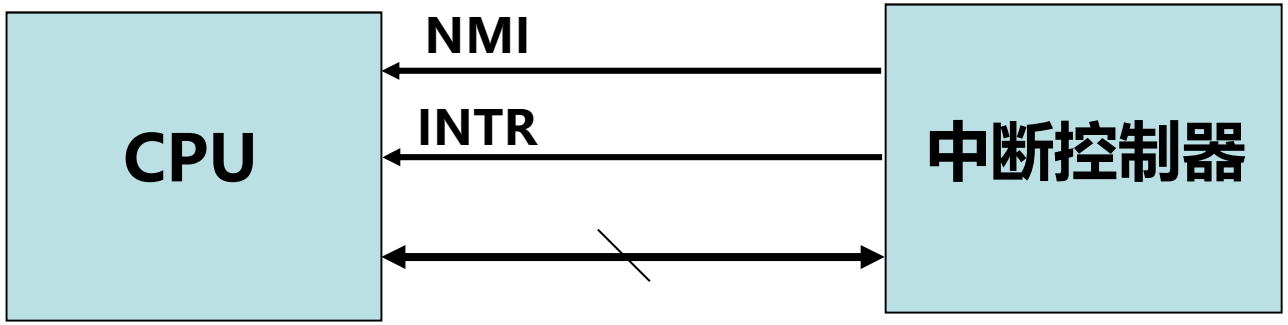
IA-32的向量中断方式

- 有256种不同类型的异常和中断
- 每个异常和中断都有唯一编号，称之为中断类型号（也称向量号）。如类型0为“除法错”，类型2为“NMI中断”，类型14为“缺页”
- 每个异常和中断有与其对应的异常处理程序或中断服务程序，其入口地址放在一个专门的中断向量表或中断描述符表中。
- 前32个类型（0~31）保留给CPU使用，剩余的由用户自行定义（这里的用户指机器硬件的用户，即操作系统）
- 通过执行INT n（指令第二字节给出中断类型号n，n=32~255）使CPU自动转到OS给出的中断服务程序执行
- 实模式下，用中断向量表描述
- 保护模式下，用中断描述符表描述

回顾：中断的分类

- Intel将中断分成**可屏蔽中断**（maskable interrupt）和**不可屏蔽中断**（nonmaskable interrupt，NMI）。
  - **可屏蔽中断**：通过 INTR 向CPU请求，可通过设置**屏蔽字**来屏蔽请求，若中断请求被屏蔽，则不会被送到CPU。
  - **不可屏蔽中断**：非常紧急的硬件故障，如：电源掉电，硬件线路故障等。通过 NMI 向CPU请求。一旦产生，就被立即送CPU，以便快速处理。这种情况下，中断服务程序会尽快保存系统重要信息，然后在屏幕上显示相应的消息或直接重启系统。



IA-32的中断类型

- **用户自定义类型号为32~255，部分用于可屏蔽中断，部分用于软中断**
- **可屏蔽中断通过CPU的INTR 引脚向CPU发出中断请求**
- **软中断指令 INT n 被设定为一种陷阱异常，例如，Linux通过int $0x80指令将128号设定为系统调用，而Windows通过int $0x2e指令将46号设定为系统调用。**

类型号	助记符	含义描述	起因或发生源
0	#DE	除法出错	div 和 idiv 指令
1	#DB	单步跟踪	任何指令和数据引用
2		NMI 中断	不可屏蔽外部中断
3	#BP	断点	int 3 指令
4	#OF	溢出	into 指令
5	#BR	边界检测（BOUND）	bound 指令
6	#UD	无效操作码	不存在的指令操作码
7	#NM	协处理器不存在	浮点或 wait/fwait 指令
8	#DF	双重故障	处理一个异常时发生另一个
9	#MF	协处理器段越界	浮点指令
.....			
19	#XM	SIMD 浮点异常	SIMD 浮点指令
20-31		保留	
32-255		可屏蔽中断和软中断	INTR 中断或 INT *n* 指令

回顾：用“系统思维”分析问题

代码段一：

```
int a = 0x80000000;
int b = a / -1;
printf("%d\n", b);
```

运行结果为-2147483648

objdump反汇编代码，得知除以 -1 被优化成取负指令neg，故未发生除法溢出

代码段二：

```
int a = 0x80000000;
int b = -1;
int c = a / b;
printf("%d\n", c);
```

a/b用除法指令IDIV实现，但它不生成OF标志，那么如何判断溢出异常的呢?

