3. 中断和异常处理

本报告已设置目录，使用 github 作为图床进行图片的挂载，如果以 markdown 格式阅读，还请麻烦打开代理软件。

3.1. 中断和异常处理概述

3.1.1 什么是中断和异常?

- 中断（Interrupts）：
  - 中断是指在程序执行过程中，由硬件发出的信号，通知处理器有需要立即处理的事件。中断通常与外部设备（如键盘、鼠标、网络接口等）相关，用于处理外部请求或事件。
- 异常（Exceptions）：
  - 异常是指在程序执行过程中，由处理器自身检测到的错误或特殊条件（如除零错误、内存访问违规等）。异常通常表示程序执行过程中出现了问题，需要立即处理。

3.1.2 处理器如何处理

- 当中断或异常发生时，处理器会执行以下步骤：
  1. 保存上下文：处理器会保存当前程序的状态（上下文），包括程序计数器（PC）、寄存器等信息。
  2. 转移控制：处理器会根据中断或异常的类型，转移到对应的处理例程（中断处理程序或异常处理程序）。
  3. 执行处理程序：执行特定的中断或异常处理程序以处理事件或错误。
  4. 恢复上下文：处理完成后，恢复先前保存的上下文，继续执行被中断的程序。

> 处理器通过使用一个中断向量表（Interrupt Vector Table，IVT）来处理中断和异常。IVT 是一个存放在内存中的表格，它包含了不同类型的中断和异常对应的处理程序的入口地址。
>
> 当处理器检测到一个中断或异常时，它会根据事件的编号（也称为向量）在 IVT 中查找相应的处理程序的地址，然后保存当前的执行状态（如程序计数器、标志寄存器等），并跳转到该地址开始执行处理程序。

> 处理程序执行完毕后，处理器会恢复之前保存的执行状态，并返回到被中断或异常的指令继续执行。

3.1.3 思考：实模式和保护模式下，中断向量表一样吗？

- 实模式（Real Mode）：
  - 实模式下，IVT 的大小是 256 个字节，每个字节对应一个中断或异常向量，每个向量包含一个 16 位的段选择子和一个 16 位的偏移量，组成一个 32 位的线性地址。
- 保护模式（Protected Mode）：
  - 保护模式下，IVT 的大小是 1024 个字节，每个字节对应一个中断或异常向量，每个向量包含一个 8 位的中断描述符表（Interrupt Descriptor Table，IDT）选择子和一个 32 位的偏移量，组成一个 48 位的门描述符（Gate Descriptor）。IDT 是一个存放在内存中的表格，它包含了不同类型的门描述符，用于描述处理程序的属性和地址。

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

- 定义：
  - 中断和异常向量是一个 8 位的无符号整数，用于标识不同类型的中断和异常事件。
- 向量号范围：
  - 向量号的范围是 0 到 255。

Table 6-1. Protected-Mode Exceptions and Interrupts

Vector No.	Mne-monic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	RESERVED	Fault/ Trap	No	For Intel use only.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT 3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD2 instruction or reserved opcode.[1]
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction.[2]
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.
15	—	(Intel reserved. Do not use.)		No	
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes (Zero)	Any data reference in memory.[3]
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent.[4]
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions[5]
20	#VE	Virtualization Exception	Fault	No	EPT violations[6]
21-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Non-reserved) Interrupts	Interrupt		External interrupt or INT *n* instruction.

  - 0 到 31：由处理器预定义的异常和中断向量。这些向量用于特定的架构定义异常和错误，例如除零错误、无效操作码等。
  - 32 到 255：用户定义的中断向量。这些向量通常分配给外部 I/O 设备，允许这些设备通过外部硬件中断机制向处理器发送中断请求。
- 中断描述符表（IDT）：
  - 向量号作为索引用于中断描述符表（IDT），该表提供了对应的异常或中断处理程序的入口点，使得处理器能够快速响应不同的中断和异常事件。

