SIMD浮点异常，该处理程序在SIMD浮点异常（#XF）产生时被调用。SIMD浮点异常仅由SSE/SSE2/SSE3/SSE4.1 SIMD浮点指令产生。操作系统或执行体必须显式设置此标志。如果此标志未设置，则处理器每当检测到未屏蔽的SIMD浮点异常时，将产生无效操作码异常（#UD）。VMXE VMX启用位（CR4的位13）——设置时启用VMX操作。参见第23章“虚拟机扩展介绍”。SMXE SMX启用位（CR4的位14）——设置时启用SMX操作。参见《英特尔® 64和IA-32架构软件开发人员手册》第2C卷第5章“安全模式扩展参考”。FSGSBASE FSGSBASE启用位（CR4的位16）——启用指令RDFSBASE、RDGSBASE、WRFSBASE和WRGSBASE。PCIDE PCID启用位（CR4的位17）——设置时启用进程上下文标识符（PCID）。参见第4.10.1节“进程上下文标识符（PCID）”。仅在IA-32e模式下可设置（如果IA32\_EFER.LMA = 1）。OSXSAVE XSAVE和处理器扩展状态启用位（CR4的位18）——当设置时，此标志：(1) 指示（通过CPUID.01H:ECX.OSXSAVE[位27]）操作系统支持通用软件使用XGETBV、XSAVE和XRSTOR指令；(2) 启用XSAVE和XRSTOR指令以保存和恢复x87浮点处理单元状态（包括MMX寄存器）、SSE状态（XMM寄存器和MXCSR），以及在XCR0中启用的其他处理器扩展状态；(3) 启用处理器执行XGETBV和XSETBV指令以读写XCR0。参见第2.6节和第13章“指令集扩展和处理器扩展状态的系统编程”。SMEP SMEP启用位（CR4的位20）——设置时启管理员模式执行保护（SMEP）。参见第4.6节“访问权限”。TPL 任务优先级（CR8的位3:0）——这设置了对应于要阻塞的最高优先级中断的阈值。值为0表示所有中断均被启用。此字段在64位模式下可用。值为15表示所有中断将被禁用。

2.5.1 控制寄存器标志位的CPUID指令限定条件

并非所有控制寄存器CR4中的标志位都在所有处理器上实现。除PCE标志位外，其他标志位可通过CPUID指令进行限定检测，以确定在使用前它们是否在当前处理器上实现。

CR8寄存器在支持Intel 64架构的处理器上可用。

NOTE

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

OSXMMEXCPT

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

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

VMXE

**VMX-Enable Bit (bit 13 of CR4)** — Enables VMX operation when set. See Chapter 23, “Introduction to Virtual-Machine Extensions.”

SMXE

**SMX-Enable Bit (bit 14 of CR4)** — Enables SMX operation when set. See Chapter 5, “Safer Mode Extensions Reference” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2C*.

FSGSBASE

**FSGSBASE-Enable Bit (bit 16 of CR4)** — Enables the instructions RDFSBASE, RDGSBASE, WRFSBASE, and WRGSBASE.

PCIDE

**PCID-Enable Bit (bit 17 of CR4)** — Enables process-context identifiers (PCIDs) when set. See Section 4.10.1, “Process-Context Identifiers (PCIDs)”. Can be set only in IA-32e mode (if IA32\_EFER.LMA = 1).

OSXSAVE

**XSAVE and Processor Extended States-Enable Bit (bit 18 of CR4)** — When set, this flag: (1) indicates (via CPUID.01H:ECX.OSXSAVE[bit 27]) that the operating system supports the use of the XGETBV, XSAVE and XRSTOR instructions by general software; (2) enables the XSAVE and XRSTOR instructions to save and restore the x87 FPU state (including MMX registers), the SSE state (XMM registers and MXCSR), along with other processor extended states enabled in XCR0; (3) enables the processor to execute XGETBV and XSETBV instructions in order to read and write XCR0. See Section 2.6 and Chapter 13, “System Programming for Instruction Set Extensions and Processor Extended States”.

SMEP

**SMEP-Enable Bit (bit 20 of CR4)** — Enables supervisor-mode execution prevention (SMEP) when set. See Section 4.6, “Access Rights”.

TPL

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

2.5.1 CPUID Qualification of Control Register Flags

Not all flags in control register CR4 are implemented on all processors. With the exception of the PCE flag, they can be qualified with the CPUID instruction to determine if they are implemented on the processor before they are used.

The CR8 register is available on processors that support Intel 64 architecture.

## 2.6 扩展控制寄存器（包括XCR0寄存器）

如果CPUID.01H:ECX.XSAVE[位26] 为1，则处理器支持一个或多个扩展控制寄存器（XCR）。目前唯一定义的此类寄存器是XCR0。该寄存器指定操作系统在该处理器上启用的处理器状态集合，例如x87浮点处理单元状态、SSE状态、AVX状态以及Intel 64架构未来可能引入的其他处理器扩展状态。操作系统通过编程XCR0来反映其支持的功能。

