第4课 硬件中断的原理与实践

《跟着瓦利哥学写OS》

联系方式

- 自己动手写操作系统QQ群：82616767。
- 申请考试邮箱：sangwf@gmail.com
- 所有课程的代码都在Github：

https://github.com/sangwf/walleclass

思考题回顾

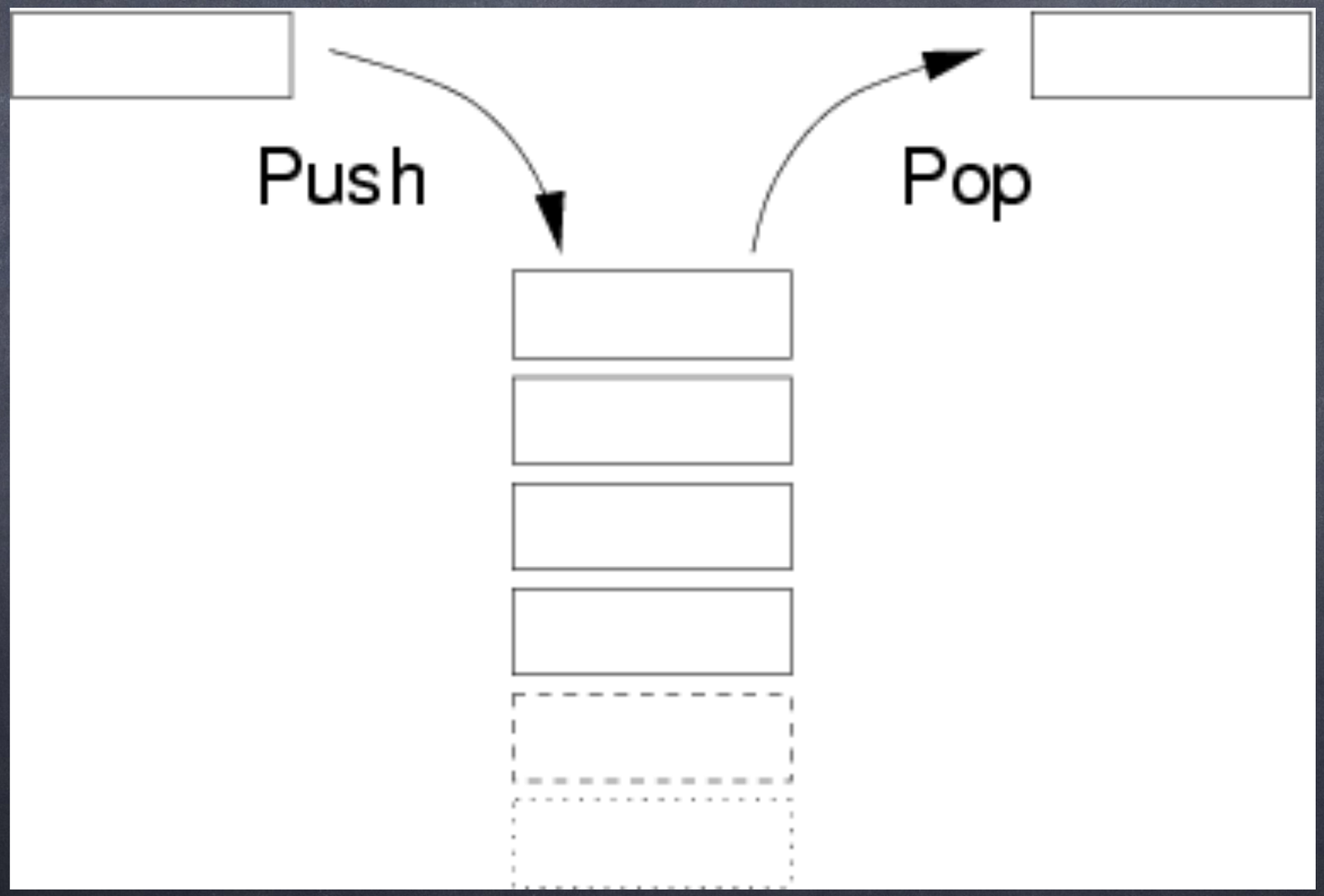
- 在实验中，如何实现虚拟地址被访问时，再分配物理内存？

认识栈（Stack）

- 栈是一种后进先出的数据结构，就像一个下水管道被封住了一头，大家依次往里钻，最先进去的最后才能出来。

- 只有两个操作：
  - 入栈（Push）：在尾部放入一个元素。
  - 出栈（Pop）：在尾部弹出一个元素。

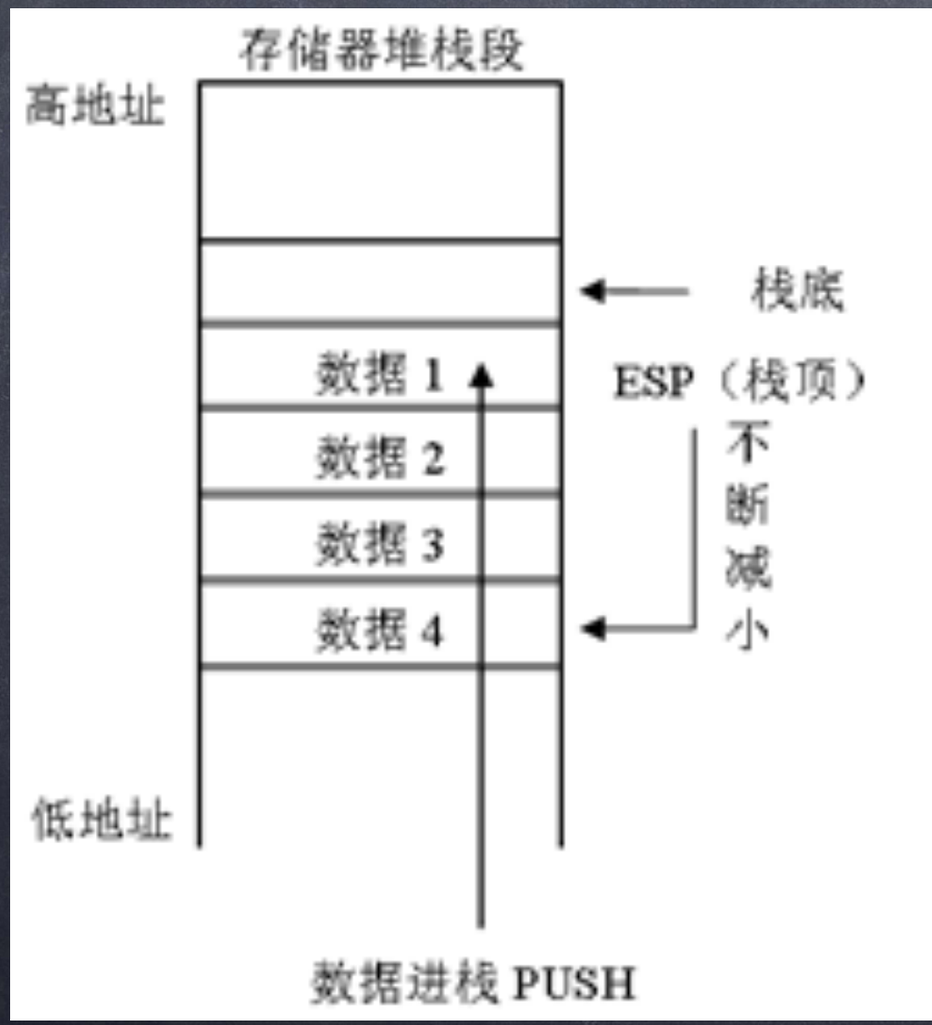
认识栈（Stack）

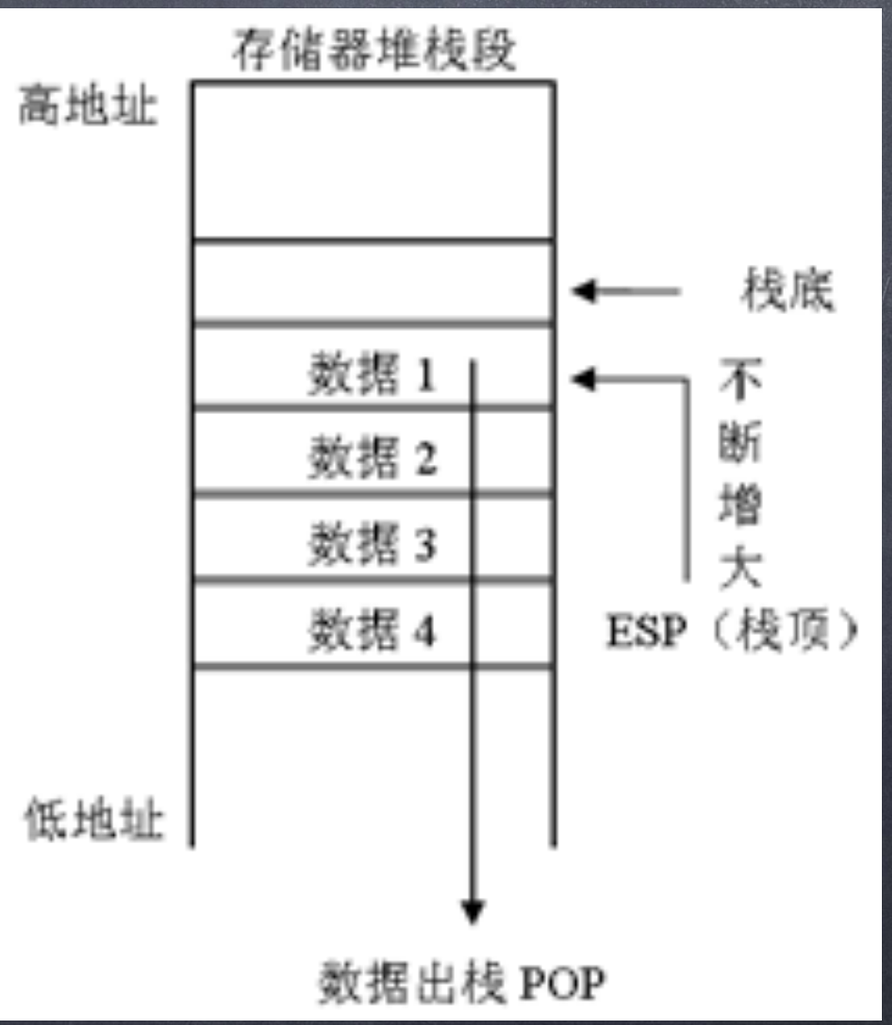