实际上是“除法错”异常#DE（类型0）

Linux中，对#DE类型发SIGFPE信号

运行结果为“Floating point exception”，显然CPU检测到了溢出异常

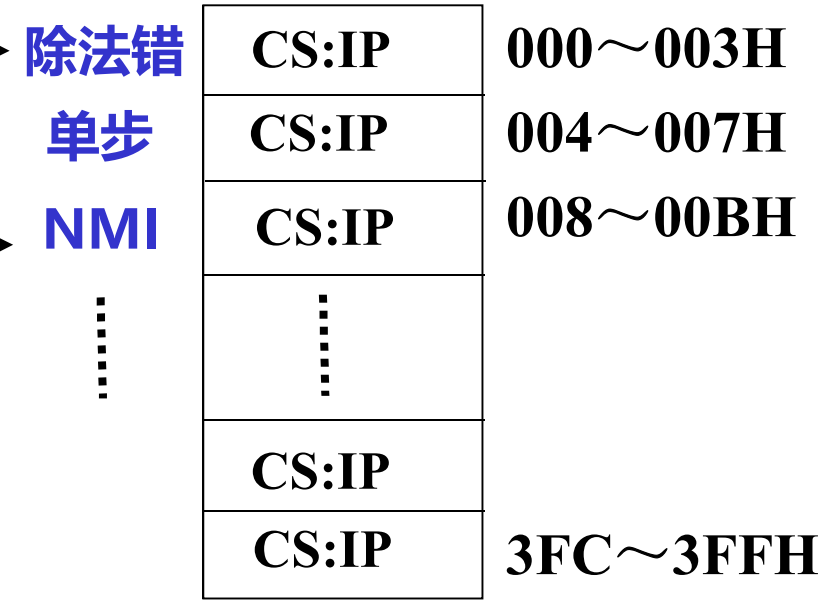
为什么两者结果不同！

实地址模式下的中断向量表

实地址模式（Real Mode）是Intel为80286及其之后的处理器提供的一种8086兼容模式。寻址空间1MB，指令地址=CS<<4+IP 。中断向量表位于0000H～03FFH。共256组，每组占四个字节 CS:IP 。

例1：除法错的中断类型号为0，故其向量地址为：0x4=0

例2：NMI的中断类型号为2，故其向量地址为：2x4=8



实地址模式下没有分页管理机制！

中断向量表中每一项是对应中断服务程序或异常处理程序的入口地址，被称为中断向量(Interrupt Vector)。

实地址模式下的中断向量表

- 开机后系统首先在实地址模式下工作（只有1MB的寻址空间）
- 开机过程中，需要先准备在实模式下的中断向量表和中断服务程序。通常，由固化在主板上一块ROM芯片中的BIOS程序完成
- BIOS程序检测显卡、键盘、内存等，并在00000H～003FFH区建立中断向量表，在中断向量所指主存区建立相应的中断服务程序
- BIOS利用INT指令执行特定的中断服务程序把OS从磁盘加载到内存中。例如，BIOS可通过执行int 0x19指令来调用中断向量0x19对应的中断服务程序，将启动盘上的0号磁头对应盘面的0磁道1扇区中的引导程序装入内存
- BIOS（Basic Input/Output System）是基本输入/输出系统的简称，是针对具体主板设计的，与安装的操作系统无关。
- BIOS包含各种基本设备驱动程序，通过执行BIOS程序，基本设备驱动程序以中断服务程序的形式被加载到内存，以提供基本I/O系统调用。
- 一旦进入保护模式，就不再使用BIOS。

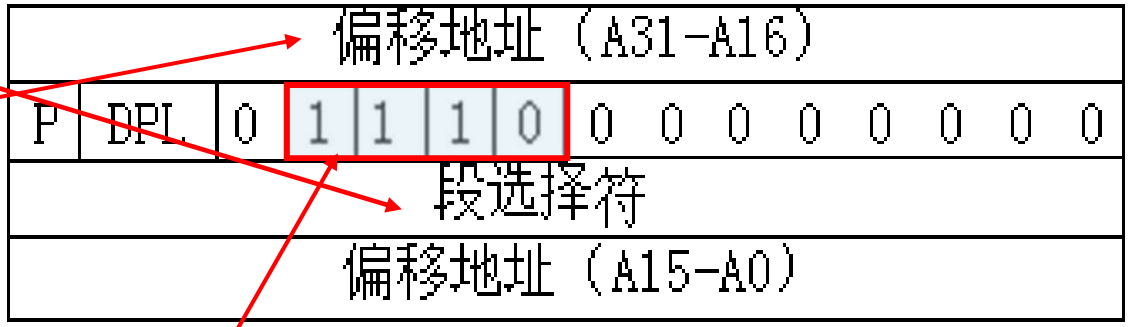
保护模式下的中断描述符表

- 保护模式下，通过中断描述符表获异常处理或中断服务程序入口地址
- 中断描述符表（Interrupt Descriptor Table，IDT）是OS内核中的一个表，共有256个表项，每个表项占8个字节，IDT共占用2KB
- IDTR中存放 IDT在内存的首地址
- 每一个表项是一个中断门描述符、陷阱门描述符或任务门描述符