3.2.2 中断源和异常源

中断源

中断源是指向处理器发送中断信号的来源。中断是一种机制，允许外部或内部事件打断正在执行的程序，以便处理器能够响应并处理这些事件。中断源主要可以分为两种类型：

1. 外部中断（Hardware Generated Interrupts）：
    - 由外部设备生成的中断信号。当外围设备（如键盘、鼠标、打印机、网络接口等）需要处理器的注意时，它们会向处理器发送中断请求。这种中断通常用于输入输出操作，确保设备能够及时与处理器进行通信。例如，当用户按下键盘键时，键盘会向处理器发出中断信号，通知其读取按键输入。
2. 软件生成的中断（Software-Generated Interrupts）：
    - 由软件指令引发的中断，通常通过执行特定的指令（如 `INT n`）来生成。软件中断允许程序主动请求操作系统提供服务，例如进行系统调用，或在遇到错误时触发异常处理程序。软件生成的中断使得应用程序能够与操作系统进行交互，处理特定的任务或错误情况。

异常源

异常源是指导致处理器产生异常的来源。异常是处理器在执行程序时检测到的特定错误或特殊条件，通常需要立即处理，例如除法溢出、缺页、保护错误、对齐检查、机器检查等处理器内部检测到的情况。。异常源可以分为三种类型：

1. 处理器检测到的程序错误异常（Processor-Detected Program-Error Exceptions）：
    - 这些异常是由处理器在执行指令时自动检测到的错误条件。例如，除以零、无效操作码、访问非法内存等。处理器会在遇到这些错误时产生异常，以便及时转移控制到异常处理程序。
2. 软件生成的异常（Software-Generated Exceptions）：
    - 这些异常是通过软件指令显式引发的。程序员可以通过特定的指令来生成异常，通常用于实现程序内部的错误检查或特定逻辑。例如，当程序遇到不可恢复的错误时，可以主动触发一个异常，以便进入错误处理流程。
3. 机器检查异常（Machine-Check Exceptions）：
    - 这些异常是由处理器内部硬件故障或系统级问题引起的。机器检查异常通常涉及对硬件状态的监测，并在检测到严重问题（如内存故障、总线错误等）时产生异常。此类异常通常由操作系统进行处理，以确保系统稳定性和数据完整性。

3.2.3 异常的分类：故障、陷阱和终止

> 异常可以分为故障（Faults）、陷阱（Traps）和中止（Aborts）三种类型，故障通常是可恢复的，而陷阱和中止通常是不可恢复的。

- 故障（Faults）：
  - 故障是一种可以被纠正的异常，它通常是由于程序的错误或缺乏资源导致的。处理器在发生故障时，会保存引起故障的指令的地址，然后执行故障处理程序。处理程序完成后，处理器会重新执行引起故障的指令，从而恢复正常的执行流程。例如，缺页异常就是一种故障，它表示虚拟内存中的某个页面没有被加载到物理内存中，需要操作系统将其调入并更新页表。
- 陷阱（Traps）：
  - 陷阱是一种用于实现系统调用或调试功能的异常，它通常是由于程序的正常请求或指令的执行导致的。处理器在发生陷阱时，会保存陷阱指令的下一条指令的地址，然后执行陷阱处理程序。处理程序完成后，处理器会继续执行陷阱指令的下一条指令，从而维持正常的执行流程。例如，`int` 指令就是一种陷阱，它可以用于实现系统调用，让用户程序请求操作系统的服务。
- 中止（Aborts）：
  - 中止是一种无法被纠正的异常，它通常是由于处理器的严重错误或不一致导致的。处理器在发生中止时，会保存一个不确定的地址，然后执行中止处理程序。处理程序通常无法恢复正常的执行流程，只能终止程序或重启系统。例如，机器检查异常就是一种中止，它表示处理器检测到了一个硬件故障或系统错误。

3.2.4 程序或任务的重新执行

程序或任务的重启是指在处理中断或异常后，将程序或任务恢复到故障指令处并重新执行，以确保程序能够继续运行而不会丢失连续性。重启通常适用于故障类和陷阱类异常，使得程序可以在适当的时机继续执行。

- 故障类异常重启：
  - 对于故障类异常，返回的指令指针（在处理器生成异常时保存）指向故障指令。因此，当程序或任务在处理完故障后重启时，故障指令将被重新执行。这种重启机制通常用于处理因操作数访问被阻塞而产生的异常。例如，页面故障异常（#PF）会在程序尝试访问未在内存中的页面时触发。处理器会触发页面故障，异常处理程序会将该页面加载到内存中，然后重启故障指令，以便程序可以继续执行。
- 陷阱类异常：
  - 陷阱类异常在执行陷阱指令后立即报告。陷阱允许程序或任务继续执行，不会丢失程序的连续性。陷阱处理程序的返回地址指向陷阱指令之后要执行的指令，这样程序能够在陷阱处理完成后恢复正常执行。

