|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 操作系统设计与实践 | | | 成 绩 |  | 教师签名 |  |
| 实验名称 | 保护模式工作机理 | | | 实验序号 | 2 | 实验日期 |  |
| 姓 名 | 张子航 | 学 号 | 2021302181026 | | | | 组长 |
| 姓 名 | 辜汝曦 | 学 号 | 2021302141194 | | | | 组员 |
| 姓 名 | 杨馨悦 | 学 号 | 2021302181212 | | | | 组员 |
| 姓 名 | 赵敏 | 学 号 | 2021302181215 | | | | 组员 |

**《操作系统设计与实践》实验报告**

# 实验目的及实验内容

（本次实验所涉及并要求掌握的知识；实验内容等）

## 实验目的

结合前章，理解保护模式，以及段式寻址

## 参考资料

1. 《 Orange’s 一个操作系统的实现》对应章节：第三章3.1-3.2节

## 实验要求

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，如果代码/a/中，第71行有dword前缀和没有前缀，编译出来的代码有区别么，为什么，请调试截图。
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

# 实验环境及实验步骤

（列出本次实验所使用的软件、工具；简要概括实验步骤）

## 实验环境

虚拟机工具：VMWare Workstation 16

虚拟机版本：Ubuntu 14.04.6(内存4GB，硬盘40GB，双核处理器)

开发与调试工具：bochs 2.6.8

## 实验步骤

**首先，完成本次实验内容步骤：**

* 1. 了解保护模式，梳理保护模式中各个数据结构的作用；
  2. 调试代码/a/，掌握由实模式到保护模式的流程；
  3. 调试代码/b/，掌握GDT由保护模式到实模式的流程；
  4. 调试代码/c/，掌握LDT切换。

**然后，完成实验解决问题与课后动手改：**

1. 解决实模式到保护模式的流程与细节问题；
2. 通过cr0来判断cpu是实模式还是保护模式；
3. 分析pmtest\_1.asm中video段的属性；
4. 自定义一个GDT代码段，一个LDT代码段，完成学号和姓名的输入与打印；

自定义两个GDT代码段A/B，它们属于不同的特权级，实现相互之间的跳转。

# 实验过程分析

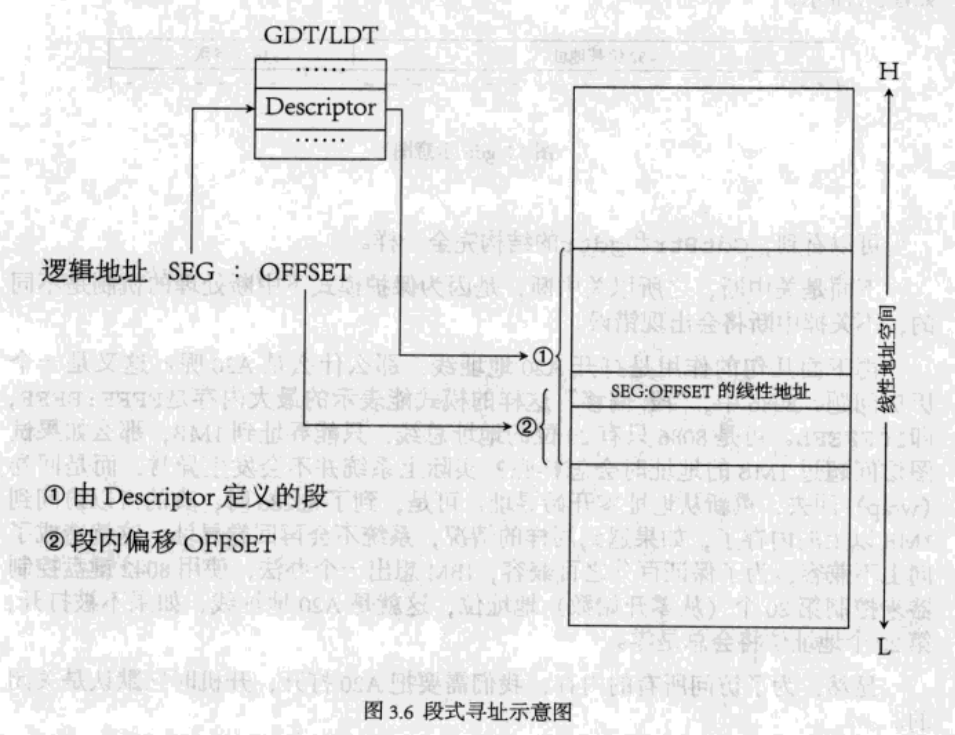
（详细记录实验过程，通过截图展示得到的结果。特别是对于实验中发生的故障和问题，要进行故障分析，说明故障排除的过程及方法。）