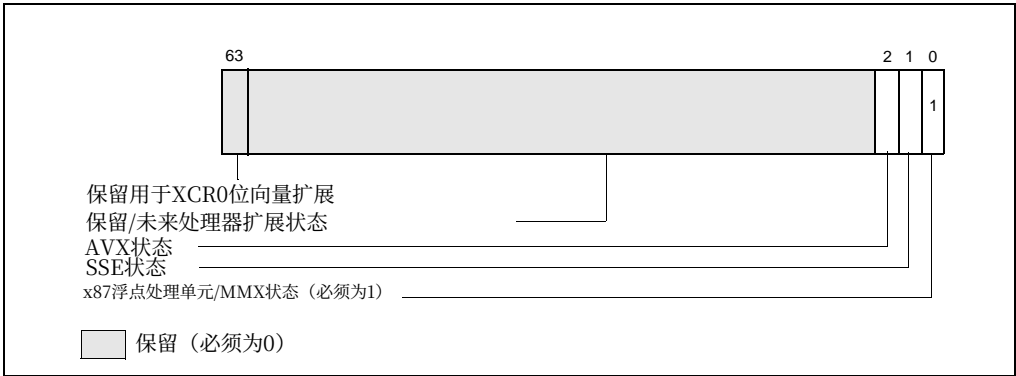


图2-8. XCR0

仅当CR4.OSXSAVE[位18] = 1时，软件才能访问XCR0寄存器。（该位也可通过CPUID.01H:ECX.OSXSAVE[位27]读取。）XCR0寄存器的布局经过架构设计，允许软件使用CPUID叶功能0DH来枚举处理器在XCR0寄存器中支持的位集合（参见《英特尔® 64和IA-32架构软件开发人员手册》第2A卷中的CPUID指令）。每个处理器状态（x87浮点处理单元状态、SSE状态、AVX状态或未来的处理器扩展状态）在XCR0寄存器中由一个位表示。操作系统可以根据CPUID叶功能0DH的结果，使用XSETBV指令指定适当的位掩码值，以向前兼容的方式启用未来的处理器扩展状态。

除第63位外，XCR0寄存器中的每个位都对应处理器状态的一个子集。因此XCR0为最多63组处理器状态扩展提供了空间。XCR0的第63位保留供未来扩展使用，不会表示处理器扩展状态。

当前，XCR0寄存器定义了三种处理器状态，最多61位保留用于未来处理器扩展状态：

- XCR0.X87（位0）：该位0必须为1。尝试向该位写入0将导致#GP异常。
- XCR0.SSE（位1）：若为1，则XSAVE、XSAVEOPT和XRSTOR可用于管理MXCSR寄存器和XMM寄存器（64位模式下为XMM0-XMM15；其他模式下为XMM0-XMM7）。
- XCR0.AVX（位2）：若为1，则可执行AVX指令，并且XSAVE、XSAVEOPT和XRSTOR可用于管理YMM寄存器的高半部分（在64位模式下为YMM0-YMM15；否则为YMM0-YMM7）。

任何尝试在XCR0寄存器中设置保留位（根据执行CPUID指令后EAX=0DH和ECX= 0H寄存器内容确定）的行为将导致#GP异常。尝试向XCR0.x87（位0）写入0将导致#GP异常。尝试向XCR0.SSE（位1）写入0并向XCR0.AVX（位2）写入1同样会导致#GP异常。

如果XCR0寄存器中的某位为1，软件可以使用XSAVE指令将相应的处理器状态保存到内存中（参见《英特尔® 64和IA-32架构软件开发人员手册》第2B卷中的XSAVE指令说明）。

重置后，XCR0寄存器中的所有位（除位0外）均被清零，XCR0[0] 被设置为1。

## 2.7 系统指令摘要

系统指令处理系统级功能，例如加载系统寄存器、管理缓存、设置调试寄存器。这些指令中有许多只能由操作

## 2.6 EXTENDED CONTROL REGISTERS (INCLUDING XCR0)

If CPUID.01H:ECX.XSAVE[bit 26] is 1, the processor supports one or more **extended control registers** (XCRs). Currently, the only such register defined is XCR0. This register specifies the set of processor states that the operating system enables on that processor, e.g. x87 FPU state, SSE state, AVX state, and other processor extended states that Intel 64 architecture may introduce in the future. The OS programs XCR0 to reflect the features it supports.

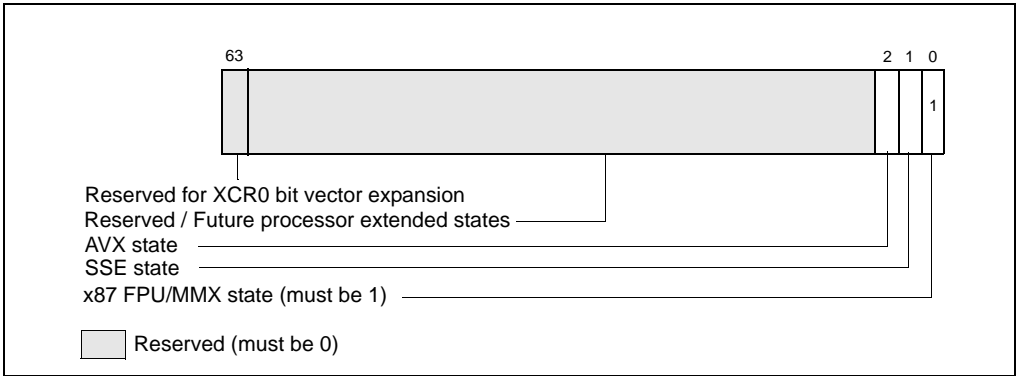


Figure 2-8. XCR0

Software can access XCR0 only if CR4.OSXSAVE[bit 18] = 1. (This bit is also readable as CPUID.01H:ECX.OSXSAVE[bit 27].) The layout of XCR0 is architected to allow software to use CPUID leaf function 0DH to enumerate the set of bits that the processor supports in XCR0 (see CPUID instruction in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). Each processor state (X87 FPU state, SSE state, AVX state, or a future processor extended state) is represented by a bit in XCR0. The OS can enable future processor extended states in a forward manner by specifying the appropriate bit mask value using the XSETBV instruction according to the results of the CPUID leaf 0DH.