- 中止类异常：
  - 中止类异常是不可恢复的，通常意味着程序遇到了致命错误或严重问题（如硬件故障）。在这种情况下，程序或任务无法重启，处理器通常会终止执行并报告错误。这类异常的处理通常涉及清理和释放资源，而不是重启指令。

3.2.5 开启和禁止中断

开用和禁用中断是指处理器是否响应外部设备或软件产生的中断的状态。通过控制中断的生成，系统可以在执行关键操作时防止中断干扰，确保任务的稳定性和一致性。

- 中断标志（IF）：
  - IF 标志控制处理器是否响应外部中断。当 IF 标志被设置为 1 时，处理器允许中断的生成；当 IF 被清除为 0 时，处理器将禁止中断。这种机制通常在需要执行重要或敏感操作时使用，以避免中断引起的上下文切换或任务干扰。
- 重启标志（RF）：
  - RF 标志用于控制异常处理后的指令重启。当发生异常时，RF 标志可以指示处理器在处理完异常后是否重启发生异常的指令。清除 RF 标志将允许指令重新执行，而设置该标志（`RF = 1`）则防止重启，确保程序能够继续执行而不重复出错。
- 处理器状态：
  - 处理器的状态还会影响中断的启用和禁用。例如，在某些特权级（如内核模式）下，处理器可能会允许中断，而在用户模式下则可能会禁用某些中断，以保护系统资源。

3.2.6 异常和中断的优先级

当多个异常或中断在指令边界处待处理时，处理器会根据优先级顺序来服务它们。以下是异常和中断的优先级（优先级从高到低）：

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

1. 最高优先级：
   - 硬件复位和机器检查：包括系统复位（RESET）和机器检查（Machine Check）。
2. 任务切换陷阱：
   - 当任务状态段（TSS）中的 T 标志被设置时，发生任务切换陷阱。
3. 外部硬件干预：
   - 包括 FLUSH、STOPCLK、SMI（系统管理中断）和 INIT。
4. 前一指令上的陷阱：
   - 如断点、调试陷阱（包括设置了 TF 标志或数据/I-O 断点）。
5. 不可屏蔽中断（NMI）：
   - NMI 是一种不能被 IF 标志屏蔽的硬件中断，通常用于处理紧急或关键任务。

6. 可屏蔽硬件中断：
    - 可屏蔽中断（可由 IF 标志控制）通常来自外部设备，可以被屏蔽以保护关键任务。
7. 代码断点故障：
    - 当代码执行中遇到断点时触发的异常。
8. 从下一条指令的取指故障：
    - 包括代码段限制违规、代码页故障等。
9. 下一条指令解码故障：
    - 如指令长度超过 15 字节、无效操作码、协处理器不可用等。
10. 最低优先级：
    - 执行指令时发生的故障，包括溢出、边界错误、无效 TSS、段不存在、堆栈故障、一般保护错误、数据页故障、对齐检查、浮点异常、虚拟化异常等。

- 异常和中断的优先级主要由以下几个因素决定：
    - 异常或中断的来源：处理器会根据异常或中断的来源，将它们分为以下四类：
        - 外部中断：由外部设备或软件产生的中断，如键盘、鼠标、时钟、系统调用等。
        - 内部异常：由处理器内部检测到的异常，如除法错误、缺页、保护错误等。
        - 软件异常：由软件主动触发的异常，如断点、溢出、单步执行等。
        - 任务切换：由任务门或任务寄存器产生的任务切换请求。
    - 异常或中断的向量：处理器会根据异常或中断的向量，将它们分为以下两类：
        - 非屏蔽中断：向量为 0 到 31 的异常或中断，它们不能被屏蔽或忽略，必须立即处理。
        - 可屏蔽中断：向量为 32 到 255 的异常或中断，它们可以被屏蔽或忽略，不一定立即处理。
    - 处理器的状态：处理器会根据自身的状态，影响异常或中断的处理方式：
        - 中断标志位：如果处理器的中断标志位（IF）为 0，表示关闭了中断，那么处理器只会处理非屏蔽中断，而忽略可屏蔽中断 1。如果处理器的中断标志位（IF）为 1，表示开启了中断，那么处理器会处理所有的异常或中断。

