第2课 从实模式到保护模式

《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

- 我们在实验2中，我们使用jmp loop1，可以用jmp BOOTSEG: loop1吗，这有什么差异?
- 为什么现代操作系统中，不使用BIOS提供的硬件访问接口，而是重新实现一套？

回顾实模式

- 地址换算：段选择符 * 16 + 偏移地址
- 最多寻址20位，即1M访问空间
- 思考：这种模式有什么问题？

实模式的问题

- 能访问的内存太小，只有1M
- 缺乏保护，程序间可以互相操作，不安全
  - 如：0x1001：0x0010和0x1000：0x0020访问的地址一样。

保护模式的解决之道

- 地址变为32位，EIP、EAX、ESP。。。
- 复用段寄存器（CS、DS、SS。。。）
- 但表达的含义变了，成了一个段选择符（索引）。
- 从哪里选择呢？

GDT

- Global Descriptor Table：全局描述符表

- 段的详细信息在GDT中记录，包括：段的起始地址，限长，访问权限控制等。
- 就是一个大数组，每一条就是一个段描述符。

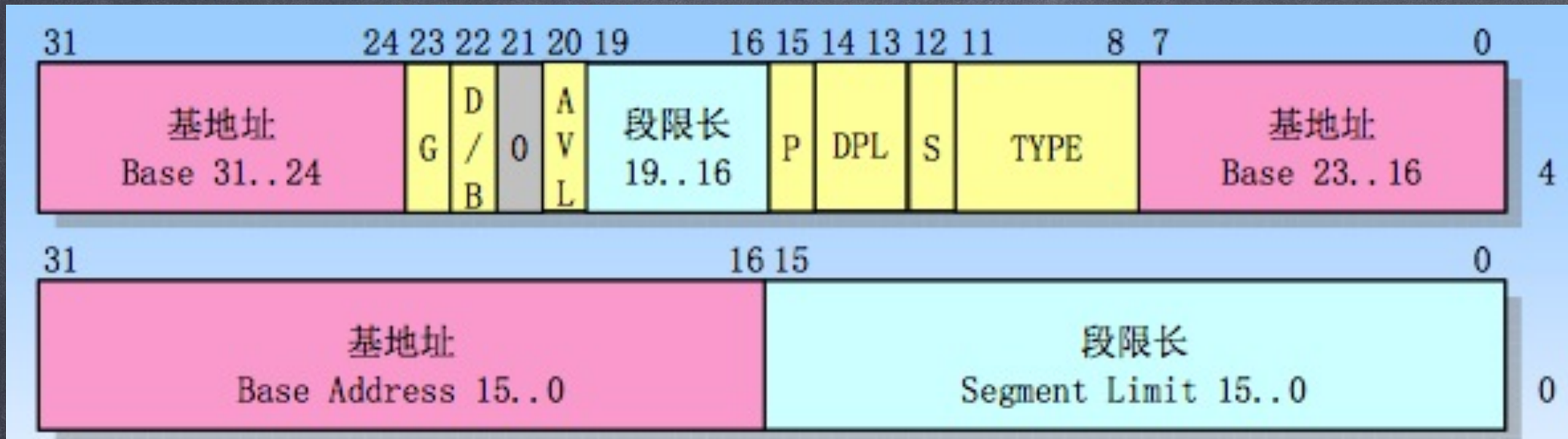
插入内容：字节序

- 对于超过一个字节的数据，我们先放哪一头到低地址？如0x12345678，存放在其实地址0x00000000中会是怎么样的？
  - A：0x12 0x34 0x56 0x78
  - B：0x78 0x56 0x34 0x12
- 注意：字节是内存寻址的最小单位。而不是bit，或者一个16进制数的1位。

插入内容：字节序

- 英特尔坚持低地址放不重要的数（0x78）
- 网络传输坚持先传输重要的数（0x12）
- Little-Endian VS Big-Endian
  - 来源于《格列佛游记》
  - Endian：“端”

段描述符通用格式



AVL -- 系统软件可用位

BASE -- 段基地址

B/D -- 默认大小(0-16 位;1-32 位段)