认识栈（Stack）

- 栈对应的寄存器：
  - SS：段选择符
  - ESP：段尾地址寄存器
- 栈对应的汇编指令：
  - lss esp, [init_stack]：将init_stack内存的内容低32位赋值给esp，并将高16位（段选择符）赋值给ss。
  - push eax：将eax压入栈尾，esp = esp - 4。
  - pop eax：将栈尾的值弹出到eax，esp = esp + 4。

认识栈（Stack）


存储器堆栈段
高地址
栈底
数据 1
数据 2
数据 3
数据 4
ESP（栈顶）
不断减小
低地址
数据进栈 PUSH


存储器堆栈段
高地址
栈底
数据 1
数据 2
数据 3
数据 4
不断增大
ESP（栈顶）
低地址
数据出栈 POP

认识栈（Stack）

- 一种约定俗成的习惯，将内存地址描述成一个竖条，高地址在上面，低地址在下面。向上生长就是地址变大，向下生长就是地址变小。

- 在X86体系中，栈是向下生长的，push压栈时，esp的地址减小。

认识栈（Stack）

- 栈的用途：
  - 函数调用时，保存函数下一条指令地址（EIP）、参数等。
  - 中断调用时，保存现场，即保存一些寄存器之类的值。
  - 申请临时变量。

Stack Overflow

- 栈溢出：在程序中，每个进程的栈大小是有限制的（8M/4M），当申请临时变量或递归调用过多时，超出了栈的最下边界，就溢出了。

中断（Interrupt）

- 我们今天只讲硬件中断，即由CPU的外围设备，如时钟，键盘等所触发的中断。

- 软件中断如int n，或缺页异常（exception）之类的放到下节课再讲。

中断（Interrupt）

- 假设你在开车，前面路口红灯了，你得等横向的车走过变绿灯后，你再继续往前走。

- CPU是很傻的，它只知道永不停歇的获取下一条指令，然后执行。为了和外部设备交互，在执行完成一条指令之后，会先检查是否有中断标记，有的话，就先执行中断，完成中断后，再继续执行下一条指令。