- 中断优先级：如果处理器正在处理一个异常或中断，那么它只会响应优先级更高的异常或中断，而忽略优先级相同或更低的异常或中断 1。优先级的判断规则如下：
  - 如果两个异常或中断的来源不同，那么按照以下顺序判断优先级：内部异常 > 外部中断 > 软件异常 > 任务切换。
  - 如果两个异常或中断的来源相同，那么按照以下规则判断优先级：非屏蔽中断 > 可屏蔽中断；向量小的 > 向量大的。

3.3 中断描述符表

3.3.1 中断描述符表的构成

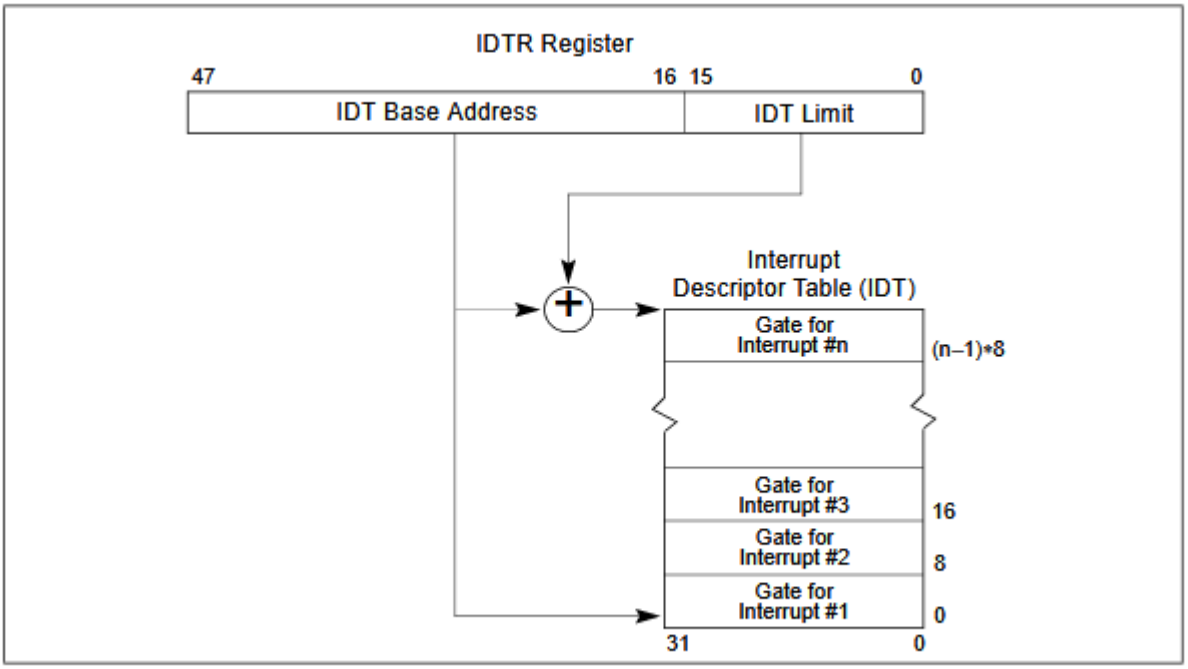

Figure 6-1. Relationship of the IDTR and IDT

- 描述符结构：IDT 是一个由 8 字节描述符组成的数组，每个描述符对应一个异常或中断向量，指向处理该中断或异常的例程。与全局描述符表（GDT）和局部描述符表（LDT）类似，IDT 也是一个数组结构，但与 GDT 不同的是，IDT 的第一个条目也可以包含描述符。
- 表的大小：IDT 的最大条目数为 256，因为 x86 体系结构中最多有 256 个异常和中断向量。可以设置少于 256 个描述符，仅为可能发生的中断和异常提供描述符。对于空闲的描述符条目，`present` 标志应设置为 0。
- 地址对齐：为了优化性能，IDT 的基址应当以 8 字节对齐，这样可以加速缓存填充。IDT 的大小限制值（limit）以字节为单位，加上基址后得出最后一个有效字节的地址。由于每个描述符固定为 8 字节长，因此限制值应为 8 的整数倍减 1（例如，8N–1）。

