LISS
信息安全研究所

# 操作系统设计及实践

《操作系统原理》配套实验

信安系操作系统课程组

2024年9月

操作系统课程实验系列（二）

# 保护模式工作机理

武漢大学

# 一、实验目标

1. 理解x86架构下的段式内存管理
2. 掌握实模式和保护模式下段式寻址的组织方式、关键数据结构、代码组织方式
3. 掌握实模式与保护模式的切换
4. 掌握特权级的概念，以及不同特权之间的转移
5. 了解调用门、任务门的基本概念

# 二、本次实验内容步骤

1. 认真阅读章节资料，掌握什么是保护模式，弄清关键数据结构：GDT、descriptor、selector、GDTR， 及其之间关系，阅读pm.inc文件中数据结构以及含义，写出对宏Descriptor的分析
2. 调试代码，/a/ 掌握从实模式到保护模式的基本方法，画出代码流程图，特别注意跳转问题，如果把跳转直接改成jmp offset，而不用selector:offset形式，会是什么结果，反汇编比较一下区别。
3. 调试代码，/b/，掌握GDT的构造与切换，从保护模式切换回实模式方法
4. 调试代码，/c/，掌握LDT切换
5. 调试代码，/d/掌握一致代码段、非一致代码段、数据段的权限访问规则，掌握CPL、DPL、RPL之间关系，以及段间切换的基本方法
6. 调试代码，/e/掌握利用调用门进行特权级变换的转移的基本方法

# 三、实验解决问题与动手改

1. GDT、Descriptor、Selector、GDTR结构，及其含义是什么？他们的关联关系如何？pm.inc所定义的宏怎么使用？
2. 从实模式到保护模式，关键步骤有哪些？为什么要关中断？为什么要打开A20地址线？从保护模式切换回实模式，又需要哪些步骤？
3. 解释不同权限代码的切换原理，call, jmp，retf使用场景如何，能够互换吗？
4. 动手改：
   - ① 自定义添加1个GDT代码段、1个LDT代码段，GDT段内要对一个内存数据结构写入一段字符串，然后LDT段内代码段功能为读取并打印该GDT的内容；
   - ② 自定义2个GDT代码段A、B，分属于不同特权级，功能自定义，要求实现A-->B的跳转，以及B-->A的跳转。

# 四、需了解的知识点

1. x86 CPU的基本模式：实模式、保护模式
   - 实模式
     - 地址总线宽度：20bit
     - 寄存器和数据总线宽度：16bit
     - 寻址空间是多少？
     - 实模式：PA=Segment*16+Offset

# 四、需了解的知识

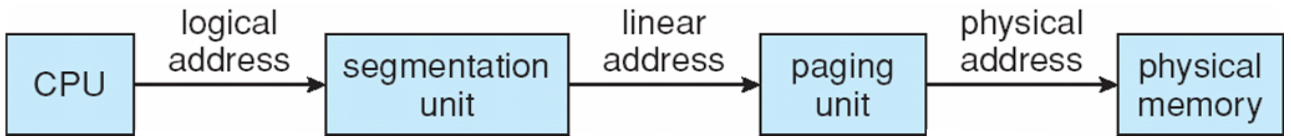
1. x86 CPU的基本模式：实模式、保护模式
   - 保护模式——回顾上学期知识
     - IA32的段寄存器
       - CS：保存代码段选择符
       - DS：保存数据段选择符
       - SS：保存堆栈段选择符
       - CR0：保存分页标志PG、保护允许标志PE等
       - CR2：保存缺页中断时，所缺页的线性地址
       - CR3：保存页目录表基址

# 四、需了解的知识

- 逻辑地址由<Selector, offset>组成
- Selector由三部分组成
  - S：段号
  - TI：GDT or LDT?
  - RPL：段描述符特权 DPL

