中断和异常处理

3.1 中断和异常处理概述

中断和异常是指示系统、处理器或当前正在执行的程序或任务中存在需要处理器注意的情况的事件。它们通常导致强制将执行从当前正在运行的程序或任务转移到称为中断处理程序或异常处理程序的特殊软件例程或任务。

• 中断在程序执行期间随机发生，以响应来自硬件的信号。系统硬件使用中断来处理处理器外部的事件，例如服务外围设备的请求。软件还可以通过执行 INT n 指令来生成中断。

• 当处理器在执行指令时检测到错误情况（例如除以零）时，就会发生异常。处理器检测各种错误情况，包括保护违规、页面错误和内部机器故障。Pentium 4、Intel Xeon、P6 系列和 Pentium 处理器的机器检查架构还允许在检测到内部硬件错误和总线错误时生成机器检查异常。

当收到中断或检测到异常时，当前正在运行的过程或任务将暂停，同时处理器执行中断或异常处理程序。处理程序执行完成后，处理器恢复执行被中断的过程或任务。除非无法从异常中恢复或中断导致当前正在运行的程序终止，否则中断的过程或任务的恢复不会影响程序的连续性。

实模式和保护模式下，中断向量表并不完全相同。

（1）在实模式下：

• 中断向量表位于内存的最低端 1KB 空间内，总共包含 256 个中断向量，每个中断向量占据 4 字节空间；

•每个中断向量的表项包含一个 2 字节的段地址和一个 2 字节的偏移量，这两部分共同构成了中断处理程序的入口地址；

• 中断向量表的位置是固定的，不能随意更改。

（2）在保护模式下：

• 中断向量表可以在内存中自由浮动，位置和大小不再受内存低端 1KB 空间的限制。

• 中断向量表项除了包含中断处理程序的目标地址（由 16 位选择子和 32 位偏移构成）外，还加入了 DPL（Descriptor Privilege Level，描述符特权级）域和其他一些控制字段。这些额外的字段使得保护模式下的中断处理过程更加灵活和强大。

• 操作系统或程序可以在运行时动态地修改中断向量表，以适应不同的需求。

• 中断的处理过程因 IDT（Interrupt Descriptor Table，中断描述符表）中的描述符而异，可以呈现出多样化的特点，包括特权级检查、栈切换等复杂的操作过程。

3.2 有关中断和异常了解性的内容

3.2.1 中断和异常向量

为了帮助处理异常和中断，每个架构定义的异常和每个需要处理器进行特殊处理的中断条件都分配有一个唯一的标识号，称为向量号。处理器使用分配给异常或中断的向量号作为 IDT 的索引，该表提供异常或中断处理程序的入口点。

向量号的允许范围是 0 到 255。0 到 31 范围内的向量号由 Intel 64 和 IA-32 架构保留，用于架构定义的异常和中断，但并非此范围内的所有向量号都具有当前定义的功能，此范围内未分配的向量号是保留的；32 到 255 范围内的向量号被指定为用户定义的中断，并且不由 Intel 64 和 IA-32 架构保留，这些中断通常分配给外部 I/O 设备，以使这些设备能够通过其中一种外部硬件中断机制向处理器发送中断。

3.2.2 中断源和异常源

1.中断源：处理器从两个来源接收中断。

（1）外部（硬件生成的）中断

外部中断通过处理器上的引脚或本地 APIC 接收。Pentium 4、Intel Xeon、P6 系列和 Pentium 处理器上的主要中断引脚是 LINT[1:0] 引脚，它们连接到本地 APIC。启用本地 APIC 后，可以通过 APIC 的本地向量表 (LVT) 对 LINT[1:0] 引脚进行编程，使其与处理器的任何异常或中断向量相关联。