3.3.2 获得中断处理程序的地址

- 向量索引：当异常或中断发生时，处理器使用中断向量号来索引 IDT。向量号乘以 8 得出对应描述符在 IDT 中的位置。该描述符中包含了处理程序的地址，指向中断或异常处理例程的入口。
- 使用 IDTR 寄存器：IDTR（中断描述符表寄存器）保存了 IDT 的 32 位基地址和 16 位大小限制值。处理器通过 IDTR 寻址到 IDT，从而找到对应的中断或异常处理程序。

3.3.3 设置中断描述符表寄存器

- 加载 IDT 的指令：可以使用 `LIDT` 指令加载 IDTR 寄存器。LIDT 指令从内存操作数中读取基地址和大小限制值，加载到 IDTR 中。由于 LIDT 指令只能在当前特权级为 0（内核模式）时执行，它通常由操作系统的初始化代码执行，用于创建或替换 IDT。
- 存储 IDT 的指令：`SIDT` 指令用于将 IDTR 中的基地址和大小限制值复制到内存，可以在任意特权级执行。
- 边界检查：如果向量引用了超出 IDT 限制的描述符，则会引发一般保护异常（#GP），提示 IDT 配置不正确。

3.4 IDT 描述符

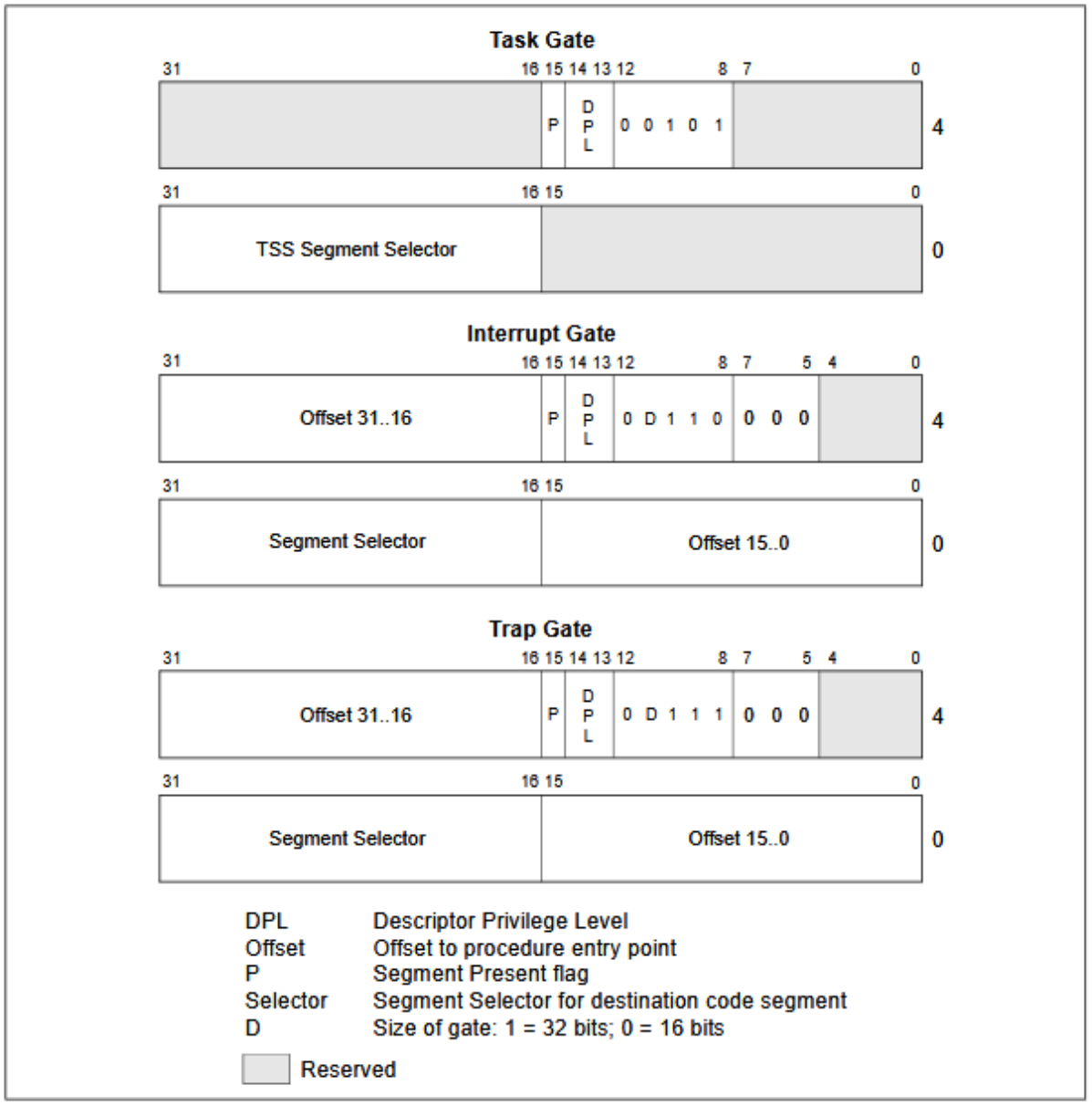

Figure 6-2. IDT Gate Descriptors

上图中 `D` 均为 `1`

- DPL：中段描述符权限等级
- Offset：偏移量
- P：P 标志位
- Selector：目标代码段的段选择子
- D：gate 的大小，1 = 32bits、0 = 16bits
- 阴影部分为保留位。

3.4.1 中断符

- 字段说明：
  - Offset 31:16：位于第 31 至 16 位，表示中断处理程序的偏移高位。

    - DPL (Descriptor Privilege Level) ：位于第 13 至 12 位，描述符特权级，控制对该中断门的访问权限。
    - P (Present Flag) ：第 15 位，表示该中断描述符是否有效。
    - Segment Selector：第 31 至 16 位，指向处理程序所在的代码段。