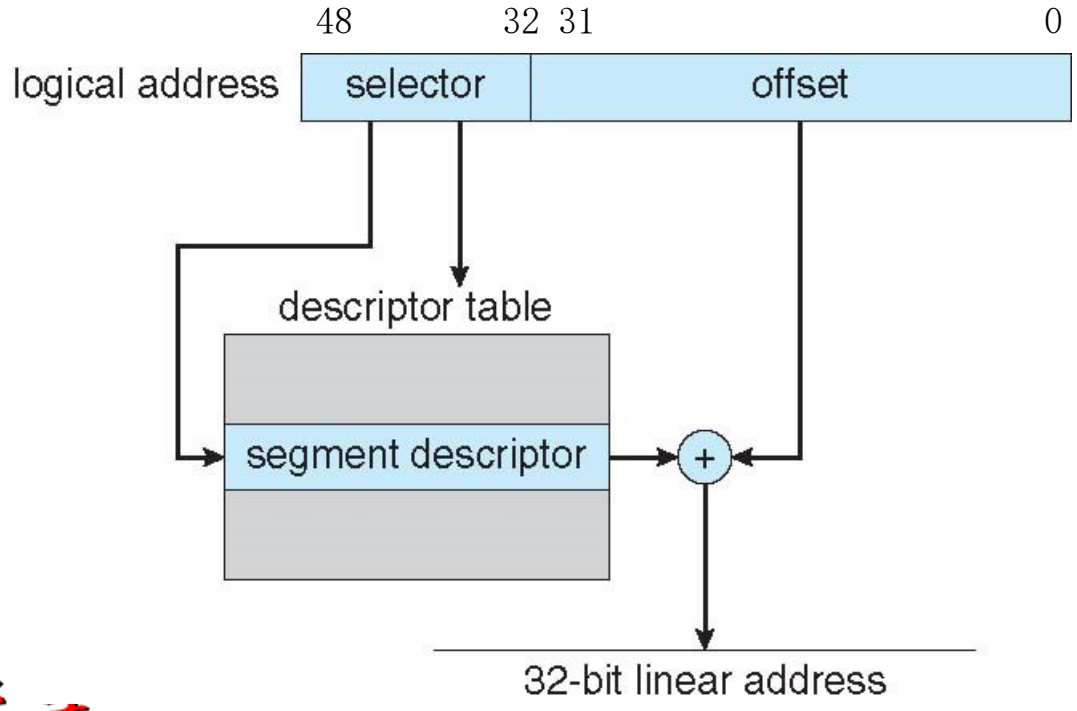
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 描述符索引 | | | | | | | | | | | | | TI | RPL | |

- CPU产生逻辑地址
- 选择子Selector被送到分段单元产生线性地址
- 线性地址被送到分页单元产生物理地址



# 四、需了解的知识

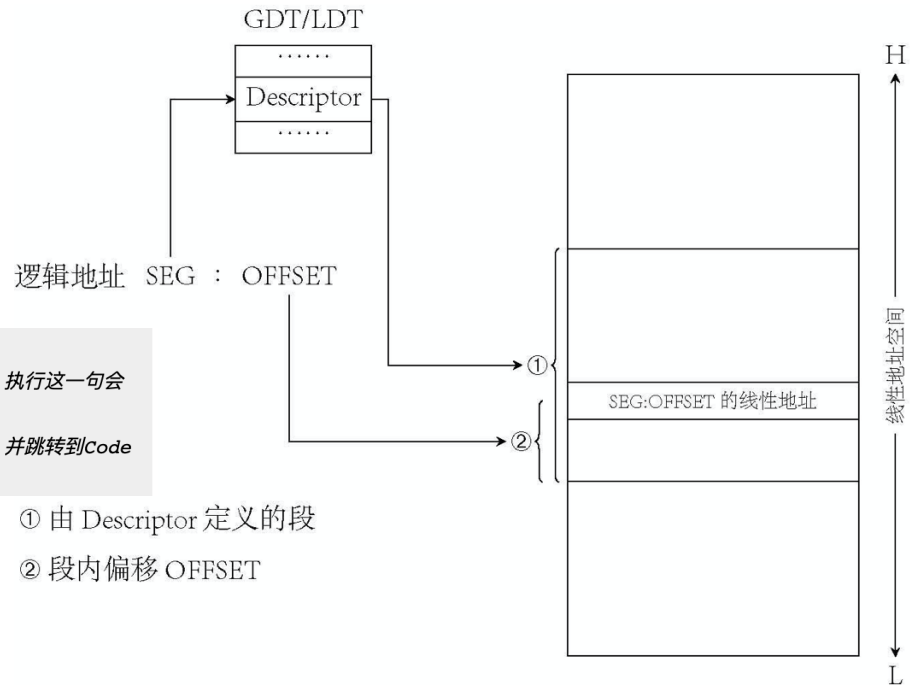
- 逻辑地址→线性地址
  - 选择子Selector被送到分段单元
  - 分段单元产生线性地址