With the exception of bit 63, each bit in XCR0 corresponds to a subset of the processor states. XCR0 thus provides space for up to 63 sets of processor state extensions. Bit 63 of XCR0 is reserved for future expansion and will not represent a processor extended state.

Currently, XCR0 has three processor states defined, with up to 61 bits reserved for future processor extended states:

- XCR0.X87 (bit 0): This bit 0 must be 1. An attempt to write 0 to this bit causes a #GP exception.
- XCR0.SSE (bit 1): If 1, XSAVE, XSAVEOPT, and XRSTOR can be used to manage MXCSR and XMM registers (XMM0-XMM15 in 64-bit mode; otherwise XMM0-XMM7).
- XCR0.AVX (bit 2): If 1, AVX instructions can be executed and XSAVE, XSAVEOPT, and XRSTOR can be used to manage the upper halves of the YMM registers (YMM0-YMM15 in 64-bit mode; otherwise YMM0-YMM7).

Any attempt to set a reserved bit (as determined by the contents of EAX and EDX after executing CPUID with EAX=0DH, ECX= 0H) in XCR0 for a given processor will result in a #GP exception. An attempt to write 0 to XCR0.x87 (bit 0) will result in a #GP exception. An attempt to write 0 to XCR0.SSE (bit 1) and 1 to XCR0.AVX (bit 2) also results in a #GP exception.

If a bit in XCR0 is 1, software can use the XSAVE instruction to save the corresponding processor state to memory (see XSAVE instruction in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*).

After reset, all bits (except bit 0) in XCR0 are cleared to zero, XCR0[0] is set to 1.

## 2.7 SYSTEM INSTRUCTION SUMMARY

System instructions handle system-level functions such as loading system registers, managing the cache, managing interrupts, or setting up the debug registers. Many of these instructions can be executed only by oper-



系统或执行过程（即运行在特权级别0的过程）执行。其他指令可以在任何特权级执行，因此可供应用程序使用。

表2-3列出了系统指令，并指明它们是否对应用程序可用且有用。这些指令在《英特尔® 64和IA-32架构软件开发人员手册》第2A、2B和2C卷中有详细描述。

表 2-3. 系统指令摘要

指令	描述	用途 应用程序?	受保护于 应用程序?
LLDT	加载LDT寄存器	No	Yes
SLDT	存储局部描述符表寄存器	No	No
LGDT	加载全局描述符表寄存器	No	Yes
SGDT	存储全局描述符表寄存器	No	No
LTR	加载任务寄存器	No	Yes
STR	存储任务寄存器	No	No
LIDT	加载IDT寄存器	No	Yes
SIDT	存储IDT寄存器	No	No
MOV CR <sub><i>n</i></sub>	加载和存储控制寄存器	No	Yes
SMSW	存储机器状态字	Yes	No
LMSW	加载机器状态字	No	Yes
CLTS	清除CR0中的任务切换标志	No	Yes
ARPL	调整请求特权级	是 <sup>1, 5</sup>	No
LAR	加载访问权限	Yes	No
LSL	加载段界限	Yes	No
VERR	验证可读性	Yes	No
VERW	验证可写性	Yes	No
MOV DR <sub><i>n</i></sub>	加载和存储调试寄存器	No	Yes
INVD	使缓存无效，无回写	No	Yes
WBINVD	使缓存无效，带回写	No	Yes
INVLPG	无效化TLB条目	No	Yes
HLT	暂停处理器	No	Yes
锁定前缀	总线锁定	Yes	No
RSM	系统管理模式返回	No	Yes
读取模型特定寄存器 <sup>3</sup>	读模型特定寄存器	No	Yes
WRMSR <sup>3</sup>	写模型特定寄存器	No	Yes
RDPMC <sup>4</sup>	读取性能监控计数器	Yes	Yes <sup>2</sup>
RDTSC <sup>3</sup>	读取时间戳计数器	Yes	Yes <sup>2</sup>
RDTSCP <sup>7</sup>	读取序列化时间戳计数器	Yes	Yes <sup>2</sup>
XGETBV	返回XCR0寄存器的状态	Yes	No
XSETBV	启用一个或多个处理器扩展状态	No <sup>6</sup>	Yes

ating-system or executive procedures (that is, procedures running at privilege level 0). Others can be executed at any privilege level and are thus available to application programs.

Table 2-3 lists the system instructions and indicates whether they are available and useful for application programs. These instructions are described in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B & 2C*.

Table 2-3. Summary of System Instructions

Instruction	Description	Useful to Application?	Protected from Application?
LLDT	Load LDT Register	No	Yes
SLDT	Store LDT Register	No	No
LGDT	Load GDT Register	No	Yes
SGDT	Store GDT Register	No	No
LTR	Load Task Register	No	Yes
STR	Store Task Register	No	No
LIDT	Load IDT Register	No	Yes
SIDT	Store IDT Register	No	No
MOV CR <sub><i>n</i></sub>	Load and store control registers	No	Yes
SMSW	Store MSW	Yes	No
LMSW	Load MSW	No	Yes
CLTS	Clear TS flag in CR0	No	Yes
ARPL	Adjust RPL	Yes <sup>1, 5</sup>	No
LAR	Load Access Rights	Yes	No
LSL	Load Segment Limit	Yes	No
VERR	Verify for Reading	Yes	No
VERW	Verify for Writing	Yes	No
MOV DR <sub><i>n</i></sub>	Load and store debug registers	No	Yes
INVD	Invalidate cache, no writeback	No	Yes
WBINVD	Invalidate cache, with writeback	No	Yes
INVLPG	Invalidate TLB entry	No	Yes
HLT	Halt Processor	No	Yes
LOCK (Prefix)	Bus Lock	Yes	No
RSM	Return from system management mode	No	Yes
RDMSR <sup>3</sup>	Read Model-Specific Registers	No	Yes
WRMSR <sup>3</sup>	Write Model-Specific Registers	No	Yes
RDPMC <sup>4</sup>	Read Performance-Monitoring Counter	Yes	Yes <sup>2</sup>
RDTSC <sup>3</sup>	Read Time-Stamp Counter	Yes	Yes <sup>2</sup>
RDTSCP <sup>7</sup>	Read Serialized Time-Stamp Counter	Yes	Yes <sup>2</sup>
XGETBV	Return the state of XCR0	Yes	No
XSETBV	Enable one or more processor extended states	No <sup>6</sup>	Yes



