第5课 异常的原理与实践

《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

- 硬件中断处理过程中，会有新的硬件中断打断它吗？

讲解更正1

- push eax等价于：
  - esp = esp - 4;
  - mov [esp], eax

回顾硬件中断

- 由CPU之外的硬件触发，是外界硬件与CPU交互的方式。

异常（Exception）

- 由CPU执行过程中，程序触发的。

异常的分类

- Fault（故障）：可以被纠正的异常，如缺页异常（Page Fault），异常处理后，引起异常的指令会重复执行。

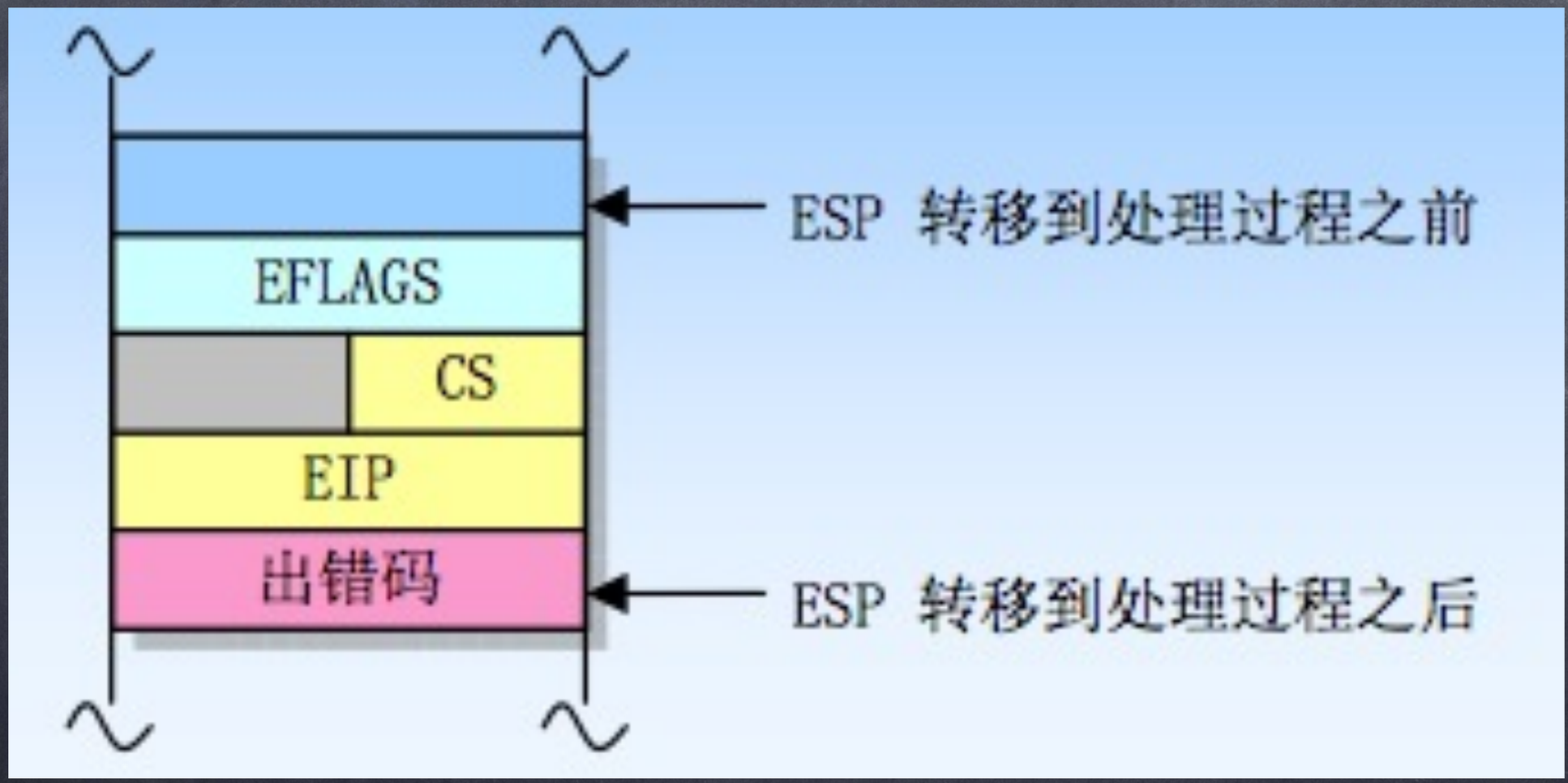
- Trap（陷阱）：专门引起陷阱的指令，有意为之，如int n。int n又称为软件中断。