当本地 APIC 全局/硬件禁用时，这些引脚分别配置为 INTR 和 NMI 引脚。断言 INTR 引脚会向处理器发出信号，表示已发生外部中断，处理器从系统总线读取外部中断控制器（例如 8259A）提供的中断向量号；断言 NMI 引脚会发出不可屏蔽中断 (NMI) 信号，该中断被分配给中断向量 2。

处理器的本地 APIC 通常连接到基于系统的 I/O APIC。在这里，在 I/O APIC 引脚处接收到的外部中断可以通过系统总线（Pentium 4、Intel Core 双核、Intel Core 2、Intel ® Atom™ 和 Intel Xeon 处理器）或 APIC 串行总线（P6 系列和 Pentium 处理器）定向到本地 APIC。I/O APIC 确定中断的向量编号并将该编号发送到本地 APIC。当系统包含多个处理器时，处理器还可以通过系统总线（Pentium 4、Intel Core 双核、Intel Core 2、Intel Atom 和 Intel Xeon 处理器）或 APIC 串行总线（P6 系列和 Pentium 处理器）相互发送中断。

LINT[1:0] 引脚不适用于不包含片上本地 APIC 的 Intel486 处理器和更早的奔腾处理器。这些处理器具有专用的 NMI 和 INTR 引脚。对于这些处理器，外部中断通常由基于系统的中断控制器 (8259A) 生成，中断通过 INTR 引脚发出信号。

（2）软件生成的中断

INT n 指令允许通过提供中断向量号作为操作数从软件内部生成中断。例如，INT 35 指令强制隐式调用中断 35 的中断处理程序。

0 到 255 中的任何中断向量都可以用作此指令的参数。但如果使用处理器的预定义 NMI 向量，则处理器的响应将与以正常方式生成的 NMI 中断不同。如果在此指令中使用向量号 2（NMI 向量），则会调用 NMI 中断处理程序，但不会激活处理器的 NMI 处理硬件。

使用 INT n 指令在软件中生成的中断不能被 EFLAGS 寄存器中的 IF 标志屏蔽。

2.异常源：处理器从三个来源接收异常。

（1）处理器检测到的程序错误异常

当处理器在应用程序或操作系统或执行程序的执行过程中检测到程序错误时，它会生成一个或多个异常（分为故障、陷阱和中止）。Intel 64 和 IA-32 体系结构为每个处理器可检测到的异常定义了一个向量号。

（2）软件生成的异常

INTO、INT 3 和 BOUND 指令允许在软件中生成异常。这些指令允许在指令流中的点执行异常条件检查。例如，INT 3 会导致生成断点异常。

INT n 指令可用于在软件中模拟异常，但有一个限制：如果 INT n 为架构定义的异常之一提供了向量，则处理器会生成对正确向量的中断（以访问异常处理程序），但不会在堆栈上推送错误代码。即使相关的硬件生成的异常通常会产生错误代码，情况也是如此，异常处理程序在处理异常时仍会尝试从堆栈中弹出错误代码。由于没有推送错误代码，处理程序将弹出并丢弃 EIP（代替丢失的错误代码），这会将返回发送到错误的位置。

（3）机器检查异常

P6 系列和 Pentium 处理器提供内部和外部机器检查机制，用于检查内部芯片硬件和总线事务的运行情况。当检测到机器检查错误时，处理器会发出机器检查异常信号（向量 18）并返回错误代码。

3.2.3 异常的分类：故障、陷阱和中止

根据异常的报告方式以及引起异常的指令是否可以在不丢失程序或任务连续性的情况

下重新启动，异常可分为故障、陷阱或中止。

• 故障 — 故障是一种通常可以纠正的异常，一旦纠正，程序就可以重新启动而不会失去连续性。报告故障时，处理器会将机器状态恢复到开始执行故障指令之前的状态。故障处理程序的返回地址（CS 和 EIP 寄存器的保存内容）指向故障指令，而不是故障指令之后的指令。