- 注意：不会出现一条CPU指令执行了一半被中断了。

硬件中断的原理

- 每一个硬件中断都会有个唯一的硬件编号，当发生中断时，CPU可以通过这个编号，到中断描述符表里找到对应的处理程序入口地址，类似执行一个函数调用。

- 但中断程序执行完成之后，CPU怎么知道该怎么继续执行呢？这就靠在函数调用前，先将EIP的值压入栈，中断调用完成时，再弹出到EIP，这样就可以继续执行了。

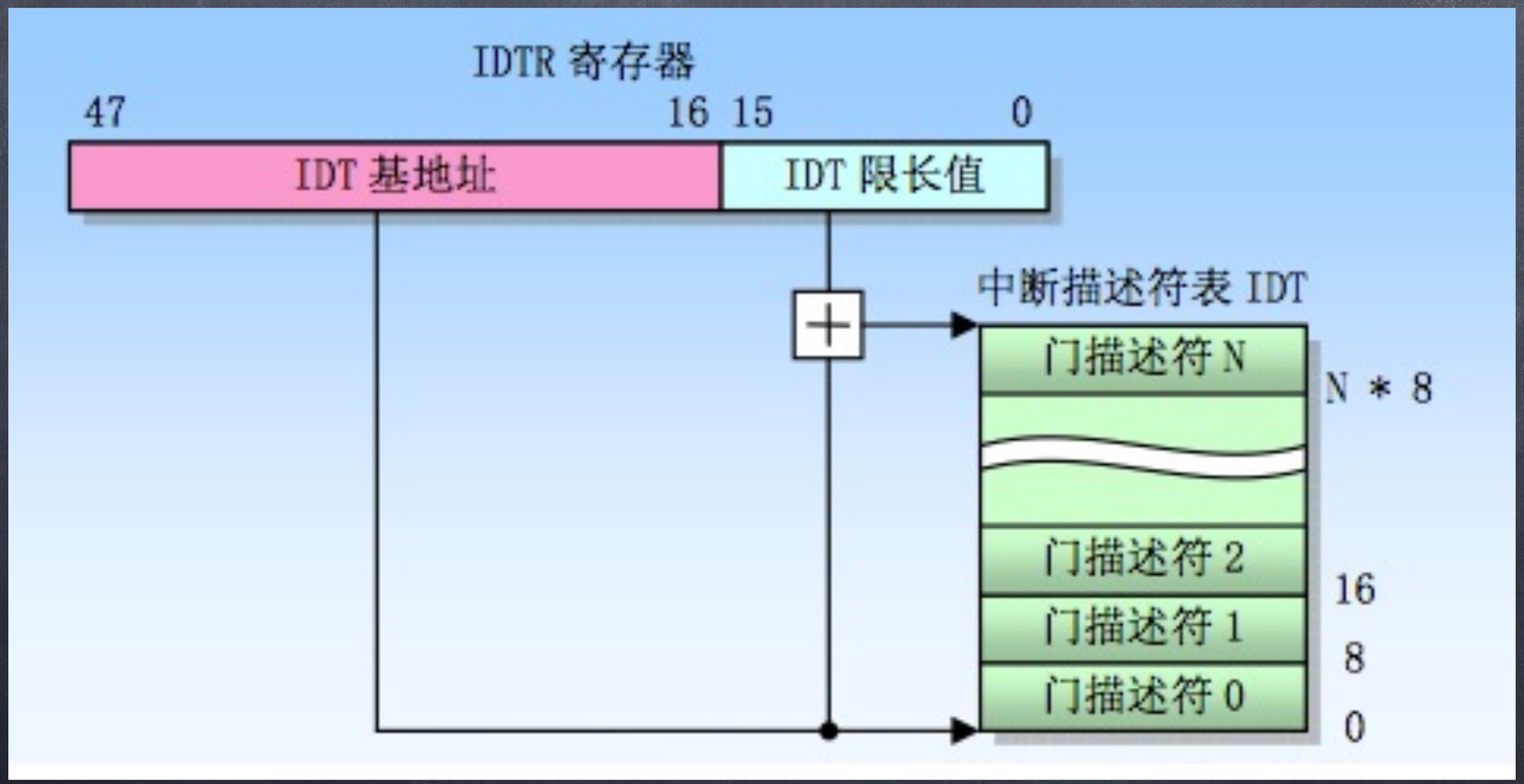
硬件中断的原理

- 硬件中断的处理过程中，其他硬件中断是被屏蔽的。也就是硬件中断不会被其他硬件中断重新打断。
- 汇编指令：
  - cli：关闭中断（if = 0），在一些关键逻辑处理时，需要先关闭中断。注意，软件中断（int n）并不受此影响。