表2-3. 系统指令摘要（续）

指令	描述	用途 应用程序?	受保护于 应用程序?
----	----	-------------	---------------

**注释：**  
1. 对运行在CPL为1或2的应用程序有用。2. 控制寄存器CR4中的TSD和PCE标志位控制运行在CPL为3的应用程序对这些指令的访问。3. 这些指令是随奔腾处理器引入IA-32架构的。4. 该指令是随奔腾Pro处理器以及带有MMX技术的奔腾处理器引入IA-32架构的。5. 该指令在64位模式下不受支持。6. 应用程序使用XGETBV指令来查询启用了哪一组处理器扩展状态。7. RDTSCP指令是在英特尔酷睿i7处理器中引入的。

2.7.1 加载与存储系统寄存器

GDTR、LDTR、IDTR和TR寄存器各自具有加载和存储指令，用于将数据加载到寄存器以及从寄存器存储数据：

- LGDT（加载全局描述符表寄存器）——从内存中将GDT基地址和界限加载到GDTR寄存器中。
  - SGDT（存储全局描述符表寄存器）——将GDT基地址和界限从GDTR寄存器存储到内存中。
  - LIDT（加载IDT寄存器）——从内存中将IDT基地址和界限加载到IDTR寄存器中。
  - SIDT（加载IDTR寄存器）——将IDTR寄存器中的IDT基地址和界限存储到内存中。
  - LLDT（加载LDT寄存器）——将LDT段选择子和段描述符从内存加载到LDTR中。（段选择子操作数也可以位于通用寄存器中。）
  - SLDT（存储LDT寄存器）——将LDTR寄存器中的LDT段选择子存储到内存或通用寄存器中。
- 
- LTR（加载任务寄存器）——将TSS的段选择子和段描述符从内存加载到任务寄存器中。（段选择子操作数也可以位于通用寄存器中。）
  - STR（存储任务寄存器）——将当前任务TSS的段选择子从任务寄存器存储到内存或通用寄存器中。

LMSW（加载机器状态字）和SMSW（存储机器状态字）指令操作控制寄存器CR0的0到15位。这些指令是为了与16位Intel 286处理器兼容而提供的。为在32位IA-32处理器上运行而编写的程序不应使用这些指令，而应使用MOV指令访问控制寄存器CR0。

CLTS（清除CR0中的任务切换标志）指令用于处理设备不可用异常（#NM），该异常在处理器尝试执行浮点指令且TS标志被设置时发生。该指令允许在保存x87浮点处理单元上下文后清除TS标志，从而防止进一步的#NM异常。有关TS标志的更多信息，请参阅第2.5节“控制寄存器”。

控制寄存器（CR0、CR1、CR2、CR3、CR4和CR8）通过MOV指令加载。该指令从通用寄存器加载控制寄存器，或将控制寄存器的内容存储到通用寄存器中。

2.7.2 访问权限验证

处理器提供了若干指令用于检查段选择子和段描述符，以确定是否允许访问其关联的段。这些指令复制了处理器自动执行的部分访问权限和类型检查功能，从而使操作系统或执行软件能够避免产生异常。

ARPL（调整请求特权级）指令用于调整段选择子的请求特权级（RPL），以匹配提供该段选择子的程序或过程的特权级。详见第5.10.4节“检查调用者访问权限-

Table 2-3. Summary of System Instructions (Contd.)

Instruction	Description	Useful to Application?	Protected from Application?
-------------	-------------	---------------------------	--------------------------------

**NOTES:**  
1. Useful to application programs running at a CPL of 1 or 2.  
2. The TSD and PCE flags in control register CR4 control access to these instructions by application programs running at a CPL of 3.  
3. These instructions were introduced into the IA-32 Architecture with the Pentium processor.  
4. This instruction was introduced into the IA-32 Architecture with the Pentium Pro processor and the Pentium processor with MMX technology.  
5. This instruction is not supported in 64-bit mode.  
6. Application uses XGETBV to query which set of processor extended states are enabled.  
7. RDTSCP is introduced in Intel Core i7 processor.

2.7.1 Loading and Storing System Registers

The GDTR, LDTR, IDTR, and TR registers each have a load and store instruction for loading data into and storing data from the register:

- **LGDT (Load GDTR Register)** — Loads the GDT base address and limit from memory into the GDTR register.
- **SGDT (Store GDTR Register)** — Stores the GDT base address and limit from the GDTR register into memory.
- **LIDT (Load IDTR Register)** — Loads the IDT base address and limit from memory into the IDTR register.
- **SIDT (Load IDTR Register)** — Stores the IDT base address and limit from the IDTR register into memory.
- **LLDT (Load LDT Register)** — Loads the LDT segment selector and segment descriptor from memory into the LDTR. (The segment selector operand can also be located in a general-purpose register.)
- **SLDT (Store LDT Register)** — Stores the LDT segment selector from the LDTR register into memory or a general-purpose register.
- **LTR (Load Task Register)** — Loads segment selector and segment descriptor for a TSS from memory into the task register. (The segment selector operand can also be located in a general-purpose register.)
- **STR (Store Task Register)** — Stores the segment selector for the current task TSS from the task register into memory or a general-purpose register.