• 陷阱 — 陷阱是在执行捕获指令后立即报告的异常。陷阱允许继续执行程序或任务而不会失去程序连续性。陷阱处理程序的返回地址指向在捕获指令之后要执行的指令。

• 中止 — 中止是一种异常，它并不总是报告导致异常的指令的精确位置，并且不允许重新启动导致异常的程序或任务。中止用于报告严重错误，例如硬件错误和系统表中不一致或非法的值。

3.2.4 程序或任务的重新执行

为了允许在处理异常或中断后重新启动程序或任务，所有异常（中止除外）都应保证在指令边界上报告异常。所有中断都应保证在指令边界上执行。

• 对于故障类异常，返回指令指针（在处理器生成异常时保存）指向故障指令。因此，当在处理故障后重新启动程序或任务时，故障指令将重新启动（重新执行）。重新启动故障指令通常用于处理在阻止访问操作数时生成的异常。这种类型故障最常见的例子是页面错误异常（#PF），当程序或任务引用位于不在内存中的页面上的操作数时发生这种情况。当发生页面错误异常时，异常处理程序可以将页面加载到内存中并通过重新启动故障指令来恢复程序或任务的执行。为了确保重新启动对当前正在执行的程序或任务透明，处理器会保存必要的寄存器和堆栈指针，以允许重新启动到执行错误指令之前的状态。

• 对于陷阱类异常，返回指令指针指向陷阱指令之后的指令。如果在转移执行的指令期间检测到陷阱，则返回指令指针会反映转移。例如，如果在执行 JMP 指令时检测到陷阱，则返回指令指针指向 JMP 指令的目标，而不是 JMP 指令之后的下一个地址。所有陷阱异常都允许程序或任务重新启动而不会失去连续性。例如，溢出异常就是陷阱异常。在这里，返回指令指针指向测试 EFLAGS.OF（溢出）标志的 INTO 指令之后的指令。此异常的陷阱处理程序解决溢出情况。从陷阱处理程序返回后，程序或任务继续执行 INTO 指令之后的指令。

• 中止类异常不支持可靠地重新启动程序或任务。中止处理程序旨在收集发生中止异常时处理器状态的诊断信息，然后尽可能正常地关闭应用程序和系统。

中断严格支持重新启动中断的程序和任务，而不会失去连续性。为中断保存的返回指令指针指向处理器中断的指令边界处要执行的下一条指令。如果刚刚执行的指令有重复前缀，则在当前迭代结束时执行中断，并将寄存器设置为执行下一个迭代。

3.2.5 开启和禁止中断

处理器会抑制某些中断的产生，这取决于处理器的状态以及 EFLAGS 寄存器中的 IF 和 RF 标志。

1. 屏蔽可屏蔽硬件中断

IF 标志可以禁用对处理器 INTR 引脚或通过本地 APIC 接收的可屏蔽硬件中断的服务。当 IF 标志清除时，处理器会禁止传递到 INTR 引脚或通过本地 APIC 的中断生成内部中断请求；当 IF 标志设置时，传递到 INTR 或通过本地 APIC 引脚的中断将作为普通外部中断进行处理。

IF 标志不会影响传递到 NMI 引脚的非可屏蔽中断 (NMI) 或通过本地 APIC 传递的传递模式 NMI 消息，也不会影响处理器生成的异常。与 EFLAGS 寄存器中的其他标志一样，处理器会响应硬件重置清除 IF 标志。

可屏蔽硬件中断组包括保留的中断和异常向量 0 到 32，这一事实可能会引起混淆。从

架构上讲，当设置 IF 标志时，可以通过 INTR 引脚将 0 到 32 中的任何向量的中断传送到处理器，并且可以通过本地 APIC 传送 16 到 32 中的任何向量。然后，处理器将生成中断并调用向量号指向的中断或异常处理程序。因此，例如，可以通过 INTR 引脚（通过向量 14）调用页面错误处理程序；但是，这不是真正的页面错误异常。它是一个中断。与 INT n 指令一样，当通过 INTR 引脚向异常向量生成中断时，处理器不会将错误代码推送到堆栈上，因此异常处理程序可能无法正常运行。