- Abort（终止）：严重错误，指令位置无法确切定位，关键系统数据存在异常，最好关闭。

Fault

- 最常见的如Page Fault、除0问题。
- 指令会在异常返回后，尝试重新执行。
- 错误码会压在栈上，需要iret前手工去除掉。

Fault的栈



Trap

- 最常用的就是软件中断int n，主要用于实现系统调用。

系统调用

- 系统调用（System Call）是OS最核心的概念之一。

- 硬件中断是外部硬件设备与CPU交互的接口。

- 而系统调用（int n）是应用程序与OS内核交互的接口。

系统调用的实现

- 想象你在C程序中，会通过fwrite调用文件写操作。fwrite是C标准库实现的对系统函数write的一个封装。

- write本身也可以直接使用，且核心系统软件都是通过write调用。write内部又会嵌入汇编，比如write(...) {asm_code(mov eax, 5; int 0x80;)}这种方式。

int 0x80

- int 0x80又是怎么回事呢？这就是系统调用的总入口。在linux的实现中，通过eax保存系统调用的编号，比如fork的编号是2。open、write、close都有对应的编号。
- 根据参数的多少，使用寄存器传递调用参数或者使用栈。
- 某种意义上，实现OS的本质就是实现一组系统调用。

int 0x80

- Linus在Just for fun自传中，就描述一段经历，在不断的找手册，实现几十个系统调用。特别是在移植unix程序时，如果某个系统调用不存在，就在屏幕上打印出来，他就知道必须要实现它了。

- Posix：Portable Operating System Interface of Unix，关键就是定义了操作系统的系统调用接口，要求按照这一标准。所以你会听说某某系统实现了Posix标准。