    - Offset 15:0：第 15 至 0 位，表示中断处理程序的偏移低位。
    - Type Field：值为 1110，表示这是一个中断门。
- 功能：中断门在中断发生时将控制转移到指定的中断处理程序。与陷阱门不同，中断门在调用时会清除 EFLAGS 寄存器中的 IF 标志，禁用其他硬件中断，以确保中断处理过程不会被其他中断打断。

3.4.2 陷进门

- 字段说明：
    - Offset 31:16：位于第 31 至 16 位，表示陷阱处理程序的偏移高位。
    - DPL (Descriptor Privilege Level) ：位于第 13 至 12 位，控制访问该陷阱门的权限级别。
    - P (Present Flag) ：第 15 位，表示描述符是否有效。
    - Segment Selector：第 31 至 16 位，指向陷阱处理程序所在的代码段。
    - Offset 15:0：第 15 至 0 位，表示陷阱处理程序的偏移低位。
    - Type Field：值为 1111，表示这是一个陷阱门。
- 功能：陷阱门在调用时不会修改 IF 标志，这意味着在陷阱处理期间，处理器可以继续响应其他中断。陷阱门通常用于调试和诊断目的，允许在处理特定异常时保持系统的响应能力。

3.4.3 任务门

- 字段说明：
    - DPL (Descriptor Privilege Level) ：位于第 13 至 12 位，表示描述符的特权级，控制调用该门所需的权限级别。
    - P (Present Flag) ：第 15 位，表示描述符是否存在，1 表示有效，0 表示无效。
    - TSS Segment Selector：位于第 31 至 16 位，指向任务状态段（TSS）的段选择子。
    - Type Field：值为 00101，表示该描述符为任务门。
- 功能：任务门用于在发生异常或中断时切换到指定的任务，通过 TSS 段选择子找到对应的任务状态段，保存当前状态并转移到新的任务。