  - sti：开启中断（if = 1）。
  - 这两条指令都是修改的寄存器eflags中的if标记位。

中断描述符表（IDT）

- IDT：Interrupt Descriptor Table
  - 类似全局描述符表（GDT），格式相近，内容不同。
  - 存放了中断门、陷阱门、任务门三类描述符，本节只讲中断门。

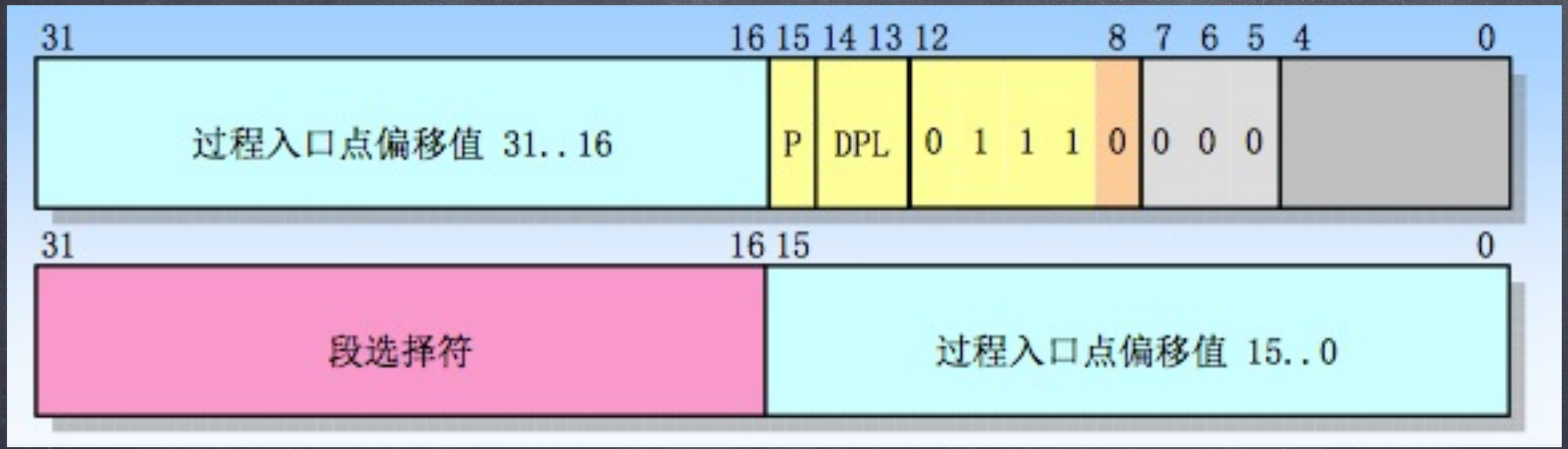
中断描述符表（IDT）


IDTR 寄存器
47
16 15
0
IDT 基地址
IDT 限长值
中断描述符表 IDT
门描述符 N
N * 8
门描述符 2
门描述符 1
门描述符 0
16
8
0