DPL -- 描述符特权级

G -- 颗粒度

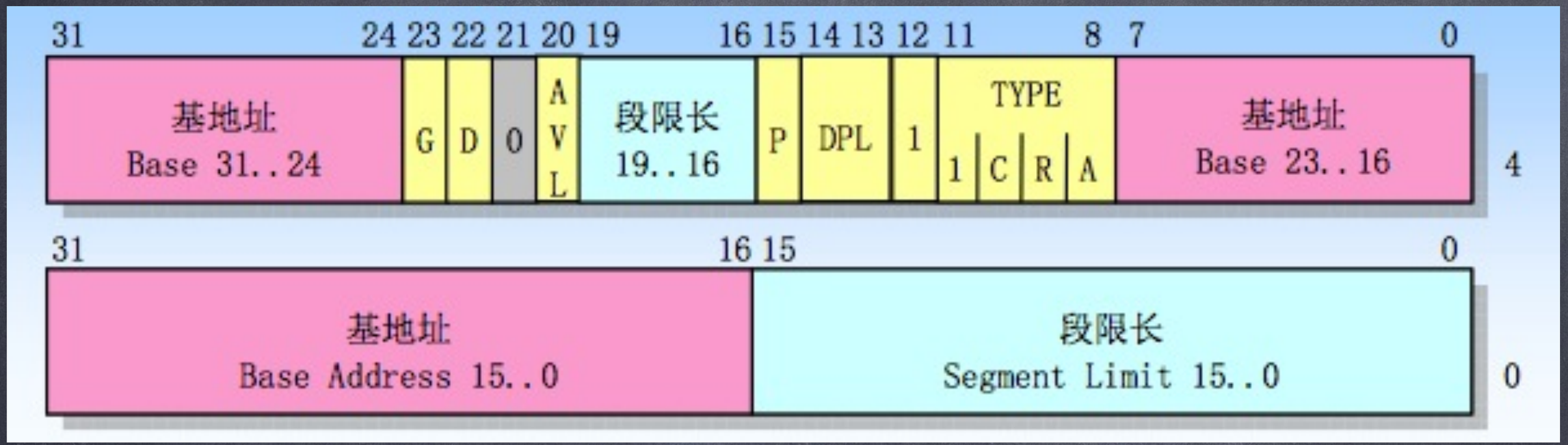
LIMIT -- 段限长

P -- 段存在

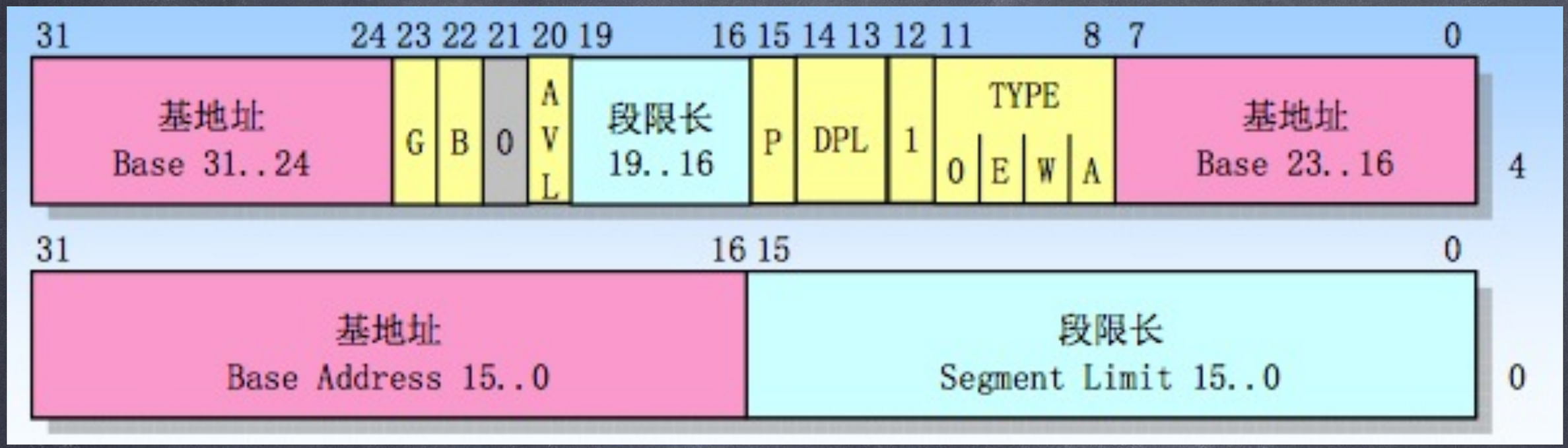
S -- 描述符类型(0-系统;1-代码或数据)