IF 标志可以分别使用 STI（设置中断启用标志）和 CLI（清除中断启用标志）指令来设置或清除。只有当 CPL 等于或小于 IOPL 时，才可以执行这些指令。如果在 CPL 大于 IOPL 时执行这些指令，则会产生一般保护异常 (#GP)。（当通过在控制寄存器 CR4 中设置 VME 标志来启用虚拟模式扩展时，IOPL 对这些指令的影响会略有改变，行为也会受到 PVI 标志的影响。

IF 标志还受以下操作的影响：

• PUSHF 指令将所有标志存储在堆栈上，可以在堆栈中检查和修改它们；POPF 指令可用于将修改后的标志重新加载到 EFLAGS 寄存器中。

• 任务切换以及 POPF 和 IRET 指令加载 EFLAGS 寄存器；。

• 当通过中断门处理中断时，IF 标志会自动清除，从而禁用可屏蔽的硬件中断（如果通过陷阱门处理中断，则不会清除 IF 标志）。

2. 屏蔽指令断点

EFLAGS 寄存器中的 RF（恢复）标志控制处理器对指令断点条件的响应。

设置时，它会阻止指令断点生成调试异常（#DB）；清除时，指令断点将生成调试异常。RF 标志的主要功能是防止处理器在指令断点上进入调试异常循环。

3. 切换堆栈时屏蔽异常和中断

要切换到不同的堆栈段，软件通常使用一对指令，例如： MOV SS，AX MOV ESP，StackTop 如果在段选择器加载到 SS 寄存器之后但在 ESP 寄存器加载之前发生中断或异常，则进入堆栈空间的逻辑地址的这两个部分在中断或异常处理程序的持续时间内不一致。

为了防止这种情况，处理器会在 MOV 到 SS 指令或 POP 到 SS 指令之后抑制中断、调试异常和单步陷阱异常，直到到达下一条指令之后的指令边界（但仍可能会发生其他故障）。如果使用 LSS 指令修改 SS 寄存器的内容（推荐方法），则不会发生此问题。

3.2.6 异常和中断的优先级

如果在指令边界处有多个异常或中断待处理，处理器将按照可预测的顺序处理它们。表 6-2 显示了异常和中断源类别之间的优先级。

Table 6-2. Priority Among Simultaneous Exceptions and Interrupts

Priority	Description
1 (Highest)	Hardware Reset and Machine Checks - RESET - Machine Check
2	Trap on Task Switch - T flag in TSS is set
3	External Hardware Interventions - FLUSH - STOPCLK - SMI - INIT
4	Traps on the Previous Instruction - Breakpoints - Debug Trap Exceptions (TF flag set or data/I-O breakpoint)
5	Nonmaskable Interrupts (NMI) [1]
6	Maskable Hardware Interrupts [1]
7	Code Breakpoint Fault
8	Faults from Fetching Next Instruction - Code-Segment Limit Violation - Code Page Fault
9	Faults from Decoding the Next Instruction - Instruction length > 15 bytes - Invalid Opcode - Coprocessor Not Available
10 (Lowest)	Faults on Executing an Instruction - Overflow - Bound error - Invalid TSS - Segment Not Present - Stack fault - General Protection - Data Page Fault - Alignment Check - x87 FPU Floating-point exception - SIMD floating-point exception - Virtualization exception

3.3 中断描述符表

中断描述符表 (IDT) 将每个异常或中断向量与用于处理相关异常或中断的过程或任务的门描述符相关联。与 GDT 和 LDT 一样，IDT 是一个 8 字节描述符数组（在受保护模式下）；与 GDT 不同，IDT 的第一个条目可能包含一个描述符。为了形成 IDT 的索引，处理器将异常或中断向量按 8 缩放（门描述符中的字节数）。由于只有 256 个中断或异常向量，因此 IDT 不需要包含超过 256 个描述符；也可以包含少于 256 个描述符，因为描述符仅对可能发生的中断和异常向量是必需的。IDT 中所有空的描述符槽都应将描述符的当前标志设置为 0。