## 本次实验内容步骤

### 保护模式及其相关数据结构的简要介绍

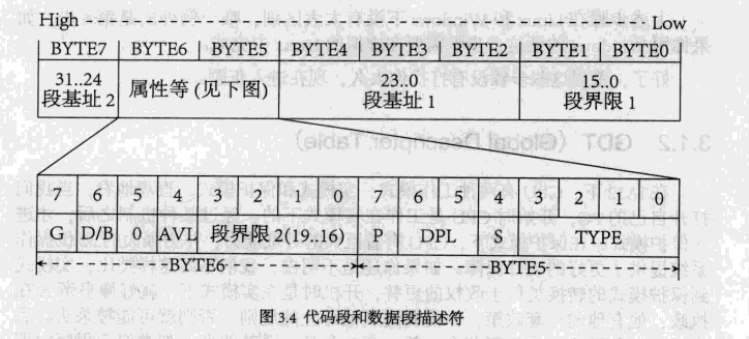
* 实地址模式和保护地址模式

实地址模式指16位cpu的时代计算机【段基址：段内偏移量】的寻址模式。当时计算机需要1M的寻址空间，因此实际物理地址=[16位段寄存器的段基址, 0000]+偏移量。由于当时用户可以随意修改段寄存器的内容，因此用户可以随意读写物理内存中的任何位置，系统对内存没有任何保护措施。



保护模式的思想为：改变段寄存器的功能，它不表示单一的基地址，而是指向数据结构的指针。该数据结构就是段描述符（Segment Descriptor）,每一个段描述符描述一个段的信息。所有的段描述符是一起存储的，构成一张表，这张表就是段描述符表。当某一个段描述符表是共有的，它就叫GDT；若它是进程私有的，就叫LDT。

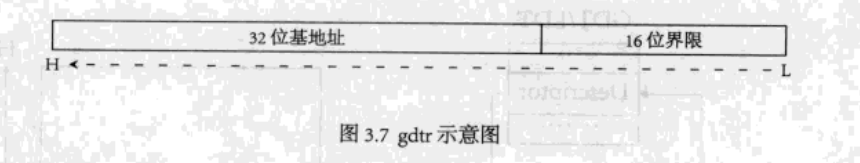
其中，Descriptor的结构分析如下：



我们主要关注Descriptor的属性：G表示粒度位，表示段界限是以字节为单位还是以4KB为单位；D/B表示默认的操作数大小，为了在32位处理器上兼容16位保护模式的程序；P表示段存在位，表示描述符对应的段是否存在；S位表示段类型，可能为系统段，也可能为数据段或代码段；TYPE的4位与段的读写可执行属性有关。

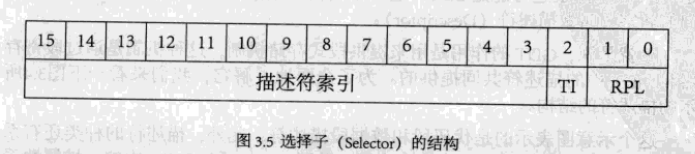
* GDT和LDT

整个系统只有一张GDT。GDT可以被放在内存的任何位置，只需要一个寄存器保存GDT的入口地址供CPU使用。该寄存器名为GDTR，一共48位，前32位表示GDT的基地址，后16位表示GDT的表长度。



系统内部可以拥有多张LDT，每个任务都可以拥有自己的LDT。同样地，LDT也有对应的LDTR来描述LDT的起始地址。但与GDTR不同，因为LDT本身也是内存的组成部分，也被GDT管辖，GDT中可能拥有多段LDT，因此LDTR中装载的并不是一个确定的地址，而是一个段选择子：当selector1正在执行，LDTR指向GDT中的LDT1;当selector2正在执行，LDTR指向GDT中的LDT2。LDTR的内容可以随着执行任务的不同而随时更改。