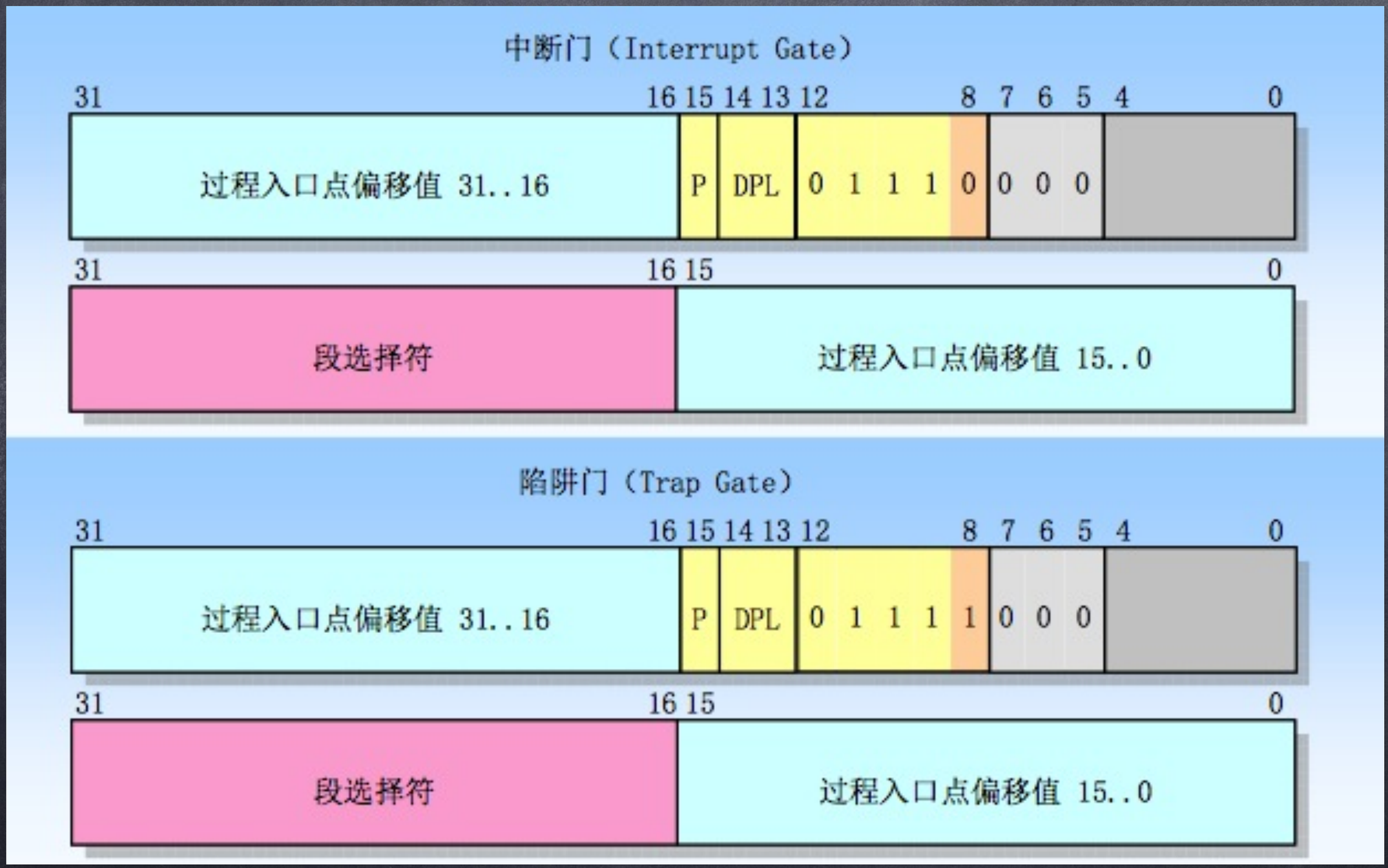
第一个实验

- 我们写一个简单的系统调用，在屏幕的bh行，bl列，打印ax的二进制数值，ch存放有显示属性。

关键代码1

```
39 ;二进制打印中断
40 mov eax, 0x00080000 ;段选择符0x08
41 lea edx, [int_print_binary_16]
42 mov ax, dx
43
44 mov edx, 0xef00 ;P DPL ...
45 mov ecx, 0x81 ;
46 lea edi, [idt + ecx * 8]
47 mov [edi], eax
48 mov [edi + 4], edx
```

中断门 vs 陷阱门


中断门（Interrupt Gate）
31
16 15 14 13 12
8 7 6 5 4
0
过程入口点偏移值 31..16
P
DPL
0 1 1 1 0
0 0 0
31
16 15
0
段选择符
过程入口点偏移值 15..0
陷阱门（Trap Gate）
31
16 15 14 13 12
8 7 6 5 4
0
过程入口点偏移值 31..16
P
DPL
0 1 1 1 1
0 0 0
31
16 15
0
段选择符
过程入口点偏移值 15..0

关键代码2

```
92 mov ax, 0x1234
93 mov bh, 20
94 mov bl, 10
95 mov ch, 0x02
96 int 0x81
97
98 LOOP1:
99     jmp LOOP1 ;程序在这里就结束了
```

关键代码3

```
105 align 4
106 int_print_binary_16:
107     call print_binary
108     iret
109
110 ;打印一个整数ax到bh行，bl列，颜色信息存放在ch
111 print_binary:
112     push eax
113     push ebx
114     push ecx
115     push edx
116     push gs
117     push si
118     push di
```

关键代码4

```
120         ;备份
121         mov di, ax
122         mov edx, SCREEN_SEL
123         mov gs, dx
124
125         mov si, 0 ;共16位，从左向右打印
126 repeat_pos:
127         mov ax, di ;还原ax
128
129         mov dx, si
130         mov cl, dl
131         shl ax, cl
132         shr ax, 15 ;移到最右侧1位
133
134         ;计算屏幕位置，bh*80+bl+si
135         push eax
136         push ebx
```