TYPE -- 段类型

代码段

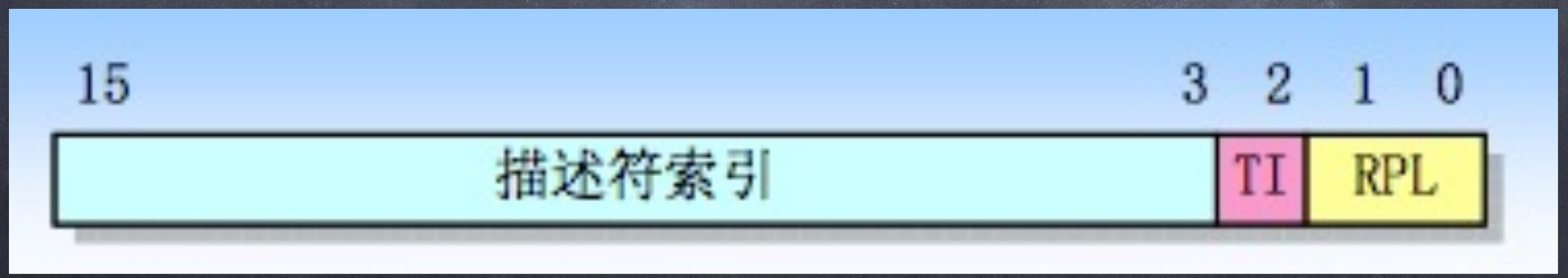

31
24
23
22
21
20
19
16
15
14
13
12
11
8
7
0
基地址
Base 31..24
G
D
0
AVL
段限长
19..16
P
DPL
1
TYPE
1
C
R
A
基地址
Base 23..16
4
31
16
15
0
基地址
Base Address 15..0
段限长
Segment Limit 15..0
0

数据段


31
24 23 22 21 20 19
16 15 14 13 12 11
8 7
0
基地址
Base 31..24
G
B
0
AVL
段限长
19..16
P
DPL
1
TYPE
0
E
W
A
基地址
Base 23..16
4
31
16 15
0
基地址
Base Address 15..0
段限长
Segment Limit 15..0
0