Selector更多地面向程序员。它是一个16位的值，包含了GDT或LDT的相关信息。当程序员在保护模式下访问内存时，CPU会根据Selector中的描述符索引在GDT或LDT中寻找对应的Descriptor，进而得到真正的物理地址。



* 总而言之，GDT/LDT类似于一张寻址表，Descriptor是表中的项，而GDTR/LDTR是存放该表位置的寄存器（对LDTR可能并不准确）；我们可以使用Selector对这张表进行查询操作。

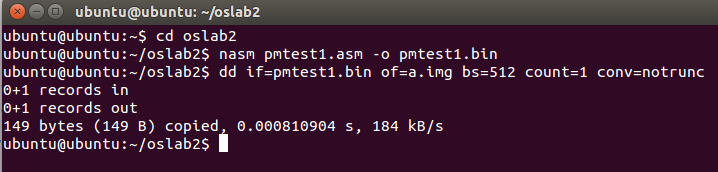
### 从实模式到保护模式

**首先，我们先运行这段代码。**

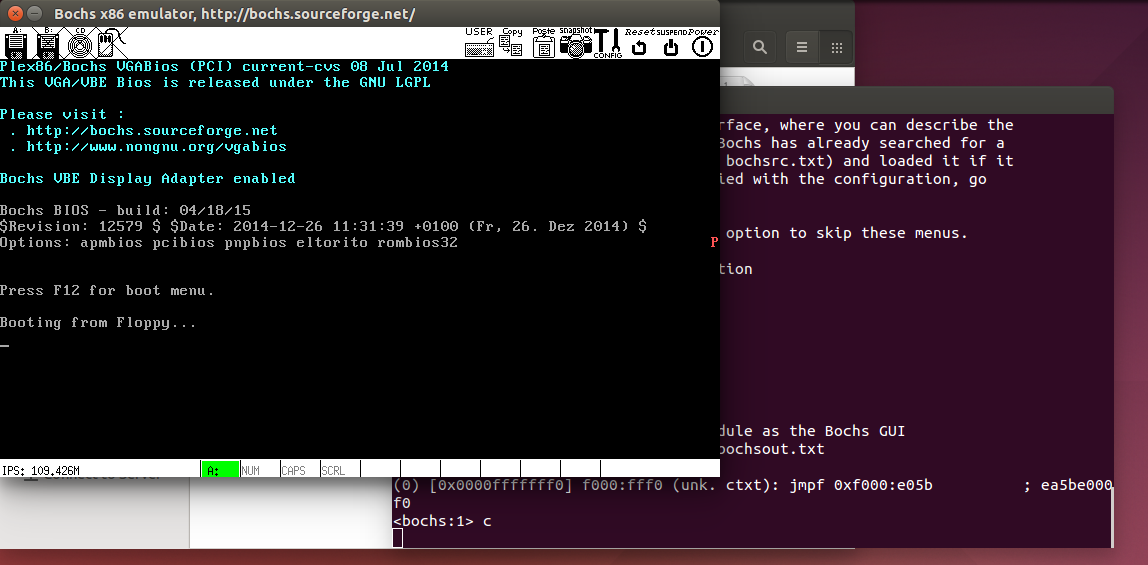
1. 编译pmtest1.asm

**nasm pmtest1.asm -o pmtest1.bin**

1. 将pmtest1.asm装入a.img



1. 运行bochs

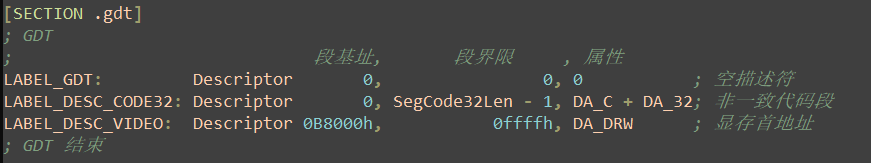


最终，程序在虚拟机的右侧打印出红色的’P’，证明实验成功。

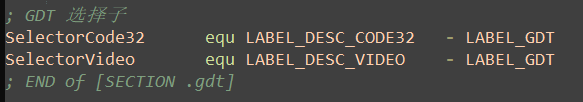
**之后，我们归纳出实模式-保护模式的流程。**

在本实验中，程序先进入16位的实模式，再通过GDTR跳转到32位的保护模式。

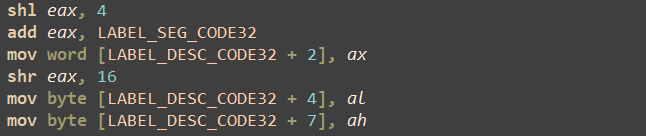
下面是GDT段描述符的相关代码。第一个GDT用于定位GDT，第二个GDT用于存放程序代码，第三个GDT用于存放数据。