IDT 的基址应在 8 字节边界上对齐，以最大限度地提高缓存行填充的性能。限制值以字节表示，并添加到基地址以获取最后一个有效字节的地址，限制值为 0 时，有效字节数恰好为 1。由于 IDT 条目的长度始终为 8 个字节，因此限制值应始终小于 8 的整数倍（即 8N – 1）。

IDT 可以位于线性地址空间中的任何位置。如图 6-1 所示，处理器使用 IDTR 寄存器定位 IDT，该寄存器保存 IDT 的 32 位基地址和 16 位限制。

LIDT（加载 IDT 寄存器）和 SIDT（存储 IDT 寄存器）指令分别加载和存储 IDTR 寄存器的内容。LIDT 指令使用保存在内存操作数中的基地址和限制加载 IDTR 寄存器，仅当 CPL 为 0 时才能执行此指令，通常由操作系统的初始化代码在创建 IDT 时使用，操作系统也可以使用它从一个 IDT 更改为另一个；SIDT 指令将存储在 IDTR 中的基数和限制值复制到内存中，此指令可以在任何特权级别执行。

如果向量引用了超出 IDT 限制的描述符，则会生成一般保护异常 (#GP)。

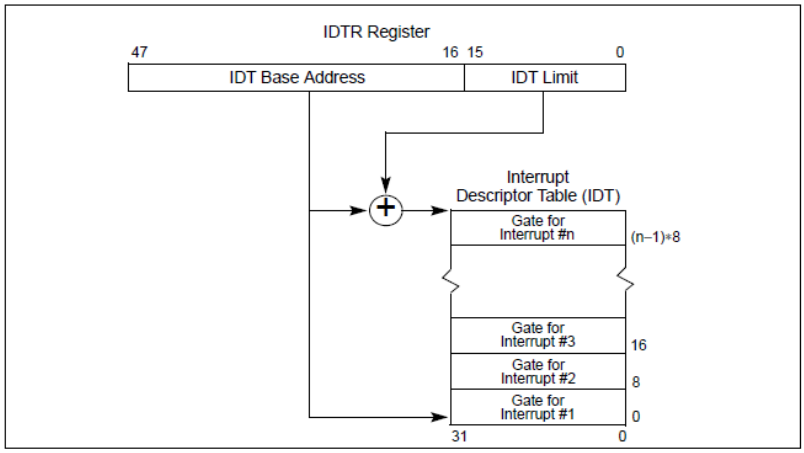

Figure 6-1. Relationship of the IDTR and IDT

3.4 IDT 描述符

IDT 可能包含任务门描述符、中断门描述符、陷阱门描述符中的任意一种。

图 6-2 展示了任务门、中断门和陷阱门描述符的格式。IDT 中使用的任务门的格式与 GDT 或 LDT 中使用的任务门的格式相同。任务门包含异常和/或中断处理程序任务的 TSS 的段选择器。

中断门和陷阱门与调用门非常相似。它们包含一个远指针（段选择器和偏移量），处理器使用该指针将程序执行转移到异常或中断处理程序代码段中的处理程序过程。这些门在处理器处理 EFLAGS 寄存器中的 IF 标志的方式上有所不同。