段选择符用来指示异常处理程序或中断服务程序所在段的段描述符在GDT中的位置，其RPL=0；

偏移地址则给出异常处理程序或中断服务程序第一条指令所在偏移量。

中断门描述符格式：



P：Linux总把P置1。DPL：访问本段要求的最低特权级。主要用于防止恶意应用程序通过 INT n 指令模拟非法异常而进入内核态执行破坏性操作

TYPE：标识门的类型。TYPE=1110B：中断门；TYPE=1111B：陷阱门；

TYPE=0101B ： 任务门

IA-32中异常和中断的处理

- 引导程序被读到内存后，开始执行引导程序，以装入操作系统内核，并对GDT、IDT等进行初始化，系统启动后，进入保护模式
- IA-32中，每条指令执行后，下条指令的逻辑地址（虚拟地址）由CS和EIP指示　　实地址模式下：指令地址=CS<<4+IP
- 每条指令执行过程中，CPU会根据执行情况判定是否发生了某种内部异常事件，并在每条指令执行结束时判定是否发生了外部中断请求

  （由此可见，异常事件和中断请求的检测都是在某一条指令执行过程中进行的，显然由硬件完成）
- 在CPU根据CS和EIP取下条指令之前，会根据检测的结果判断是否进入中断响应阶段

  （异常和中断的响应也都是在某一条指令执行过程中或执行结束时进行的，显然也由硬件完成）

IA-32中异常和中断响应过程 SKIP

(1) 确定中断类型号 i，从 IDTR 指向的 IDT 中取出第 i 个表项 IDTi。

(2) 根据 IDTi 中段选择符，从 GDTR 指向的 GDT 中取出相应段描述符，得到对应异常或中断处理程序所在段的 DPL、基地址等信息。Linux下中断门和陷阱门对应的即为内核代码段，所以DPL为0，基地址为0。

(3) 若CPL<DPL或编程异常 IDTi 的 DPL<CPL，则发生13号异常。Linux下，前者不会发生。后者用于防止恶意程序模拟 INT n 陷入内核进行破坏性操作。

(4) 若CPL≠DPL，则从用户态换至内核态，以使用内核栈。切换栈的步骤：

① 读 TR 寄存器，以访问正在运行的用户进程的 TSS段；

② 将 TSS段中保存的内核栈的段选择符和栈指针分别装入寄存器 SS 和 ESP，然后在内核栈中保存原来用户栈的 SS 和 ESP。

(5) 若是故障，则将发生故障的指令的逻辑地址写入 CS 和 EIP，以使处理后回到故障指令执行。其他情况下，CS 和 EIP 不变，使处理后回到下条指令执行。

(6) 在当前栈中保存 EFLAGS、CS 和 EIP 寄存器的内容（断点和程序状态）。

(7) 若异常产生了一个硬件出错码，则将其保存在内核栈中。

(8) 将IDTi中的段选择符装入CS，IDTi中的偏移地址装入EIP，它们是异常处理程序或中断服务程序第一条指令的逻辑地址（Linux中段基址=0）。

下个时钟周期开始，从CS:EIP所指处开始执行异常或中断处理程序！

回顾：IA-32/Linux中的分段机制

- 为使能移植到绝大多数流行处理器平台， Linux简化了分段机制
- RISC对分段支持非常有限，因此Linux仅使用IA-32的分页机制，而对于分段，则通过在初始化时将所有段描述符的基址设为0来简化
- 若把运行在用户态的所有Linux进程使用的代码段和数据段分别称为**用户代码段和用户数据段**；把运行在内核态的所有Linux进程使用的代码段和数据段分别称为**内核代码段和内核数据段**，则Linux初始化时，将上述4个段的段描述符中各字段设置成下表中的信息：

段	基地址	G	限界	S	TYPE	DPL	D	P
用户代码段	0x0000 0000	1	0xFFFFF	1	10	3	1	1
用户数据段	0x0000 0000	1	0xFFFFF	1	2	3	1	1
内核代码段	0x0000 0000	1	0xFFFFF	1	10	0	1	1
内核数据段	0x0000 0000	1	0xFFFFF	1	2	0	1	1