什么是中断门？

- 玩过Diablo的同学都知道，其中有个传送门。中断门也类似，就是这个门里记录了中断程序所处的段选择符，入口偏移地址，以及权限控制信息。其实就是个中介。

中断门描述符



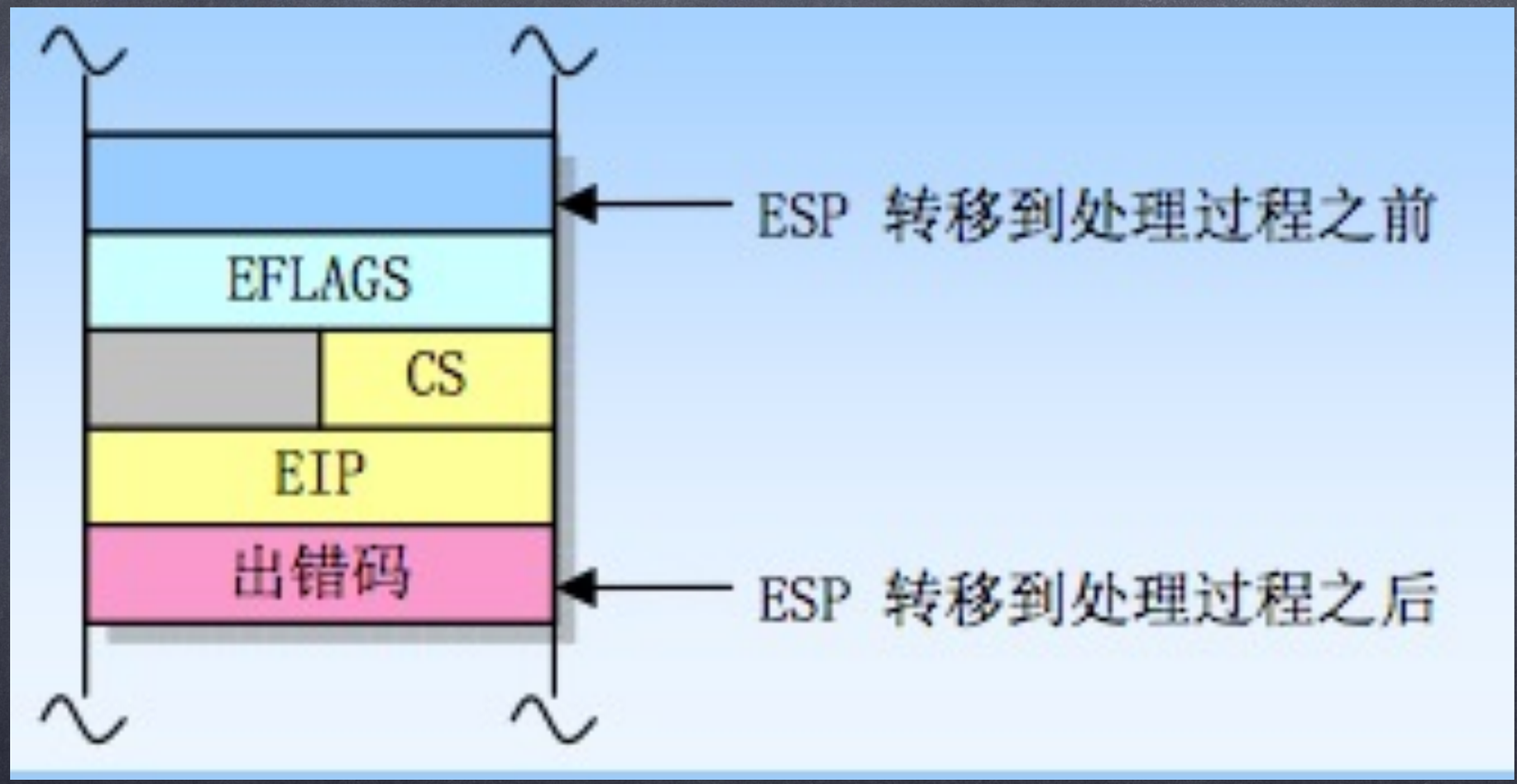
P：段是否存在的标志

DPL：描述符特权级

中断的初始化步骤

- 0，先初始化栈，以备后用。
- 1，初始化中断描述符表（IDT）。
- 2，关闭中断（cli）。
- 3，通过lidt加载中断描述符表基地址和限长。
- 4，开启中断（sti）。
- 5，等着硬件中断的到来。