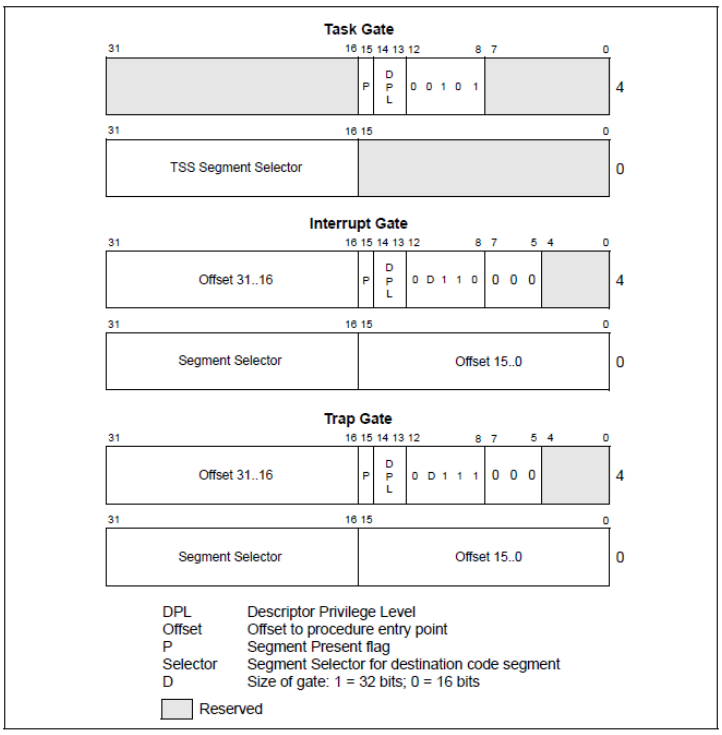

Figure 6-2. IDT Gate Descriptors

3.5 中断与异常处理

处理器处理对异常和中断处理程序的调用的方式类似于使用 CALL 指令处理对过程或任务的调用。在响应异常或中断时，处理器使用异常或中断向量作为 IDT 中描述符的索引。如果索引指向中断门或陷阱门，则处理器以类似于对调用门的 CALL 的方式调用异常或中断处理程序。

3.5.1 异常或中断处理程序

中断门或陷阱门引用在当前执行任务的上下文中运行的异常或中断处理程序（见图 6-3）。门的段选择器指向 GDT 或当前 LDT 中可执行代码段的段描述符。门描述符的偏移字段指向异常或中断处理程序的开头。