# 四、需了解的知识

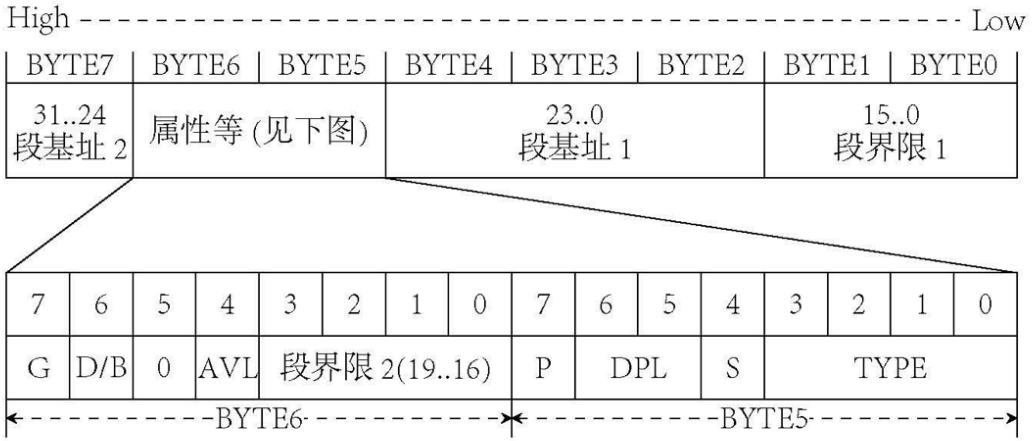
1. x86 CPU的基本模式：实模式、保护模式
   - 保护模式
     - 段描述符

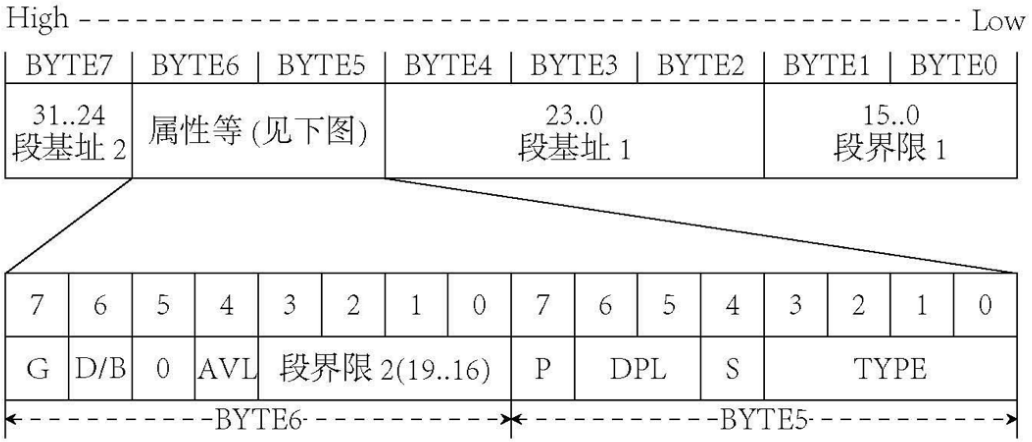
```
70          ; 真正进入保护模式
71          jmp         dword SelectorCode32:0    ; 执行这一句会
把SelectorCode32 装入cs,
72                                                ; 并跳转到Code
32Selector:0    处
```



# 四、需了解的知识

- 代码段、数据段段描述符
  - 描述符表中的描述符是存储管理硬件MMU管理虚存空间分段的依据。
  - 一个描述符直接对应于虚存空间中的一个主存分段，定义段的基址、大小和属性 。8字节