The LMSW (load machine status word) and SMSW (store machine status word) instructions operate on bits 0 through 15 of control register CR0. These instructions are provided for compatibility with the 16-bit Intel 286 processor. Programs written to run on 32-bit IA-32 processors should not use these instructions. Instead, they should access the control register CR0 using the MOV instruction.

The CLTS (clear TS flag in CR0) instruction is provided for use in handling a device-not-available exception (#NM) that occurs when the processor attempts to execute a floating-point instruction when the TS flag is set. This instruction allows the TS flag to be cleared after the x87 FPU context has been saved, preventing further #NM exceptions. See Section 2.5, “Control Registers,” for more information on the TS flag.

The control registers (CR0, CR1, CR2, CR3, CR4, and CR8) are loaded using the MOV instruction. The instruction loads a control register from a general-purpose register or stores the content of a control register in a general-purpose register.

2.7.2 Verifying of Access Privileges

The processor provides several instructions for examining segment selectors and segment descriptors to determine if access to their associated segments is allowed. These instructions duplicate some of the automatic access rights and type checking done by the processor, thus allowing operating-system or executive software to prevent exceptions from being generated.

The ARPL (adjust RPL) instruction adjusts the RPL (requestor privilege level) of a segment selector to match that of the program or procedure that supplied the segment selector. See Section 5.10.4, “Checking Caller Access Priv-

指令（ARPL指令），” 以获取关于该指令功能和用途的详细说明。请注意，ARPL在64位模式下不受支持。

LAR（加载访问权限）指令验证指定段的可访问性，并将该段的段描述符中的访问权限信息加载到通用寄存器中。随后，软件可以检查这些访问权限，以确定段类型是否与其预期用途兼容。有关此指令功能和用途的详细说明，请参阅第5.10.1节“检查访问权限（LAR指令）”。

LSL（加载段界限）指令验证指定段的可访问性，并将该段的段描述符中的段限长加载到通用寄存器中。随后，软件可以将段限长与段内偏移量进行比较，以确定该偏移量是否位于段内。有关此指令功能和用途的详细说明，请参阅第5.10.3节“检查指针偏移量是否在限长范围内（LSL指令）”。

VERR（验证可读性）和VERW（验证可写性）指令分别验证在给定CPL下所选段是否可读或可写。有关这些指令功能和用途的详细说明，请参阅第5.10.2节“检查读/写权限（VERR和VERW指令）”。

### 2.7.3 加载和存储调试寄存器

处理器内部的调试功能由8个调试寄存器（DR0-DR7）控制。MOV指令允许向这些寄存器加载设置数据或从中存储数据。

在支持Intel 64架构的处理器上，调试寄存器DR0-DR7为64位。在32位模式和兼容模式下，对调试寄存器的写入操作会将高32位填充为零，读取操作则返回低32位。在64位模式下，DR6-DR7的高32位为保留位且必须写入零值。若向高32位任意位写入1将引发通用保护异常(0)。

在64位模式下，MOV DRn指令可读取或写入调试寄存器的全部64位（操作数大小前缀将被忽略）。DR0-DR3的所有64位均可由软件写入。但需注意，MOV DRn指令不会检查写入DR0-DR3的地址是否在实现范围内。地址匹配功能仅支持处理器实现所生成的有效地址。

### 2.7.4 使缓存和TLB失效

处理器提供了若干条指令，用于显式地使其缓存和TLB条目失效。INVD（无回写使缓存无效）指令会使内部缓存中的所有数据和指令条目失效，并向外部缓存发送信号，指示它们也应被失效。

WBINVD（回写并使缓存无效）指令执行的功能与INVD指令相同，区别在于它在使缓存失效之前，会将其内部缓存中已修改的缓存行写回到内存。在执行逻辑处理器或处理器核心的本地缓存失效后，WBINVD会向缓存层次结构中更高层级的缓存（与执行失效操作的逻辑处理器或核心共享的缓存）发出信号，要求它们在指令执行时将处于已修改状态的任何数据写回，并使其内容失效。

请注意，非共享缓存可能不会被写回也不会失效。在下图2-9中，如果LP0或LP1上执行的代码要执行WBINVD指令，则LP0/LP1共享的L1和L2缓存将被写回并失效，共享的L3缓存也是如此。然而，未与LP0和LP1共享的L1和L2缓存将不会被写回也不会失效。

ileges (ARPL Instruction),” for a detailed explanation of the function and use of this instruction. Note that ARPL is not supported in 64-bit mode.

The LAR (load access rights) instruction verifies the accessibility of a specified segment and loads access rights information from the segment’s segment descriptor into a general-purpose register. Software can then examine the access rights to determine if the segment type is compatible with its intended use. See Section 5.10.1, “Checking Access Rights (LAR Instruction),” for a detailed explanation of the function and use of this instruction.

The LSL (load segment limit) instruction verifies the accessibility of a specified segment and loads the segment limit from the segment’s segment descriptor into a general-purpose register. Software can then compare the segment limit with an offset into the segment to determine whether the offset lies within the segment. See Section 5.10.3, “Checking That the Pointer Offset Is Within Limits (LSL Instruction),” for a detailed explanation of the function and use of this instruction.