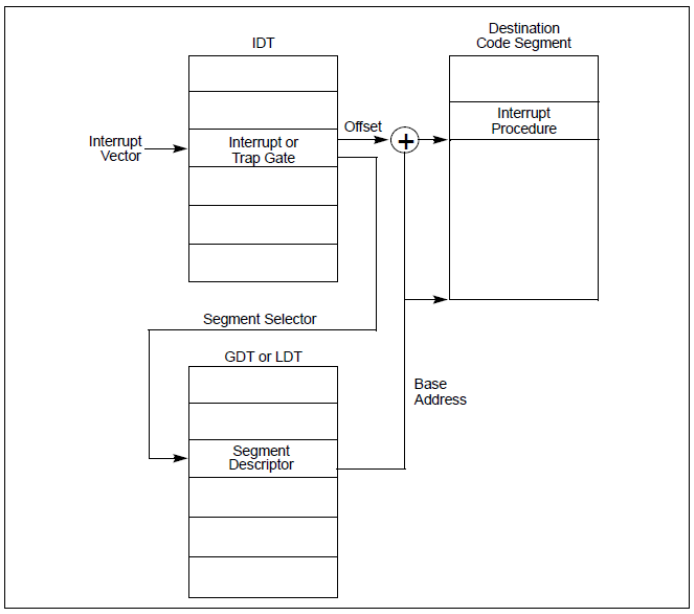

Figure 6-3. Interrupt Procedure Call

当处理器执行对异常或中断处理程序的调用时：

• 如果处理程序程序将在数值较低的特权级别上执行，则会发生堆栈切换：

（1）从当前执行任务的 TSS 获取处理程序要使用的堆栈的段选择器和堆栈指针。在这个新堆栈上，处理器推送被中断程序的堆栈段选择器和堆栈指针。

（2）处理器在新堆栈上保存 EFLAGS、CS 和 EIP 寄存器的当前状态（参见图 6-4）。

（3）如果异常导致保存错误代码，则将其推送到新堆栈上的 EIP 值之后。

• 如果处理程序程序将在与被中断程序相同的特权级别上执行：

（1）处理器在当前堆栈上保存 EFLAGS、CS 和 EIP 寄存器的当前状态（参见图 6-4）。

（2）如果异常导致错误代码被保存，则将其推送到当前堆栈上的 EIP 值之后。

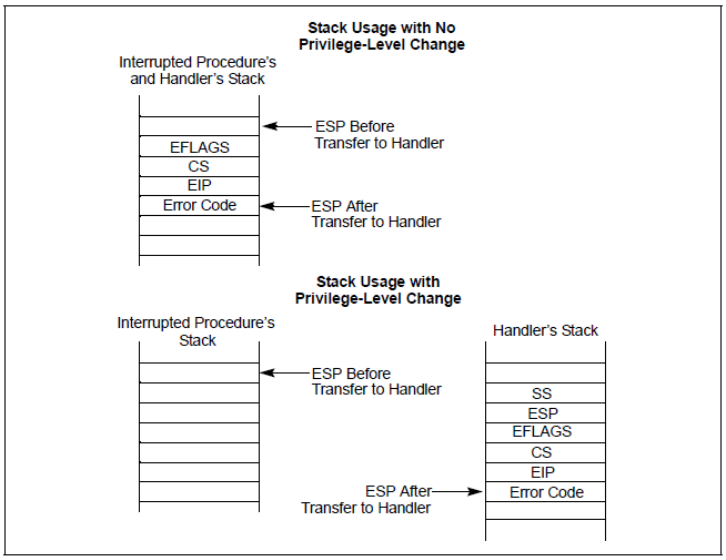

Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

要从异常或中断处理程序返回，处理程序必须使用 IRET（或 IRETD）指令。IRET 指令与 RET 指令类似，不同之处在于它将保存的标志恢复到 EFLAGS 寄存器中。仅当 CPL 为 0 时，才会恢复 EFLAGS 寄存器的 IOPL 字段；仅当 CPL 小于或等于 IOPL 时，才会更改 IF 标志。

如果在调用处理程序时发生堆栈切换，则 IRET 指令在返回时切换回中断程序的堆栈。

3.5.2 异常和中断处理程序的保护

异常和中断处理程序的特权级保护类似于通过调用门调用普通程序调用时的特权级保护。处理器不允许将执行转移到特权级低于 CPL 的代码段（特权级数值更高）中的异常或中断处理程序。

试图违反此规则会导致一般保护异常（#GP）。异常和中断处理程序的保护机制在以下方面有所不同：由于中断和异常向量没有 RPL，因此在隐式调用异常和中断处理程序时不会检查 RPL。仅当使用 INT n、INT 3 或 INTO 指令生成异常或中断时，处理器才会检查中断或陷阱门的 DPL，此处 CPL 必须小于或等于门的 DPL。此限制可防止在特权级别 3 上运行的应用程序或程序使用软件中断来访问关键异常处理程序（例如页面错误处理程序），前提是这些处理程序位于特权更高的代码段中（特权级别数值较低）。对于硬件生成的中断和处理器检测到的异常，处理器将忽略中断和陷阱门的 DPL。

由于异常和中断通常不会在可预测的时间发生，因此这些特权规则实际上对异常和中断处理程序可以运行的特权级别施加了限制。可以使用以下任一技术来避免特权级别违规。

•异常或中断处理程序可以放置在符合要求的代码段中。此技术可用于只需要访问堆栈上可用数据的处理程序（如除法错误异常）。如果处理程序需要数据段中的数据，则数据段需要从特权级别 3 访问，这将使其不受保护。