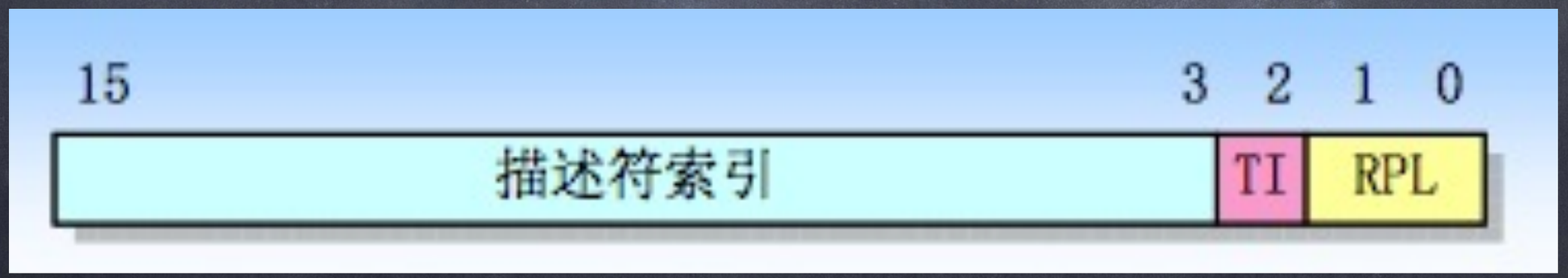
段选择符

- 共包括三个部分：
  - GDT/LDT表索引号
  - TI：0表示GDT，1表示LDT（暂时不用）
  - RPL：请求特权级

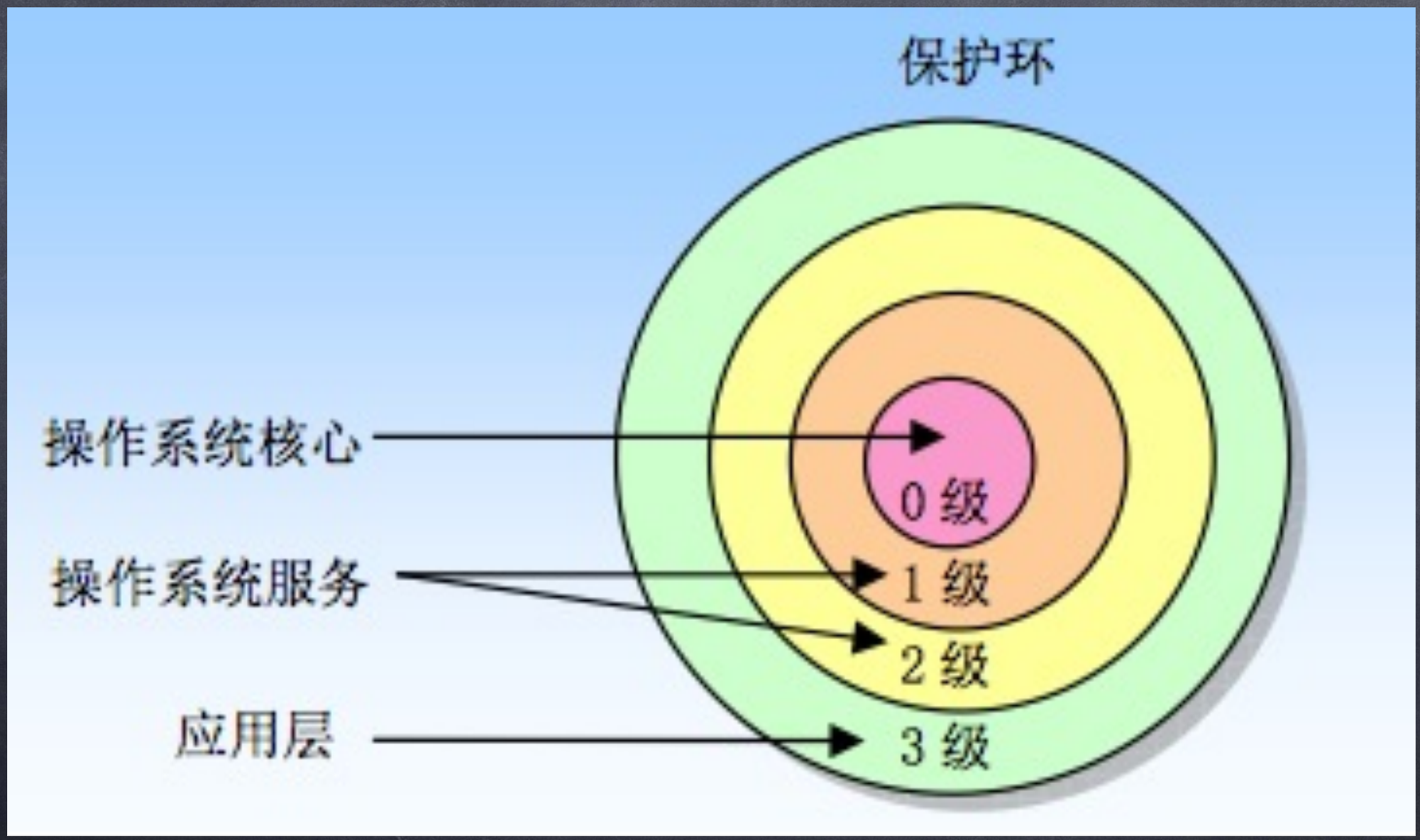


段选择符

- 例子：
  - 全局描述符表中的第1项：00001 0 00 = 0x08
  - 第2项：00010 0 00 = 0x10



特权级



CPL/RPL/DPL

- CPL：当前特权级（Current Privilege Level），存放于CS的低2位中。

- RPL：请求特权级（Requested P L），jmp调用的描述符的低2位中。

- DPL：描述符特权级（Descriptor P L），存放于段描述符中。

CPL/RPL/DPL

- 20岁的亚里沙到了谈婚论嫁的年纪，她可以号称自己是20/30/40，去找一个40岁的钻石王老五。
- 警察叔叔可能会管。
- 王老五可能喜欢越小越好的，也可能只喜欢和自己年龄一致的。