中断调用时的栈变化



第一个实验

- 在第二节课的保护模式的代码基础上（不用分页），实现所有的中断产生时，都在屏幕上打印一个字符I。

boot.s

```
15 mov al, 10          ;读取的扇区数为10
16                    ;注意: 如果读取的内容超过512byte, 则加大
17 mov ah, 0x02       ;AH = 2, 表示读扇区
18 int 0x13
19
20 cli ;关闭中断
21 mov ax, 0x1000
22 mov ds, ax
23 xor ax, ax ;移动到内存0地址
24 mov es, ax
25 mov cx, 1280 ;移动cx个double word(4 bytes)
26 sub si, si
27 sub di, di
28 rep movsd
```

代码（printi.s）

```
1  KERNEL_SEL equ 0x08
2  SCREEN_SEL equ 0x10
3  DATA_SEL   equ 0x18
4
5  bits 32 ;这是32位的指令
6
7  mov eax, DATA_SEL ;先设置数据段
8  mov ds, eax
9  mov es, eax
10
11 ;初始化栈
12 lss esp, [init_stack]
13
14 cli ;关中断
```

代码（printi.s）

```
16 lea edx, [int_ignore] ;将int_ignore的地址写入edx
17 mov eax, 0x00080000 ;eax高16位设置段选择符0x0008
18 mov ax, dx ;将int_ignore写入ax
19                ;到此为止，eax存放了一个中断描述符的
20                ;低32位：段选择符 + 偏移地址(低16位)
21 mov dx, 0x8E00 ;1000 1110 0000 0000
22                ;P = 1, DPL = 00
23                ;中断门的标记01110000
24                ;最后5位无用
25                ;dx用于中断门的第32~47位
26                ;目前为止，edx为中断门的高32位
27 lea edi, [idt] ;将idt的地址写入edi
28 mov ecx, 256 ;一共256个中断描述符
```

代码（printi.s）

```
30 ;循环初始化所有中断门为int_ignore
31 rp_idt:
32 mov [edi], eax
33 mov [edi + 4], edx
34 add edi, 8
35 dec ecx
36 jne rp_idt
37
38
39 ;加载中断描述符表基地址/限长
40 lidt [lidt_opcode]
41
42 sti ;开中断
43
44 LOOP1:
45     jmp LOOP1 ;程序在这里就结束了
```

代码（printi.s）

```
47 align 4
48 int_ignore: ;默认的硬件中断处理函数
49     push eax
50     push ebx
51     push ecx
52     push edx
53     push ds
54     push es
55     push gs
56
57     mov eax, DATA_SEL ;先设置数据段
58     mov ds, eax
59     mov es, eax
60
61     mov al, 'I'
62
63     mov ebx, SCREEN_SEL
64     mov gs, bx
65     mov ebx, [scr_loc] ;现在的输出地址
```