- 段基址：32位，三部分组成
- 段限长：20位，两部分组成
- P：P=1,该段在内存中；否则P=0，该段不在内存中，会引发异常
- DPL：描述符特权级(Descriptor Privilege level)，用于段访问时的特权检查
- S：段内容标志，S=1，代码和数据段描述符；S=0，系统段描述符/或门描述符
- TYPE：段类型和保护方式，如可执行数据段、只读数据段等
- G：段界限粒度，G=0，以字节为单位，G=1以页面4K为单位，故段长为220B或者220*4KB=4G
- D：多种条件下，语义不同，D=1 32位方式访问、D=0 16位方式访问，
- AVL：软件可利用位，保留。

## GDT例子

```
11  [SECTION .gdt]
12  ; GDT
13  ;                                段基址,      段界限       ,
属性
14  LABEL_GDT:          Descriptor       0,                0,
0           ; 空描述符
15  LABEL_DESC_CODE32:  Descriptor       0, SegCode32Len - 1,
DA_C + DA_32; 非一致代码段
16  LABEL_DESC_VIDEO:   Descriptor 0B8000h,           0ffffh,
DA_DRW      ; 显存首地址
17  ; GDT 结束
18
19  GdtLen              equ              $ - LABEL_GDT       ;
 GDT长度
20  GdtPtr              dw               GdtLen - 1          ;
 GDT界限
21                      dd               0                   ;
 GDT基地址
22
23  ; GDT 选择子
24  SelectorCode32                       equ    LABEL_DESC_COD
E32           - LABEL_GDT
25  SelectorVideo                        equ    LABEL_DESC_VID
EO            - LABEL_GDT
26  ; END of [SECTION .gdt]
```

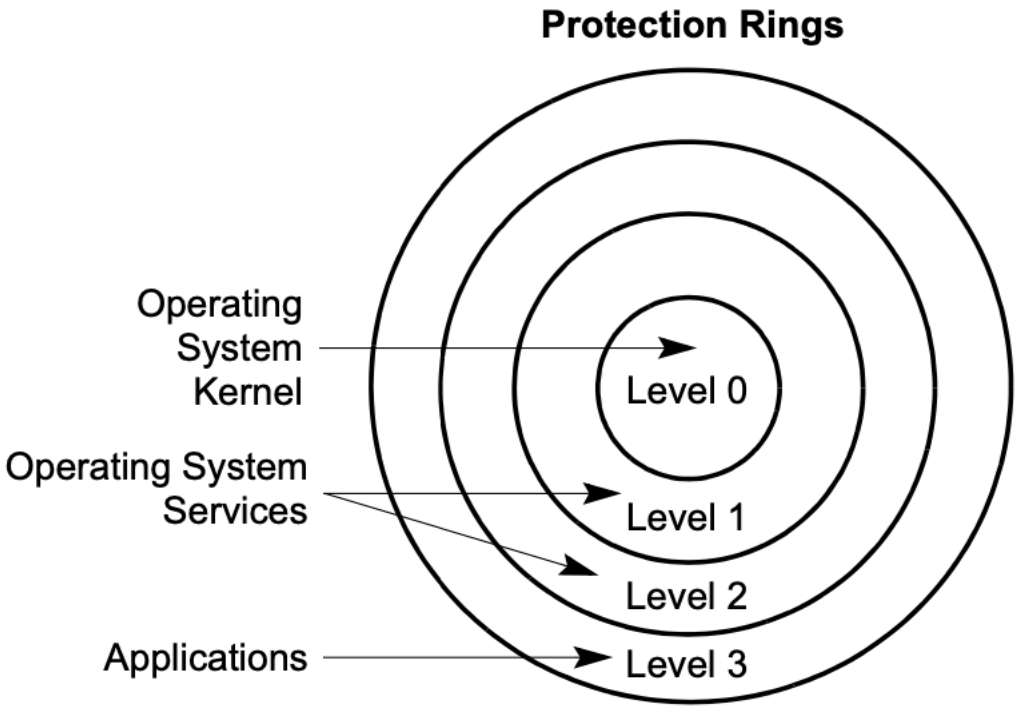
## GDT初始化

```
37          ; 初始化 32 位代码段描述符
38          xor         eax, eax
39          mov         ax, cs
40          shl         eax, 4
41          add         eax, LABEL_SEG_CODE32
42          mov         word [LABEL_DESC_CODE32 + 2], ax
43          shr         eax, 16
44          mov         byte [LABEL_DESC_CODE32 + 4], al
45          mov         byte [LABEL_DESC_CODE32 + 7], ah
46
47          ; 为加载GDTR 作准备
48          xor         eax, eax
49          mov         ax, ds
50          shl         eax, 4
51          add         eax, LABEL_GDT          ; eax <- gdt
基地址
52          mov         dword [GdtPtr + 2], eax ; [GdtPtr + 2]
 <- gdt 基地址
53
54          ; 加载  GDTR
55          lgdt        [GdtPtr]
56
```

```
76  [SECTION .s32];     32 位代码段. 由实模式跳入.
77  [BITS 32]
78
79  LABEL_SEG_CODE32:
```