CPL/RPL/DPL

- 我们暂时忽略各种组合的可能，等到用的时候再看。
- 在出现时，都将这三者设为0，即最高权限级别，跳过这一问题。
- 做个baby os，没必要考虑那么多复杂的情况。

如何进入保护模式

- 1，设置GDT描述符表，通过LGDT加载表。
- 2，将寄存器CR0的PE位设置为1，进入保护模式。
- 3，紧跟一条jmp指令，使用一个段选择符，跳转到保护模式下的有效代码段。
- 4，进入保护模式。

第一个实验

- 保护模式下，在屏幕打印一个字符串Hello, world!

代码（boot.s）

```
1  BOOTSEG equ 0x07C0 ;宏定义，用于初始化ES
2
3  ;使用bios中断0x13，功能号(AH = 2)进行软盘读取
4  mov dx, 0x0000 ;DH = 0，磁头号
5                 ;DL = 0，驱动器号
6  mov cx, 0x0002 ;CH，10位磁道号低8位
7                 ;CL - 位7、6是磁道号高2位
8                 ;CL - 位5~0表示扇区号（从1开始）
9                 ;本指令表示读取0号驱动器0号磁道第2号扇区
10                ;第1扇区就是boot扇区
11 mov ax, 0x1000 ;设置拷贝的目的地段描述符为0x1000
12 mov es, ax
13 xor bx, bx     ;设置bx为0，[es:bx]表示目的地内存地址
14                ;即0x1000 * 16 + 0 = 0x10000
15 mov al, 1      ;读取的扇区数为1
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
```

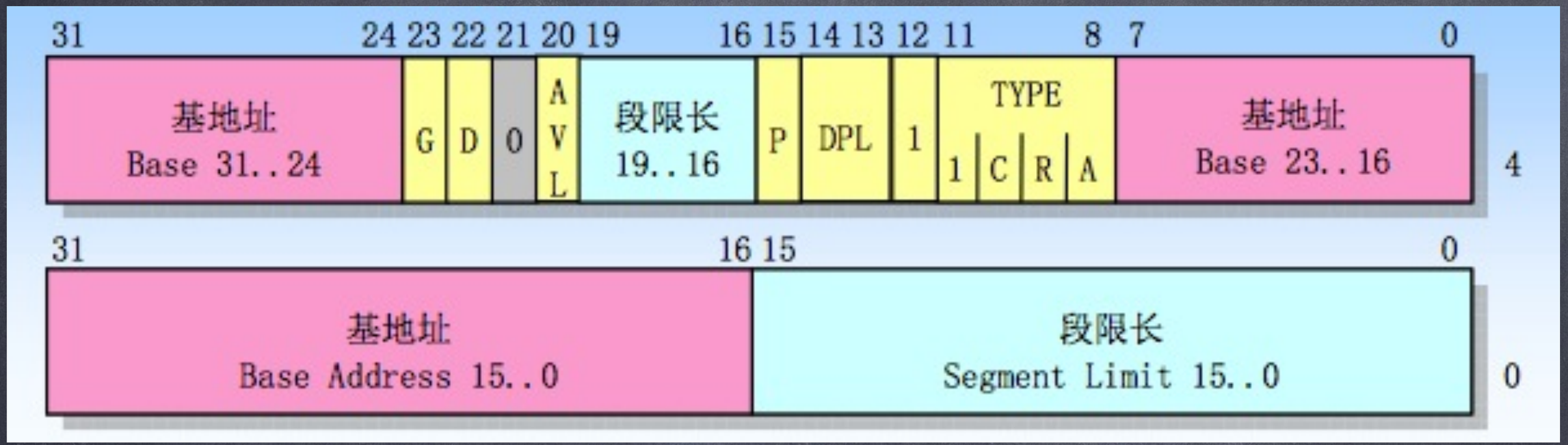
代码（boot.s）

```
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, 128 ;移动128个double word(4 bytes)
26 sub si, si
27 sub di, di
28 rep movsd
29
30 mov ax, BOOTSEG
31 mov ds, ax ;要访问内存数据，则需设置数据段
32 lidt [idt_48] ;加载中断表，请忽略，以后会讲
33 lgdt [gdt_48] ;加载GDT描述符
34
35
36 mov eax, cr0
37 or al, 1 ;将保护模式的标记PE设置为1
38 mov cr0, eax
```