The VERR (verify for reading) and VERW (verify for writing) instructions verify if a selected segment is readable or writable, respectively, at a given CPL. See Section 5.10.2, “Checking Read/Write Rights (VERR and VERW Instructions),” for a detailed explanation of the function and use of this instruction.

### 2.7.3 Loading and Storing Debug Registers

Internal debugging facilities in the processor are controlled by a set of 8 debug registers (DR0-DR7). The MOV instruction allows setup data to be loaded to and stored from these registers.

On processors that support Intel 64 architecture, debug registers DR0-DR7 are 64 bits. In 32-bit modes and compatibility mode, writes to a debug register fill the upper 32 bits with zeros. Reads return the lower 32 bits. In 64-bit mode, the upper 32 bits of DR6-DR7 are reserved and must be written with zeros. Writing one to any of the upper 32 bits causes an exception, #GP(0).

In 64-bit mode, MOV DRn instructions read or write all 64 bits of a debug register (operand-size prefixes are ignored). All 64 bits of DR0-DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0-DR3 are in the limits of the implementation. Address matching is supported only on valid addresses generated by the processor implementation.

### 2.7.4 Invalidating Caches and TLBs

The processor provides several instructions for use in explicitly invalidating its caches and TLB entries. The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and sends a signal to the external caches indicating that they should be also be invalidated.

The WBINVD (invalidate cache with writeback) instruction performs the same function as the INVD instruction, except that it writes back modified lines in its internal caches to memory before it invalidates the caches. After invalidating the caches local to the executing logical processor or processor core, WBINVD signals caches higher in the cache hierarchy (caches shared with the invalidating logical processor or core) to write back any data they have in modified state, at the time of instruction execution and to invalidate their contents.

Note, non-shared caches may not be written back nor invalidated. In Figure 2-9 below, if code executing on either LP0 or LP1 were to execute a WBINVD, the shared L1 and L2 for LP0/LP1 will be written back and invalidated as do the shared L3. However, the L1 and L2 caches not shared with LP0 and LP1 will not be written back nor invalidated.

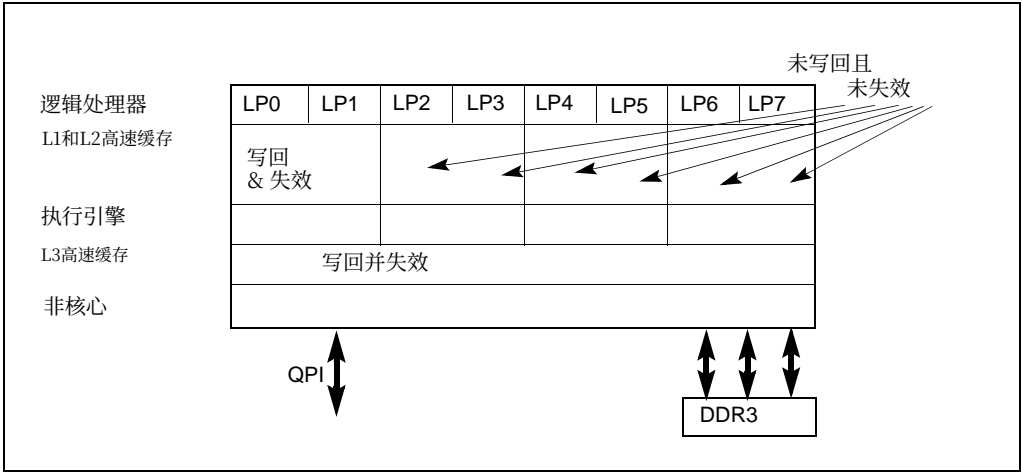


图2-9 WBINVD对共享与非共享缓存层次结构的失效操作

INVLPG（无效化TLB条目）指令会使指定页的TLB条目失效（刷新）。

2.7.5 控制处理器

HLT（暂停处理器）指令会停止处理器，直到接收到启用的中断（如通常启用的NMI或SMI）、调试异常、BINIT#信号、INIT#信号或RESET#信号。处理器会生成一个特殊的总线周期来指示已进入暂停模式。

硬件可通过多种方式响应此信号。前面板上的指示灯可能被点亮，可能生成用于记录诊断信息的NMI中断，也可能调用重置初始化（请注意BINIT#引脚是随奔腾Pro处理器引入的）。如果在关机期间有任何非唤醒事件处于待处理状态，这些事件将在处理完关机唤醒事件后得到处理（例如A20M#中断）。

LOCK前缀在修改内存操作数时会调用锁定（原子）的读-修改-写操作。如下所述，该机制用于实现多处理器系统中处理器之间的可靠通信：

- 在奔腾处理器及更早的IA-32处理器中，LOCK前缀会使处理器在执行指令期间发出LOCK#信号。这总是会导致显式的总线锁定发生。
- 在奔腾4、英特尔至强和P6系列处理器中，锁定操作通过缓存锁定或总线锁定来处理。如果内存访问可缓存且仅影响单个缓存行，则会调用缓存锁定，此时系统总线和系统内存中的实际内存位置在操作期间不会被锁定。在这种情况下，总线上其他奔腾4、英特尔至强或P6系列处理器会写回任何已修改的数据，并根据需要使其缓存失效以维持系统内存的一致性。如果内存访问不可缓存和/或跨越缓存行边界，则处理器的LOCK#信号会被置位，且处理器在锁定操作期间不响应总线控制请求。