# 四、需了解的知识

2. 环保护



# 四、需了解的知识

2. 环保护扩展
   - CPL
     - 描述当前执行程序或者任务的特权级。存储在CS和SS的bit 0，bit 1。当程序在不同特权级代码见转移，CPL会发生改变。
   - DPL
     - 描述段或者门的特权级，存储在描述符的DPL字段中，是这个段的特权级别，用来表明访问这个段时候，所需要的特权。

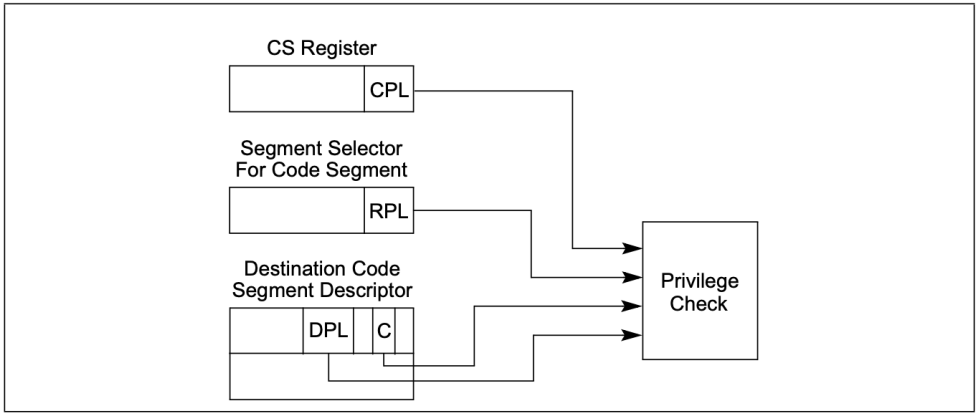
# 四、需了解的知识

2. 环保护扩展
   - RPL
     - 描述选择子的特权级，段选择子（Selector）的bit0和bit1，是可以重载的。例如：被调用的系统过程（CPL=0）从调用应用过程（CPL=3）收到一个选择子，就会把这个选择子的RPL设置成调用者的，从而表明当前是代表调用者的特权级在工作CPL=3，而不是CPL=0
   - **注意，我们后面讨论的CPL是当前运行代码的CPL，DPL是目标代码段/数据段描述符的DPL，RPL是指向目标代码的段选择子的RPL，选择子的RPL是可以被应用程序控制修改的。**

# 四、需了解的知识

## 3. 段的特权类型

- 一致性代码段：用于提供一些非敏感的系统功能
  - 允许低特权级的代码段（A），访问高特权级的代码段（B），即CPL-A>=DPL-B，此处RPL并不检查
  - 这里的高特权级代码不会访问敏感资源、也不会访问异常处理等系统代码，但可以是一个如某个纯粹的数学计算库
  - 一致性：当低特权级代码段，访问高特权级代码段时候，其CPL不发生变化，Why?
  - 不允许B代码段访问A，Why?