代码（boot.s）

```
40 ;这里加dword是为了生成一个32位偏移地址的指令
41 ;这条指令执行的时候，已经是在保护模式了
42 ;是利用了CPU预取的机制，在执行mov cr0, eax时，
43 ;就将其读入了指令寄存器
44 ;0x0008表示GDT中第一个描述符
45 ;0x00000000是跳转的32位偏移地址，有效的
46 jmp dword 0x0008: 0x00000000
```

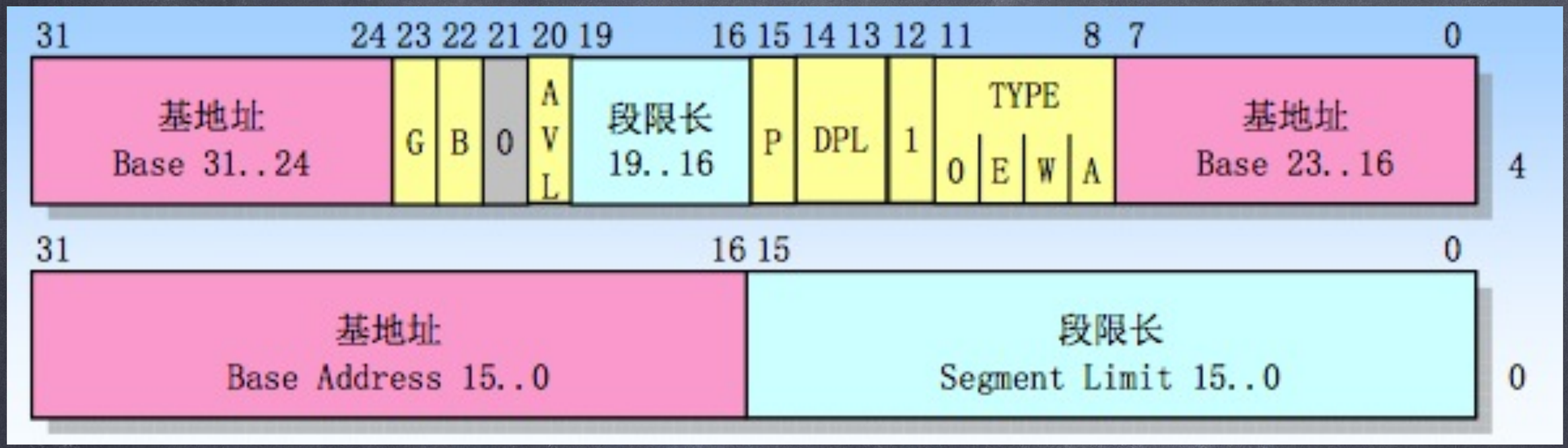
代码段


31
24
23
22
21
20
19
16
15
14
13
12
11
8
7
0
基地址
Base 31..24
G
D
0
AVL
段限长
19..16
P
DPL
1
TYPE
1
C
R
A
基地址
Base 23..16
4
31
16
15
0
基地址
Base Address 15..0
段限长
Segment Limit 15..0
0

代码（boot.s）

```
48 ;注意：以下GDT的设置一定要对照格式图表
49 gdt:
50      dw 0, 0, 0, 0 ;第0个描述符，不使用
51
52      ;第一个描述符(0x08)，表示代码段
53      dw 0x07FF ;段限长低16位，则段限长为：
54                ;(0x07FF + 1) * 4KB = 8MB
55      dw 0x0000 ;基地址低16位0，所以基地址为0
56      dw 0x9A00 ;0x9A = 1001 1010
57                ;高4位分别表示P = 1段在内存中
58                ;DPL = 0，最高权限
59                ;S = 1，非系统段
60                ;且TYPE的bit 3为1，表示代码段
61                ;低8位表示基地址23~16为0
62      dw 0x00C0 ;高8位表示基地址31~24为0
63                ;G标志(bit 7) = 1，表示颗粒度为4KB
64                ;注意：保护模式下颗粒度一般都设4KB
65                ;最后4位为0，表示段限长19~16位是0
```