RSM（从SMM返回）指令将处理器（从上下文转储中）恢复到进入系统管理模式（SMM）中断之前的状态。

2.7.6 读取性能监控与时间戳计数器

RDPMC（读取性能监控计数器）和RDTSC（读取时间戳计数器）指令允许应用程序分别读取处理器的性能监控计数器和时间戳计数器。基于英特尔NetBurst微架构的处理器拥有18个40位性能监控计数器；P6系列处理器则配备两个40位计数器。英特尔凌动处理器及大多数基于英特尔酷睿微架构的处理器支持两类性能监控计数器：两个可编程性能计数器（类似于P6系列中的配置），以及三个固定功能的性能监控计数器。

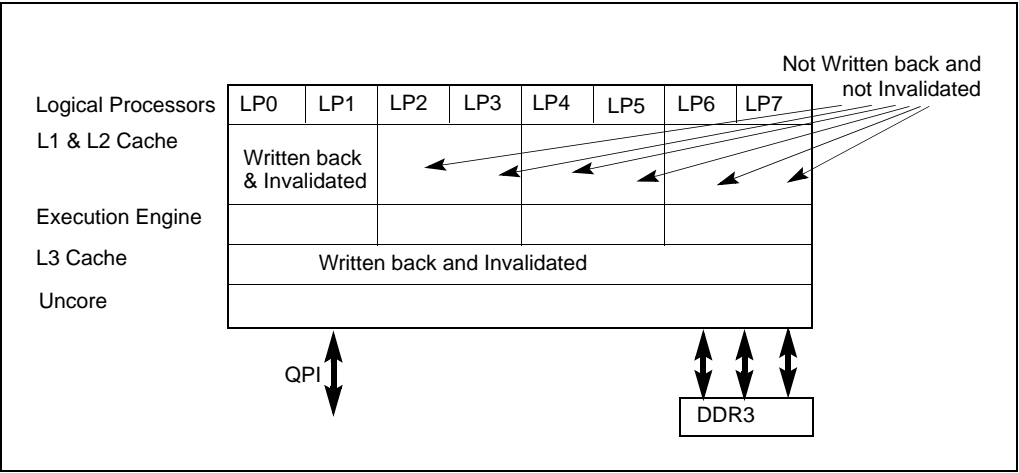


Figure 2-9. WBINVD Invalidation of Shared and Non-Shared Cache Hierarchy

The INVLPG (invalidate TLB entry) instruction invalidates (flushes) the TLB entry for a specified page.

2.7.5 Controlling the Processor

The HLT (halt processor) instruction stops the processor until an enabled interrupt (such as NMI or SMI, which are normally enabled), a debug exception, the BINIT# signal, the INIT# signal, or the RESET# signal is received. The processor generates a special bus cycle to indicate that the halt mode has been entered.

Hardware may respond to this signal in a number of ways. An indicator light on the front panel may be turned on. An NMI interrupt for recording diagnostic information may be generated. Reset initialization may be invoked (note that the BINIT# pin was introduced with the Pentium Pro processor). If any non-wake events are pending during shutdown, they will be handled after the wake event from shutdown is processed (for example, A20M# interrupts).

The LOCK prefix invokes a locked (atomic) read-modify-write operation when modifying a memory operand. This mechanism is used to allow reliable communications between processors in multiprocessor systems, as described below:

- In the Pentium processor and earlier IA-32 processors, the LOCK prefix causes the processor to assert the LOCK# signal during the instruction. This always causes an explicit bus lock to occur.
- In the Pentium 4, Intel Xeon, and P6 family processors, the locking operation is handled with either a cache lock or bus lock. If a memory access is cacheable and affects only a single cache line, a cache lock is invoked and the system bus and the actual memory location in system memory are not locked during the operation. Here, other Pentium 4, Intel Xeon, or P6 family processors on the bus write-back any modified data and invalidate their caches as necessary to maintain system memory coherency. If the memory access is not cacheable and/or it crosses a cache line boundary, the processor's LOCK# signal is asserted and the processor does not respond to requests for bus control during the locked operation.

The RSM (return from SMM) instruction restores the processor (from a context dump) to the state it was in prior to an system management mode (SMM) interrupt.

2.7.6 Reading Performance-Monitoring and Time-Stamp Counters

The RDPMC (read performance-monitoring counter) and RDTSC (read time-stamp counter) instructions allow application programs to read the processor's performance-monitoring and time-stamp counters, respectively. Processors based on Intel NetBurst® microarchitecture have eighteen 40-bit performance-monitoring counters; P6 family processors have two 40-bit counters. Intel® Atom™ processors and most of the processors based on the Intel Core microarchitecture support two types of performance monitoring counters: two programmable perfor-



性能计数器，类似于P6系列中可用的那些，以及三个固定功能的性能监控计数器。

可编程性能计数器可支持统计事件的发生次数或持续时间。可在可编程计数器上监控的事件通常是型号特定的（CPLD叶0AH枚举的架构性能事件除外）；它们可能包括已解码指令数量、接收的中断数量或缓存加载次数。可以设置单个计数器来监控不同事件。使用系统指令WRMSR在IA32\_PERFEVTSEL0/1（适用于英特尔Atom、英特尔酷睿2、英特尔酷睿双核和英特尔奔腾M处理器）、45个ESCR之一和18个CCCR MSR之一（适用于奔腾4和英特尔至强处理器）中设置值；或在PerfEvtSel0或PerfEvtSel1 MSR（适用于P6系列处理器）中设置值。RDPMC指令将选定计数器的当前计数值加载到EDX:EAX寄存器中。

固定功能性能性能计数器仅记录第19章“性能监控”中定义的特定事件。“监控事件”，以及任务切换 固定功能计数器的宽度/数量由CPLD叶0AH枚举。

时间戳计数器是一个与型号相关的64位计数器，每次处理器重置时都会被清零。若不进行重置，当处理器以3GHz时钟频率运行时，该计数器每年将递增~9.5 x 10^16次。在此时钟频率下，计数器需要超过190年才会回绕。RDTSC指令可将时间戳计数器的当前计数值加载到EDX:EAX寄存器中。