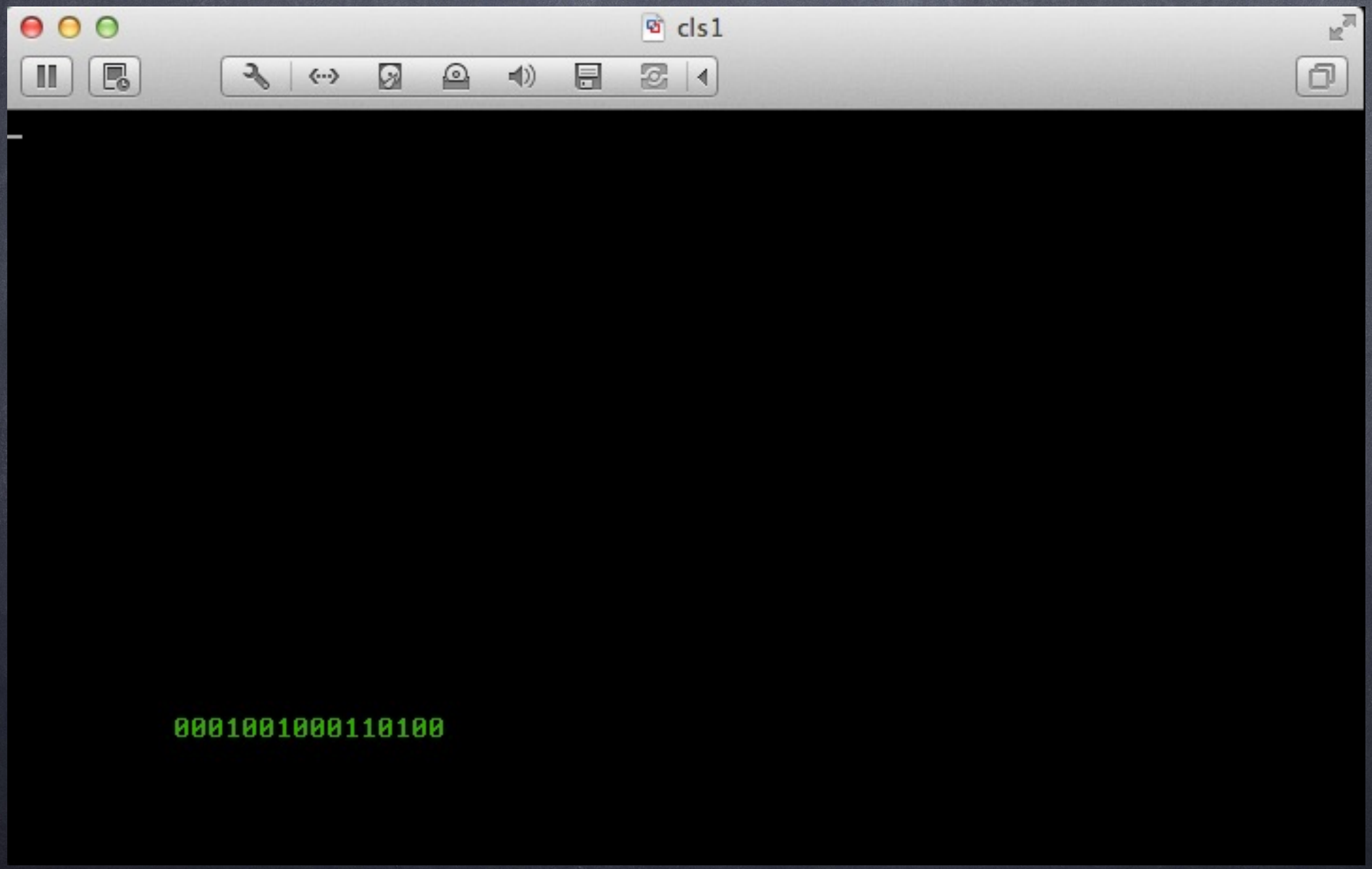
关键代码5

```
138         mov al, 80
139         mul bh
140         mov dx, ax
141
142         mov bh, 0
143         add dx, bx ; add bl
144         add dx, si ; dx存放位置
145
146         pop ebx
147         pop eax
148
149         ;print
150         shl edx, 1
151         add al, '0';加上asc偏移
152         mov [gs: edx], al
153         mov [gs: edx + 1], ch
154
155
156         inc si
157         cmp si, 16
158         jb repeat_pos
```

关键代码6

```
160         pop di
161         pop si
162         pop gs
163         pop edx
164         pop ecx
165         pop ebx
166         pop eax
167         ret
```