• 处理程序可以放置在特权级别为 0 的非符合要求的代码段中。无论中断的程序或任务在哪个 CPL 下运行，此处理程序都会始终运行。

3.5.3 异常或中断处理程序的标志使用情况

当通过中断门或陷阱门访问异常或中断处理程序时，处理器在将 EFLAGS 寄存器的内容保存到堆栈上之后，会清除 EFLAGS 寄存器中的 TF 标志。清除 TF 标志可防止指令跟踪影响中断响应，后续 IRET 指令将 TF（以及 VM、RF 和 NT）标志恢复为堆栈上 EFLAGS 寄存器的保存内容中的值。

中断门和陷阱门之间的唯一区别在于处理器处理 EFLAGS 寄存器中的 IF 标志的方式。当通过中断门访问异常或中断处理程序时，处理器会清除 IF 标志以防止其他中断干扰当前中断处理程序，后续的 IRET 指令将 IF 标志恢复为堆栈上 EFLAGS 寄存器保存内容中的值。通过陷阱门访问处理程序不会影响 IF 标志。

3.5.4 中断任务

当通过 IDT 中的任务门访问异常或中断处理程序时，将导致任务切换。使用单独的任务处理异常或中断具有以下几个优点：

• 被中断程序或任务的整个上下文会自动保存。

• 新的 TSS 允许处理程序在处理异常或中断时使用新的特权级别 0 堆栈。如果在当前特权级别 0 堆栈损坏时发生异常或中断，则通过任务门访问处理程序可以为处理程序提供新的特权级别 0 堆栈，从而防止系统崩溃。

• 通过为处理程序提供单独的地址空间，可以进一步将其与其他任务隔离。这可以通过为其提供单独的 LDT 来实现。

使用单独的任务处理中断的缺点：在任务切换时必须保存的机器状态量使其比使用中断门慢，从而导致中断延迟增加。

IDT 中的任务门引用 GDT 中的 TSS 描述符（参见图 6-5）。切换到处理程序任务的方

式与普通任务切换相同。返回到中断任务的链接存储在处理程序任务的 TSS 的上一个任务链接字段中。如果异常导致生成错误代码，则此错误代码将复制到新任务的堆栈中。

当在操作系统中使用异常或中断处理程序任务时，实际上有两种机制可用于调度任务：软件调度程序（操作系统的一部分）和硬件调度程序（处理器中断机制的一部分）。软件调度程序需要适应在启用中断时可能调度的中断任务。

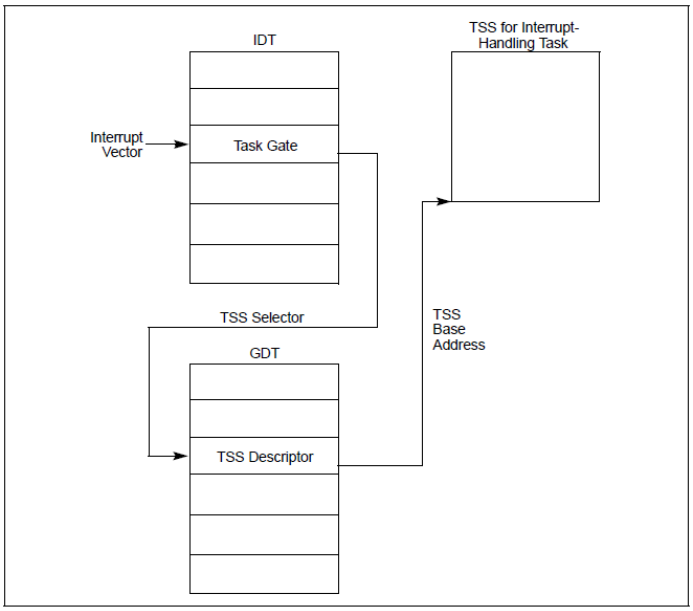

Figure 6-5. Interrupt Task Switch