数据段


31
24 23 22 21 20 19
16 15 14 13 12 11
8 7
0
基地址
Base 31..24
G
B
0
AVL
段限长
19..16
P
DPL
1
TYPE
0
E
W
A
基地址
Base 23..16
4
31
16 15
0
基地址
Base Address 15..0
段限长
Segment Limit 15..0
0

代码（boot.s）

```
67      ;第2个段描述符（0x10），表示显卡内存
68      ;显卡默认在CGA模式，字符彩屏
69      ;映射在内存地址0xb8000~0xc8000，共16K
70      ;我们只用前面的4K
71      ;所以颗粒度G = 1，且段限长Limit = 0
72      dw 0x0002
73      dw 0x8000
74      dw 0x920B ;0x92 = 1001 0010
75                ;S = 1且TYPE(bit 3)为0，表示数据段
76                ;DPL = 0
77                ;E = 0，表示地址是从小往大增长
78                ;W = 1表示可写入
79      dw 0x00C0
80
81  end_gdt:
82
83  idt_48:
84      dw 0
85      dw 0
86      dw 0
```

代码（boot.s）

```
88 gdt_48:
89       dw (end_gdt - gdt) - 1 ;gdt的限长
90                              ;-1的作用和段描述符的Limit类似
91                              ;都是表示最后一个有效的地址(<=)
92       dw BOOTSEG * 16 + gdt ;gdt的起始地址
93       dw 0
94
95 times 510 - ($ - $$) db 0 ;填充一堆的0
96                           ;$表示当前位置，$$表示文件头部
97 db 0x55
98 db 0xAA ;上面两行用于设置引导扇区的标识
```

代码

(printhello_pe.s)

```
1  KERNEL_SEL equ 0x08
2  SCREEN_SEL equ 0x10
3
4  bits 32 ;这是32位的指令
5  mov eax, SCREEN_SEL
6  mov ds, eax
7
8  mov eax, 0
9  mov byte [eax], 'H'
10 mov byte [eax + 1], 0x02 ;color
11 add eax, 2
12 mov byte [eax], 'e'
13 mov byte [eax + 1], 0x02 ;color
14 add eax, 2
15 mov byte [eax], 'l'
16 mov byte [eax + 1], 0x02 ;color
17 add eax, 2
18 mov byte [eax], 'l'
19 mov byte [eax + 1], 0x02 ;color
20 add eax, 2
```