# 四、需了解的知识

## 3. 段的特权类型

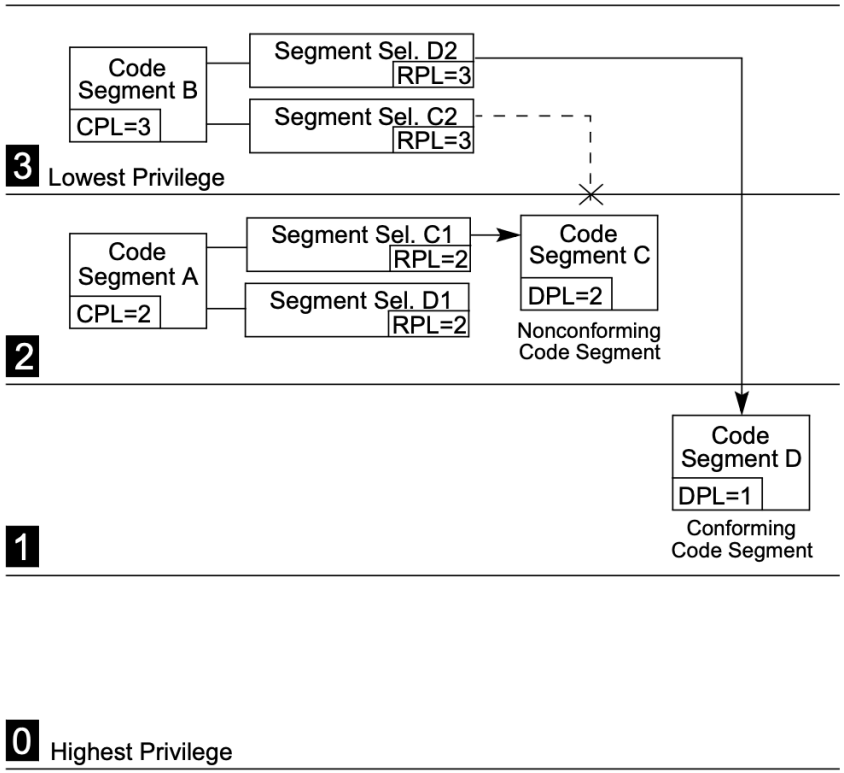
- 非一致性代码段：用于对系统敏感资源进行访问
  - 只允许同特权级的代码能够访问，
  - 不允许不同级间访问:if A<>B, A不能访问B，B不能访问A
  - 作用：为了避免低特权级代码访问被操作系统保护起来的系统代码
  - CPL-A=DPL-B，RPL-A<=DPL-B(WHY RPL如此？)
- 数据段：
  - 高特权级别代码可以访问低特权级别数据
  - 同特权级别代码可以访问同特权级别数据
  - 不允许低特权级别代码访问高特权级别数据
  - 确保数据的完整性，避免被破坏

# 四、需了解的知识

| | 特权级<br>低→高 | 特权级<br>高→低 | 相同权限级之间 |
|---|---|---|---|
| 一致代码段 | 允许 | 不允许 | 允许 |
| 非一致代码段 | 不允许 | 不允许 | 允许 |
| 数据段<br>（非一致） | 不允许 | 允许 | 允许 |