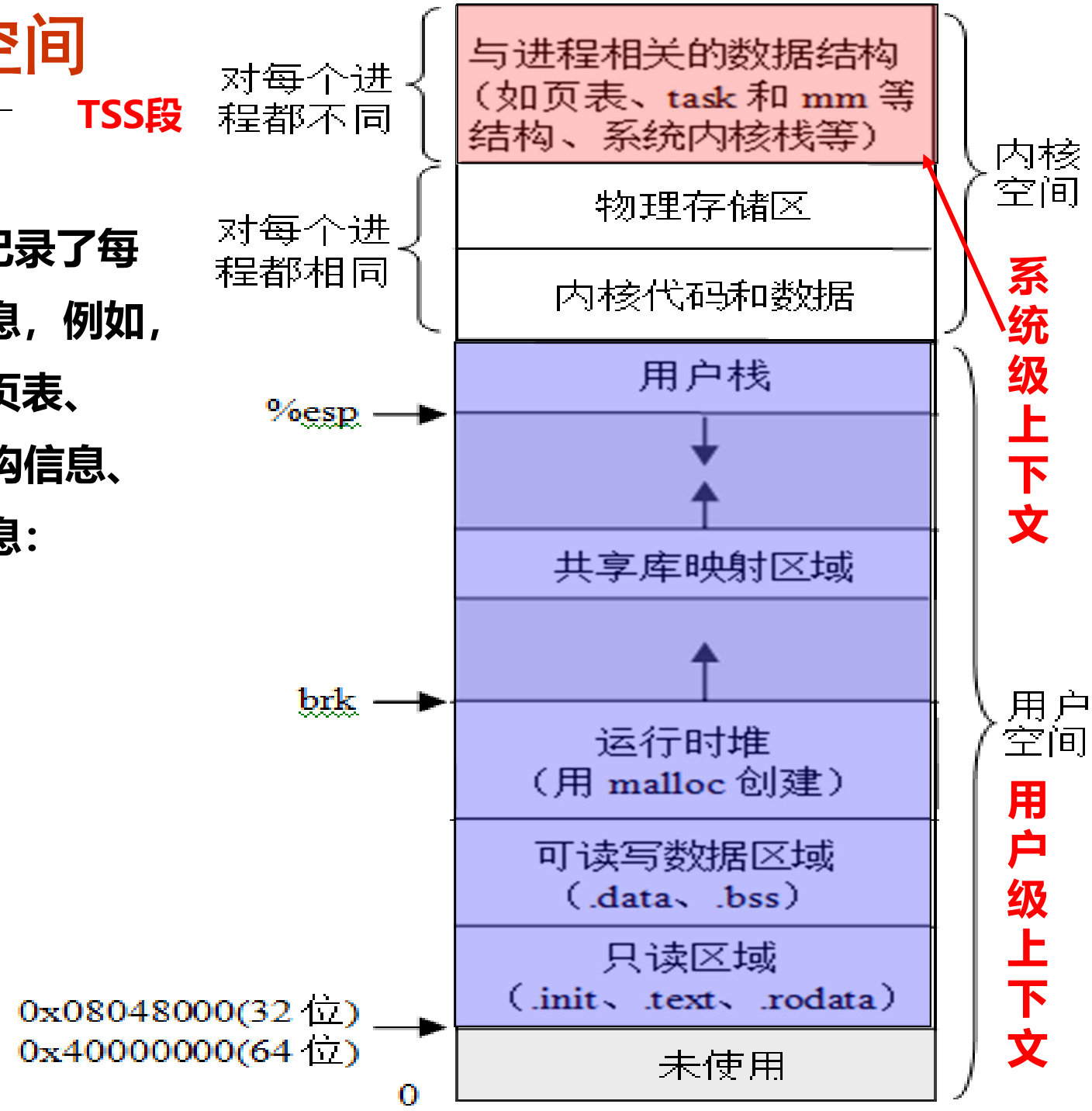
初始化时，上述4个段描述符被存放在GDT中

进程的地址空间

TSS段

内核中的TSS段记录了每个进程的状态信息，例如，每个进程对应的页表、task和mm等结构信息、内核栈的栈顶信息：SS:ESP 等

BACK



IA-32中异常和中断返回过程

中断或异常处理程序最后一条指令是IRET。CPU在执行IRET指令过程中完成以下工作：

(1) 从栈中弹出硬件出错码（保存过的话）、EIP、CS和EFLAGS

(2) 检查当前异常或中断处理程序的CPL是否等于CS中最低两位，若是则说明异常或中断响应前、后都处于同一个特权级，此时，IRET指令完成操作；否则，再继续完成下一步工作。

(3) 从内核栈中弹出SS和ESP，以恢复到异常或中断响应前的特权级进程所使用的栈。

(4) 检查DS、ES、FS和GS段寄存器的内容，若其中有某个寄存器的段选择符指向一个段描述符且其DPL小于CPL，则将该段寄存器清0。这是为了防止恶意应用程序（CPL=3）利用内核以前使用过的段寄存器（DPL=0）来访问内核地址空间。

执行完IRET指令后，CPU回到原来发生异常或中断的进程继续执行