有关性能监控和时间戳计数器的更多信息，请参见第18.1节“性能监控概述”和第17.13节“时间戳计数器”。

RDTSC指令是随奔腾处理器引入IA-32架构的。RDPMC指令是随奔腾Pro处理器及带有MMX技术的奔腾处理器引入IA-32架构的。早期的奔腾处理器有两个性能监控计数器，但只能通过RDMSR指令在特权级别0下读取。

#### 2.7.6.1 在64位模式下读取计数器

在64位模式下，RDTSC的操作与保护模式相同。时间戳计数器中的计数值存储在EDX:EAX中（或RDX[31:0]:RAX[31:0]，同时RDX[63:32]:RAX[63:32] 被清零）。

RDPMC需要一个索引来指定性能监控计数器的偏移量。在针对奔腾4或英特尔至强处理器系列的64位模式下，索引在ECX[30:0]中指定。当前性能监控计数器的计数值存储在EDX:EAX中（或RDX[31:0]:RAX[31:0]，同时RDX[63:32]:RAX[63:32] 被清零）。

### 2.7.7 读取和写入模型特定寄存器

RDMSR（读取模型特定寄存器）和WRMSR（写入模型特定寄存器）指令分别允许对处理器的64位模型特定寄存器（MSR）进行读取和写入操作。要读取或写入的MSR由ECX寄存器中的值指定。

RDMSR将指定MSR的值读取到EDX:EAX寄存器；WRMSR将EDX:EAX寄存器中的值写入指定的MSR。RDMSR和WRMSR是随奔腾处理器引入IA-32架构的。

更多信息请参阅第9.4节“模型特定寄存器（MSR）”。

#### 2.7.7.1 在64位模式下读取和写入模型特定寄存器

RDMSR和WRMSR需要一个索引来指定MSR的地址。在64位模式下，索引为32位；通过ECX寄存器指定。

### 2.7.8 启用处理器扩展状态

需要XSETBV指令来启用操作系统对XCR0中各个处理器扩展状态的支持（参见第2.6节）。

mance counters similar to those available in the P6 family, and three fixed-function performance monitoring counters.

The programmable performance counters can support counting either the occurrence or duration of events. Events that can be monitored on programmable counters generally are model specific (except for architectural performance events enumerated by CPLD leaf 0AH); they may include the number of instructions decoded, interrupts received, or the number of cache loads. Individual counters can be set up to monitor different events. Use the system instruction WRMSR to set up values in IA32\_PERFEVTSEL0/1 (for Intel Atom, Intel Core 2, Intel Core Duo, and Intel Pentium M processors), in one of the 45 ESCRs and one of the 18 CCCR MSRs (for Pentium 4 and Intel Xeon processors); or in the PerfEvtSel0 or the PerfEvtSel1 MSR (for the P6 family processors). The RDPMC instruction loads the current count from the selected counter into the EDX:EAX registers.

Fixed-function performance counters record only specific events that are defined in Chapter 19, “Performance Monitoring Events”, and the width/number of fixed-function counters are enumerated by CPLD leaf 0AH.

The time-stamp counter is a model-specific 64-bit counter that is reset to zero each time the processor is reset. If not reset, the counter will increment ~9.5 x 10<sup>16</sup> times per year when the processor is operating at a clock rate of 3GHz. At this clock frequency, it would take over 190 years for the counter to wrap around. The RDTSC instruction loads the current count of the time-stamp counter into the EDX:EAX registers.

See Section 18.1, “Performance Monitoring Overview,” and Section 17.13, “Time-Stamp Counter,” for more information about the performance monitoring and time-stamp counters.

The RDTSC instruction was introduced into the IA-32 architecture with the Pentium processor. The RDPMC instruction was introduced into the IA-32 architecture with the Pentium Pro processor and the Pentium processor with MMX technology. Earlier Pentium processors have two performance-monitoring counters, but they can be read only with the RDMSR instruction, and only at privilege level 0.

#### 2.7.6.1 Reading Counters in 64-Bit Mode

In 64-bit mode, RDTSC operates the same as in protected mode. The count in the time-stamp counter is stored in EDX:EAX (or RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared).

RDPMC requires an index to specify the offset of the performance-monitoring counter. In 64-bit mode for Pentium 4 or Intel Xeon processor families, the index is specified in ECX[30:0]. The current count of the performance-monitoring counter is stored in EDX:EAX (or RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared).

### 2.7.7 Reading and Writing Model-Specific Registers

The RDMSR (read model-specific register) and WRMSR (write model-specific register) instructions allow a processor’s 64-bit model-specific registers (MSRs) to be read and written, respectively. The MSR to be read or written is specified by the value in the ECX register.

RDMSR reads the value from the specified MSR to the EDX:EAX registers; WRMSR writes the value in the EDX:EAX registers to the specified MSR. RDMSR and WRMSR were introduced into the IA-32 architecture with the Pentium processor.

See Section 9.4, “Model-Specific Registers (MSRs),” for more information.

#### 2.7.7.1 Reading and Writing Model-Specific Registers in 64-Bit Mode

RDMSR and WRMSR require an index to specify the address of an MSR. In 64-bit mode, the index is 32 bits; it is specified using ECX.

### 2.7.8 Enabling Processor Extended States

The XSETBV instruction is required to enable OS support of individual processor extended states in XCR0 (see Section 2.6).