下面是Selector的相关代码，用于定位GDT段。



之后，将实地址模式的地址格式转换位保护模式的地址格式。



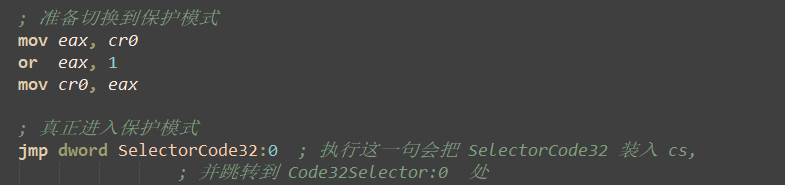
然后，通过GDTR加载GDT，同时使用cli关中断：。



由于实地址模式下使用20位数进行寻址，最大只能寻址1MB，因此需要额外打开A20地址线：

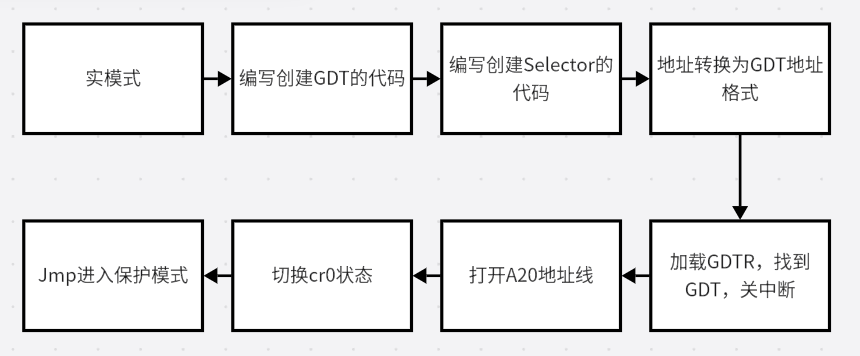


然后切换cr0为1，为进入保护模式做最后的准备，然后通过jmp进入保护模式。



这样，我们就完成了从实地址模式到保护模式的转换。

下面是相关的流程图：



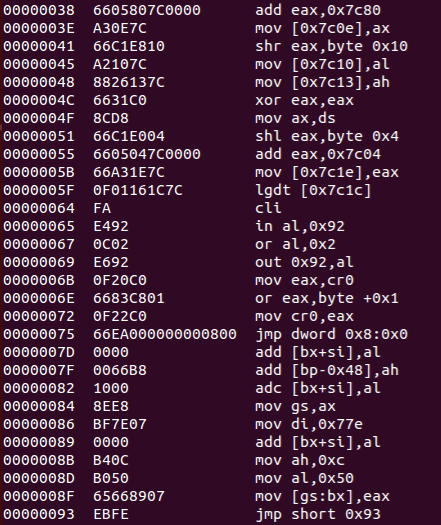
**最后，我们把代码中71行的dword删掉，对比更改前后的代码部分。**

在删掉dword之后，我们先比较更改前后编译的代码，也就是编译之后将其用ndisasm反编译回来。

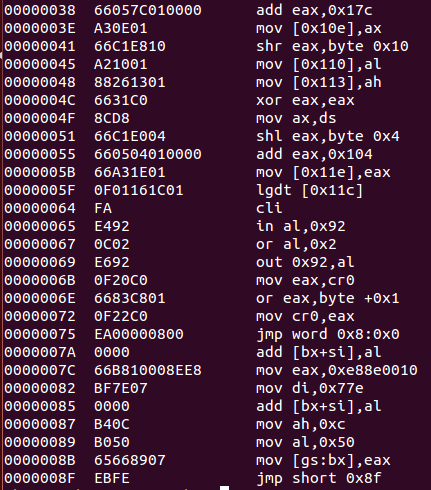
从理论上讲，**jmp dword SelectorCode32: 0**语句如果删掉dword，那么它只能得到SelectorCode32的后16位，而丧失前16位的信息。