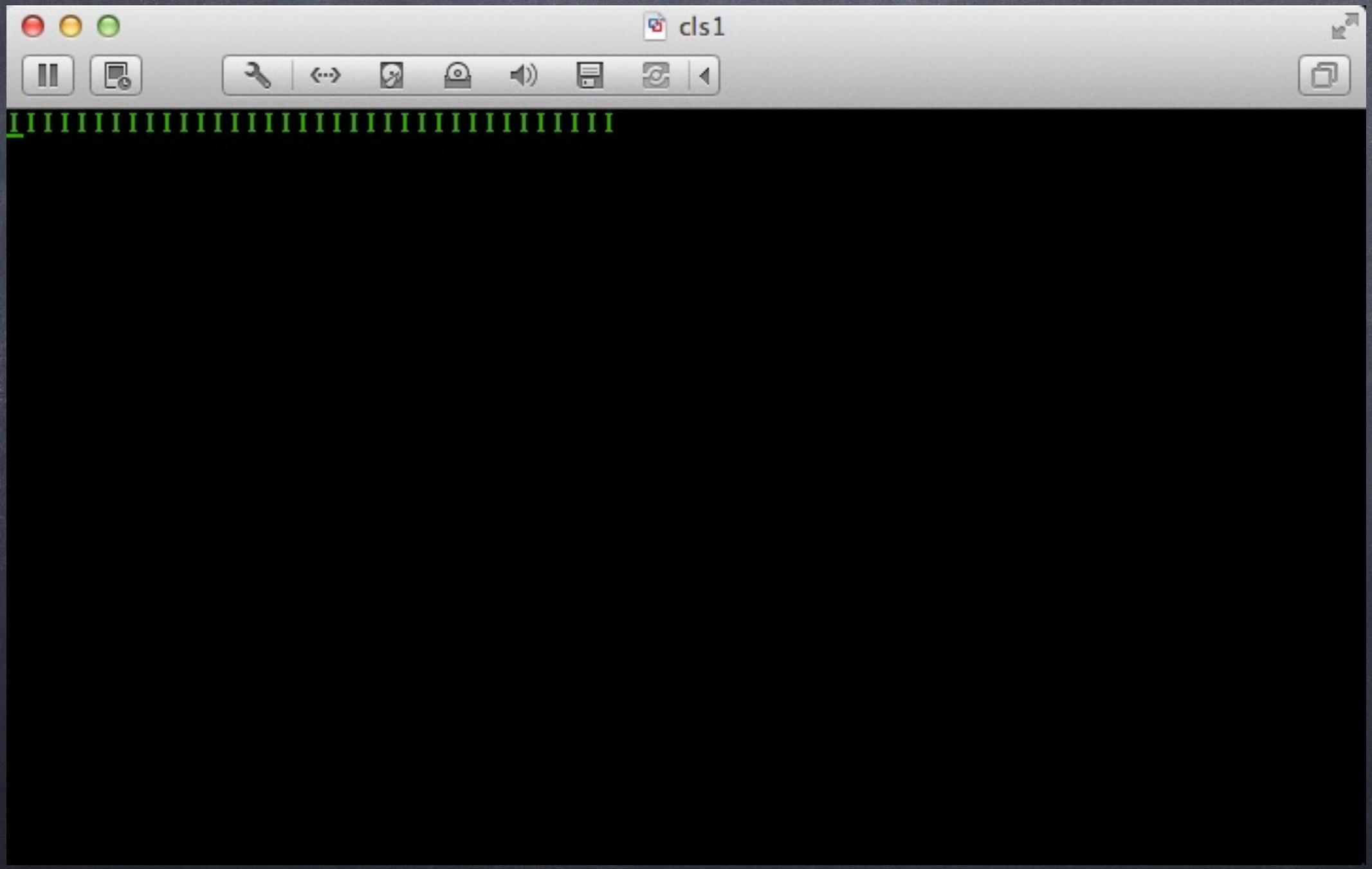
代码（printi.s）

```
67      shl ebx, 1 ; * 2
68      mov [gs: ebx], al
69      mov byte [gs: ebx + 1], 0x02
70
71      shr ebx, 1 ; / 2
72      add ebx, 1
73      cmp ebx, 2000 ;最多25*80 = 2000字符
74      jne .update_scr_loc
75      mov ebx, 0
76  .update_scr_loc:
77      mov [scr_loc], ebx ;更新位置
78      ;end
79
80      push gs
81      push es
82      push ds
83      push edx
84      push ecx
85      push ebx
86      push eax
87      iret
```

代码（printi.s）

```
89 scr_loc:
90      dd 0 ;当前位置
91 ;idt寄存器的加载数据
92 lidt_opcode:
93      dw 256 * 8 - 1
94      dd idt
95
96 align 8
97 idt:
98      times 2048 db 0 ;256个，每个8 bytes
99
100 times 128 dd 0 ;栈的大小为128个4 bytes
101 init_stack: ;栈初始化数据
102     dd init_stack ;32位偏移地址
103     dw DATA_SEL ;16位段选择符
```

运行结果



思考

- 硬件中断处理过程中，会有新的硬件中断打断它吗？

谢谢！