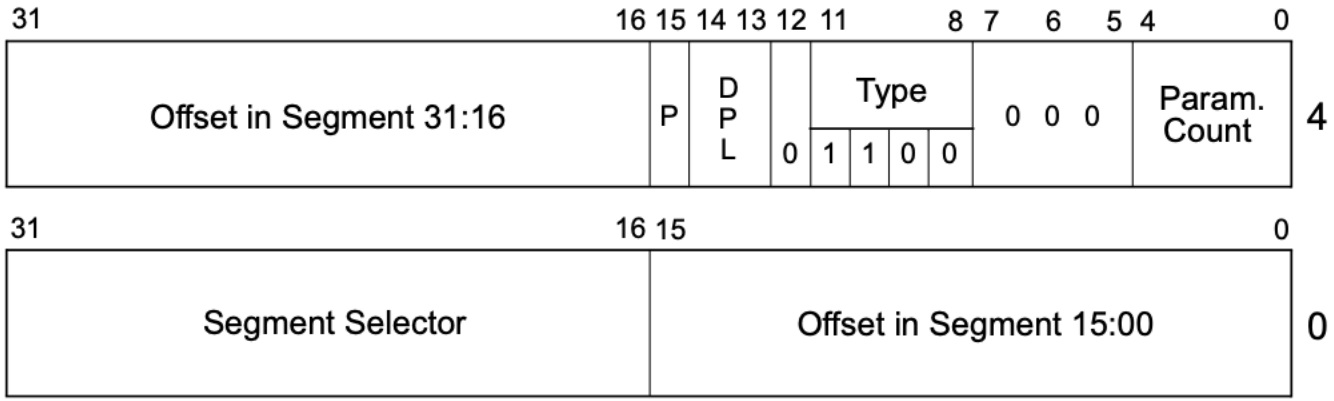
# 四、需了解的知识

## 3. 段的特权类型



# 四、需了解的知识

4. 不同特权级别段之间的代码转移
   - 一致代码段，直接访问，只能是CPL>=目标代码段的DPL
   - 非一致代码段，直接访问，只能同级CPL，且调用RPL<=目标DPL
   - 如何实现任意的呢？调用 CALL GATE




DPL Descriptor Privilege Level
P Gate Valid

# 四、需了解的知识

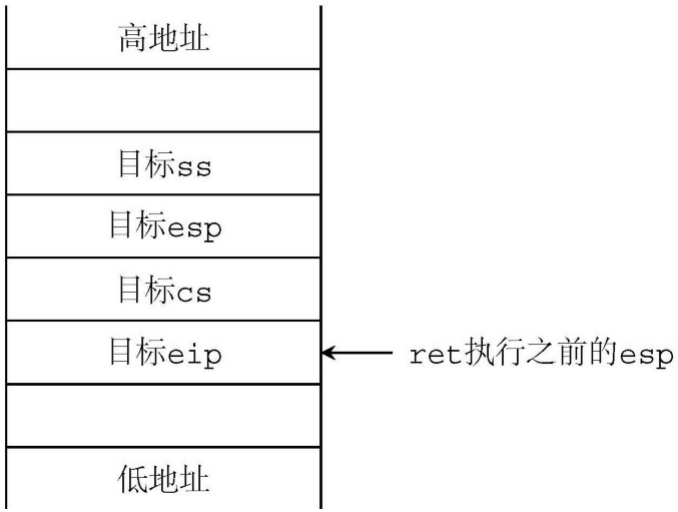
4. 不同特权级别段之间的代码转移
   - 本质上是一个添加了属性的特殊入口地址
   - 代码A→调用门G→代码B
   - CPL-A,RPL-A <= DPL_G
   - 如果B为一致代码段
     - CPL-A, RPL-A <= DPL_G
     - CPL-A>=DPL_B
     - 实现了从低特权代码A-->高特权代码B
   - 如果B为非一致代码段
     - CALL：CPL-A>=DPL_B
     - JMP:  CPL-A = DPL_B
   - 通过Call Gate + Call,实现了从低特权级-->高特权级的访问

| | call | jmp |
|---|---|---|
| 目标是一致代码段 | CPL⩽DPL_G，<br>RPL⩽DPL_G，<br>DPL_B⩽CPL | |
| 目标是非一致代码段 | CPL⩽DPL_G，<br>RPL⩽DPL_G，<br>DPL_B⩽CPL | CPL⩽DPL_G，<br>RPL⩽DPL_G，<br>DPL_B=CPL |

# 四、需了解的知识

5. 不同特权级代码之间切换，上下文如何恢复？
   - Task-State Segment

6. 如何实现高特权级→低特权级
   - 可采用return方法
   - 压栈顺序




| I/O 位图基址 | | T | 100 |
|---|---|---|---|
| | LDT 选择子 | | 96 |
| | gs | | 92 |
| | fs | | 88 |
| | ds | | 84 |
| | ss | | 80 |
| | cs | | 76 |
| | es | | 72 |
| edi | | | 68 |
| esi | | | 64 |
| ebp | | | 60 |
| esp | | | 56 |
| ebx | | | 52 |
| edx | | | 48 |
| ecx | | | 44 |
| eax | | | 40 |
| eflags | | | 36 |
| eip | | | 32 |
| gr3 (pdbr) | | | 28 |
| | ss2 | | 24 |
| esp2 | | | 20 |
| | ss1 | | 16 |
| esp1 | | | 12 |
| | ss0 | | 8 |
| esp0 | | | 4 |
| | 上一任务链接 | | 0 |

保留位，被设为 0。

# 谢　谢！