在00000038之前的反编译代码，两者一致；之后的代码出现了差别：

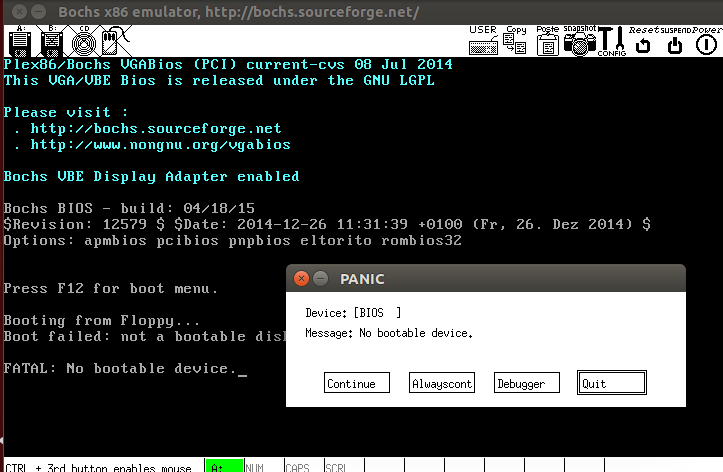
下面是原始代码的反编译结果：



下面的代码是删除dword之后的编译结果：



直观上来看，删掉dword之后反编译的代码会比原来的代码短一些；而且某些操作使用的数据也被截断了，比如0000003E行的代码，源代码mov的数值为0x7c0e，更改代码mov数值的代码变成了0x10e，在00000045、00000055处均出现了上述问题。然后较为明显的是000075行，jmp dword变成jmp word了。将编译的.bin文件使用bochs运行，删掉dword之后的程序没能成功执行。



### GDT切换，从保护模式到实模式

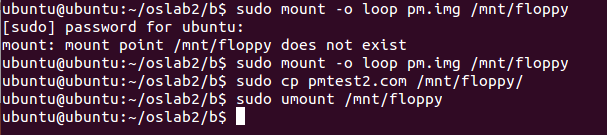
**首先，我们先运行这段代码。**

1. 首先，编译pmtest2.asm为pmtest2.com
2. 创建mnt/floppy文件夹，将pm.img挂载到该文件夹中，并pmtest2.com拷贝到该软盘内：

**sudo mkdir /mnt/floppy**

**sudo mount -o loop pm.img /mnt/floppy**

**sudo cp pmtest2.com /mnt/floppy**



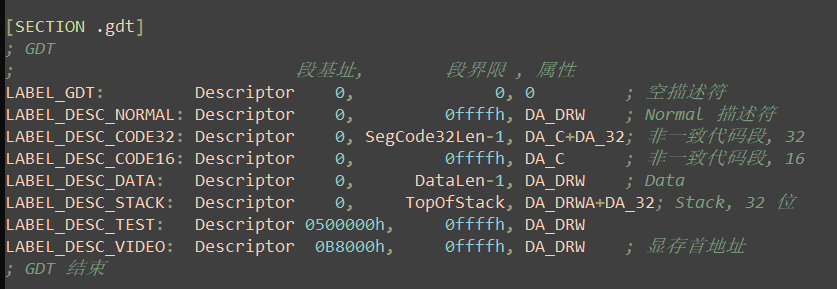
1. 启动bochs，切换到b:，执行pmtest2.com。



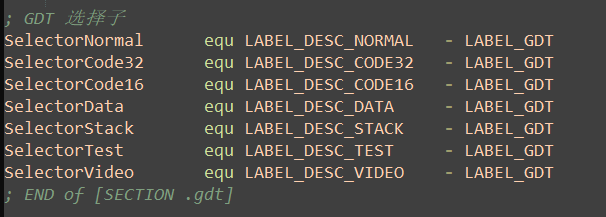
由输出内容看，我们成功运行了/b/中的程序，完成了由保护模式到实模式的转换。

**然后，我们关注保护模式-实模式的代码部分。**

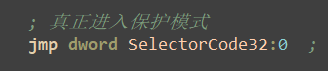
从保护模式到实模式，需要程序首先从实模式转换到保护模式。这一部分的代码与前文代码流程一致，关键在于之后返回实模式的过程。由此，需要额外增加对应的Selector。下面是GDT的定义部分：