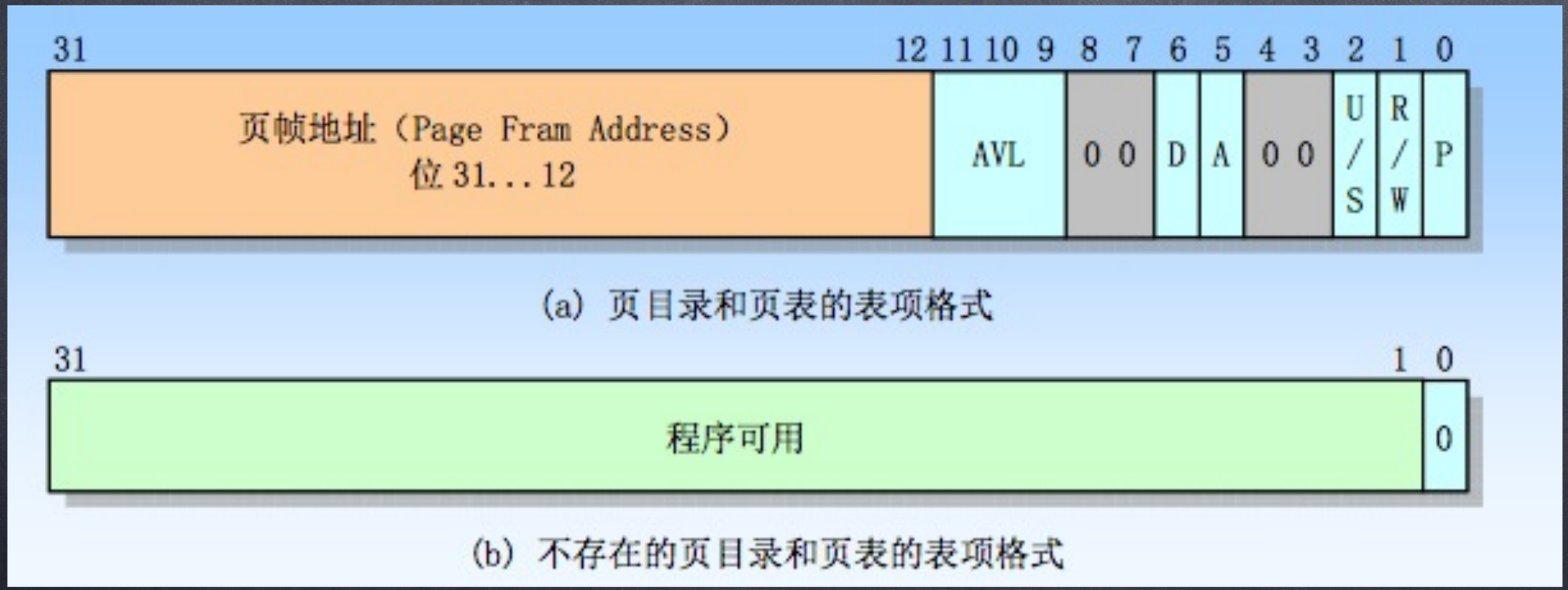
运行结果



Page Fault

- 第三节课的时候，我们讲了分页，但是并没有讲访问的页面不存在时，怎么去处理。

回顾页表


31
12 11 10 9 8 7 6 5 4 3 2 1 0
页帧地址（Page Fram Address）
位31...12
AVL
0 0
D
A
0 0
U / S
R / W
P
(a) 页目录和页表的表项格式
31
1 0
程序可用
0
(b) 不存在的页目录和页表的表项格式

Page Fault

- 当访问的页表项的P = 0时，就会产生0x0e号异常。并在CR2中，保存有引起缺页异常的线性地址。

Page Fault

- 我们可以给0x0e号中断，添加处理程序。将对应的物理页号设置到页表项，iret返回之前，再手工将错误码弹出。

第二个实验

- 我们只将前2M物理内存的页表映射好，当访问2M~4M之间的地址时，在缺页异常中，将对应的物理页码映射上去。打印输出写入的字符到屏幕。

遗憾的事

- 我自己的程序还没有调通，不能进行效果掩饰了。这个留作本节课的作业。

思考

- 我们给中断向量表初始化的默认处理程序int_ignore对哪类异常搞不定？为什么？

谢谢！