代码

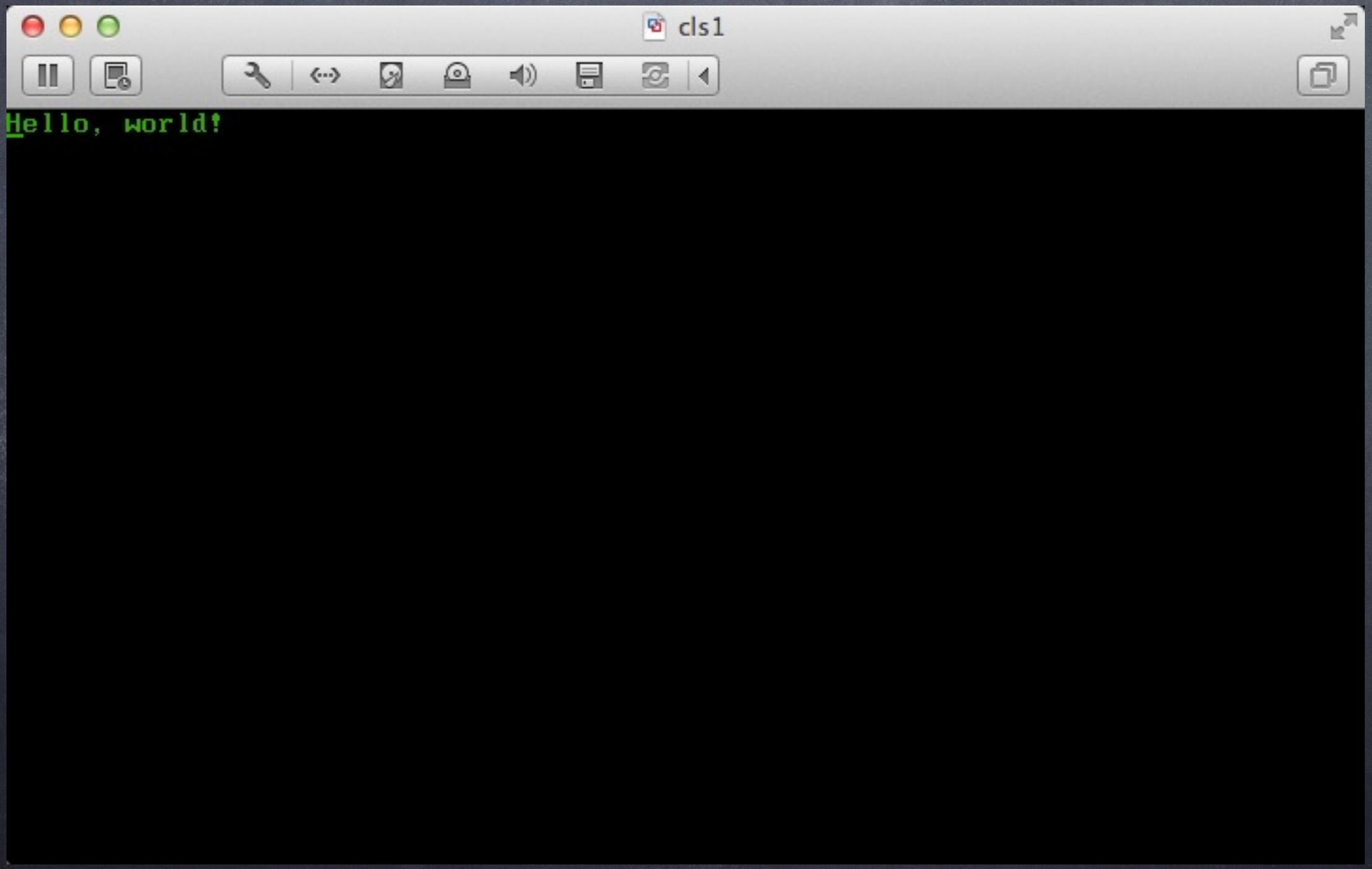
(printhello_pe.s)

```
36 mov byte [eax], 'r'
37 mov byte [eax + 1], 0x02 ;color
38 add eax, 2
39 mov byte [eax], 'l'
40 mov byte [eax + 1], 0x02 ;color
41 add eax, 2
42 mov byte [eax], 'd'
43 mov byte [eax + 1], 0x02 ;color
44 add eax, 2
45 mov byte [eax], '!'
46 mov byte [eax + 1], 0x02 ;color
47
48 LOOP1:
49     jmp LOOP1
```

生成软盘镜像

- 汇编:
  - nasm -f bin boot.s -o boot.bin
  - nasm -f bin printhello_pe.s -o printhello.bin
- 生成镜像：
  - cat boot.bin printhello.bin > system.bin
  - dd conv=sync if=system.bin of=helloboot.img bs=1440k count=1

运行结果



实验总结

- 这里已经牵涉到了程序的“链接”。

思考

- 在实验中，我们是将第2扇区先加载到0x10000，然后再复制到0x000000的，为什么不直接加载到0x000000？

扩展阅读

- 深入浅出保护模式：http://hi.baidu.com/sangwf/item/7093d43e755b544a3075a193
- 从实模式到保护模式那令人激动的一跳：http://hi.baidu.com/sangwf/item/dc50b73f11bf7ec02e8ec29c
- 数据段描述符中的E、D/B、G及Limit四者的关系：

  http://hi.baidu.com/sangwf/item/782319e6a78c66f6e0a5d4bb

谢谢！