3.5 中断与异常处理

3.5.1 中断过程调用的流程

1. 中断或异常触发：中断或异常通常是由外部事件、硬件设备或程序执行中的错误或特殊情况引发的。处理器检测到这些事件并确定其类型。
2. 中断向量查找：处理器使用中断向量作为索引，查找与中断或异常相关的 IDT 中的门描述符。这个中断向量用于标识中断或异常的类型。
3. 门描述符类型检查：处理器检查门描述符的类型，确定是中断门（Interrupt Gate）还是陷阱门（Trap Gate）。不同的类型会影响中断处理的方式。
4. 特权级别检查：处理器检查门描述符中的特权级别（DPL，Descriptor Privilege Level），以确定中断处理程序的特权级别和是否需要特权级别切换。
5. 当前状态保存：处理器会自动保存当前程序的状态，包括寄存器值、程序计数器（PC）、标志寄存器（FLAGS）等。这是为了确保在处理完中断后能够还原到原来的状态。
6. 中断处理程序执行：如果门描述符是中断门或陷阱门，处理器将跳转到描述符中指定的中断处理程序的地址。处理程序开始执行特定的操作，以处理中断或异常，这可能包括错误报告、中断服务例程、异常处理例程等。
7. 中断或异常处理程序执行完毕：一旦中断或异常处理程序执行完毕，处理器会根据处理程序的返回地址将控制返回到中断点，恢复先前保存的状态。
8. 恢复状态：处理器会恢复之前保存的状态，包括寄存器值和程序计数器，以便程序可以无缝继续执行。
9. 中断屏蔽恢复：如果中断处理程序在执行期间禁用了中断，处理器会将中断屏蔽位（中断标志位，IF）恢复为原来的状态，以允许中断再次响应。

3.5.2 如何判断中断处理过程与被中断任务的优先级?

- 中断处理程序的优先级由描述符中的特权级（DPL）和被中断任务的 CPL（当前特权级）来确定。特权级别是一个 2 位的无符号整数，取值为 0 到 3，其中 0 表示最高，3 表示最低。
- 对于硬件生成的中断和处理器检测到的异常，处理器会忽略中断或陷阱门的 DPL，因此通常会在最高特权级（内核级，PL0）运行。
- 如果中断是由软件生成的（例如 `INT n` 指令），处理器会检查中断或陷阱门的 DPL。只有当 CPL 小于或等于 DPL 时，才允许访问该中断处理程序，以防止低权限的应用程序访问关键的异常处理程序。

- 如果中断处理过程的特权级别等于被中断任务的特权级别，那么它们的优先级相同，称为同级中断。
- 如果中断处理过程的特权级别低于被中断任务的特权级别，那么它的优先级高于被中断任务，称为上级中断。
- 如果中断处理过程的特权级别高于被中断任务的特权级别，那么它的优先级低于被中断任务，称为下级中断。这种情况通常是不允许的，会导致一个异常。

3.5.3 不同优先级上，处理方式一样吗？

- 在相同特权级上（即 CPL 和中断处理程序的特权级一致），处理器会直接在当前堆栈上保存寄存器状态（包括 EFLAGS、CS、EIP），然后跳转到中断处理程序。此时不会发生堆栈切换。
- 在不同特权级上，对于上级中断（例如，从用户模式切换到内核模式），处理器会执行堆栈切换。它会从 TSS 中获取新的堆栈段选择子和堆栈指针，切换到更高特权级的堆栈，以保护低特权级的数据。对于下级中断，处理器会抛出一个异常，不会执行中断处理程序。

3.5.4 如果发生堆栈切换，处理器会做哪些操作？

当中断处理程序需要切换到不同的特权级时（例如，从用户模式 CPL=3 切换到内核模式 CPL=0），处理器会执行以下步骤：

- 获取新堆栈：处理器从 TSS（任务状态段）中获取目标特权级的段选择子和堆栈指针（SS 和 ESP）。
- 保存状态：在新堆栈上，处理器依次保存被中断任务的 SS 和 ESP（原堆栈段选择子和指针），以便在返回时恢复。
- 保存寄存器内容：处理器将 EFLAGS、CS 和 EIP 寄存器的当前状态压入新堆栈，以保存中断发生时的上下文信息。
- 保存错误代码（如果存在）：如果中断或异常包含错误代码（例如页面故障），处理器会将错误代码压入新堆栈。

完成以上操作后，中断处理程序可以在新的堆栈上执行。处理结束后，通过 `IRET` 指令返回，处理器会根据保存的堆栈信息恢复到被中断任务的原始状态。

3.5.5 如果没发生堆栈切换，处理器会做哪些操作？

如果中断处理过程发生在与被中断任务相同的特权级（即 CPL 和中断处理程序的特权级相同），则不会发生堆栈切换。在这种情况下，处理器会执行以下操作：

- 保存寄存器内容：处理器在当前堆栈上依次保存 EFLAGS、CS 和 EIP 寄存器的内容，以记录中断发生时的状态。
- 保存错误代码（如果有）：如果中断或异常有错误代码（如页面故障），处理器会将错误代码压入当前堆栈。

这样，中断处理程序可以直接在当前堆栈上执行，无需切换堆栈。

3.5.6 中断处理过程后，如何返回，处理器做了哪些操作?

中断处理过程结束后，处理程序通过 `IRET`（或 `IRETD`）指令返回。处理器在执行 `IRET` 时会进行以下操作：

- 恢复寄存器：`IRET` 会从堆栈中依次弹出 EIP、CS 和 EFLAGS 的值，恢复到中断发生前的状态。
- 恢复堆栈：如果发生了特权级切换，`IRET` 会同时切换回被中断任务的堆栈段选择子和堆栈指针（SS 和 ESP）。
- 恢复标志：EFLAGS 中的 IF（中断标志）和 IOPL（I/O 特权级）字段会根据中断前的值进行恢复，从而确保返回后程序的运行环境与中断前一致。

这些操作确保了程序在中断处理完成后可以无缝恢复到中断发生之前的状态。

3.5.7 异常和中断处理过程的保护

异常和中断处理过程的保护是指处理器在执行异常或中断处理程序时，采取的一些措施，以防止被其他的异常或中断干扰或破坏。

- 特权级保护：中断和异常处理过程的特权保护类似于通过调用门的普通过程调用。处理器不允许转移到特权级高于当前特权级的处理程序（即数值上更低的特权级）。
- 检查 DPL：处理器会检查中断或陷阱门的 DPL（描述符特权级），尤其是当使用 `INT n`、`INT 3` 或 `INTO` 指令生成中断时。仅当 CPL 小于或等于 DPL 时才允许访问，从而防止低特权级的程序访问关键的异常处理程序（例如页面故障处理程序）。
- 硬件中断和处理器检测的异常：对于这些情况，处理器会忽略 DPL，不考虑 RPL（请求特权级），直接访问对应的处理程序，以确保重要的异常和中断能够在高特权级执行。

这种保护机制有效地限制了不同特权级别的处理程序的访问，防止低权限任务干扰系统级的中断和异常处理。

3.5.8 异常和中断处理过程的标志使用方式

在异常或中断处理过程中，处理器会对 EFLAGS 中的某些标志进行特别处理：

- TF（单步标志）：当进入*异常或中断处理程序*时，处理器会清除 EFLAGS 中的 TF 标志，以防止指令跟踪（调试）影响中断响应。
- IF（中断标志）：如果通过中断门访问处理程序，处理器会清除 IF 标志，禁用其他硬件中断，确保中断处理过程不会被其他中断打断。
- VM、RF 和 NT 标志：在保存到堆栈后，处理器也会清除这些标志，以确保中断或异常处理过程的正常执行。
- IRET 恢复标志：`IRET` 指令在返回时会从堆栈中恢复 EFLAGS，确保返回后 EFLAGS 恢复到中断发生前的状态，包括恢复 TF 和 IF 等标志的值。

这种标志处理机制确保了中断或异常处理过程的隔离性和稳定性。

3.5.9 中断门与陷阱门的唯一区别是什么？

中断门与陷阱门的唯一区别在于它们对 EFLAGS 中 IF（中断标志） 的处理方式：

- 中断门：进入中断处理程序时，处理器会清除 IF 标志，禁用硬件中断，以防止其他中断干扰当前的中断处理过程。随后的 IRET 指令将 IF 标志恢复为堆栈上 EFLAGS 寄存器中保存的内容中的值。这种方式用于需要无干扰的中断处理。
- 陷阱门：进入陷阱处理程序时，处理器不会修改 IF 标志，即其他硬件中断在陷阱处理过程中仍然可以被触发。这种方式适用于调试和诊断用途，允许系统保持响应能力。

中断门和陷阱门的其他方面，如格式、特权级别、堆栈切换等，都是相同的。