然后是Selector的定义部分：



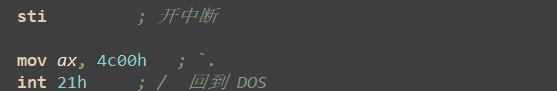
进入保护模式的代码和前文代码一致。我们从jmp进入保护模式开始：



要返回实模式，首先需要关闭A20地址线：



然后开中断，并回到DOS环境。



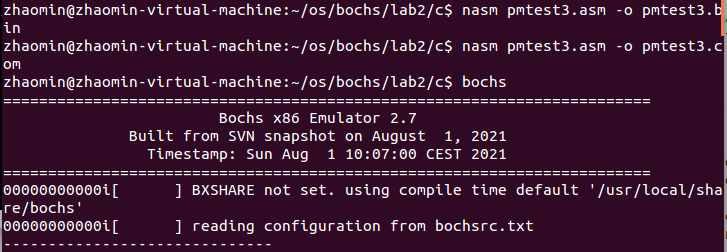
上述操作被封装为一个代码块LABEL\_REAL\_ENTRY，被以下代码调用。下面的代码会在调用之前先操作寄存器cr0变为0，表示cpu处于实模式。



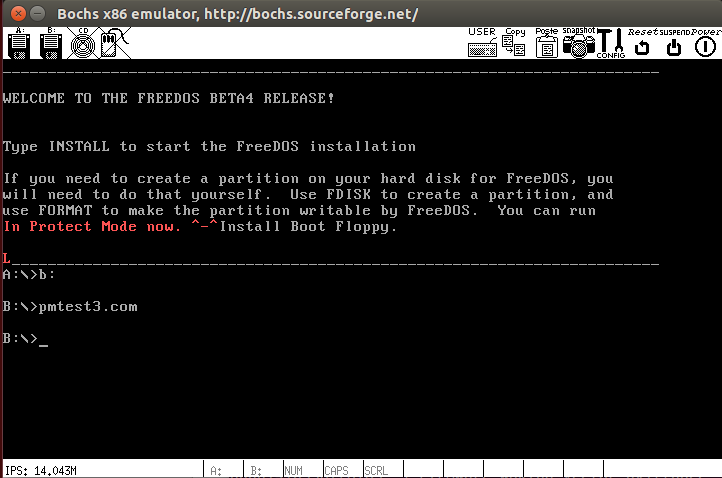
### LDT切换

实验步骤：

1. 首先，编译pmtest3.asm为pmtest3.bin
2. 再编译pmtest3.asm为pmtest3.com
3. 启动bochs



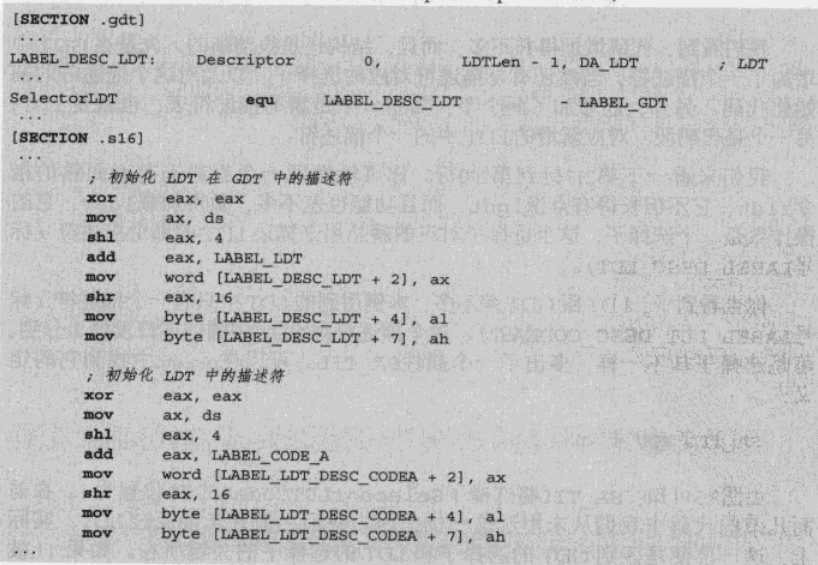
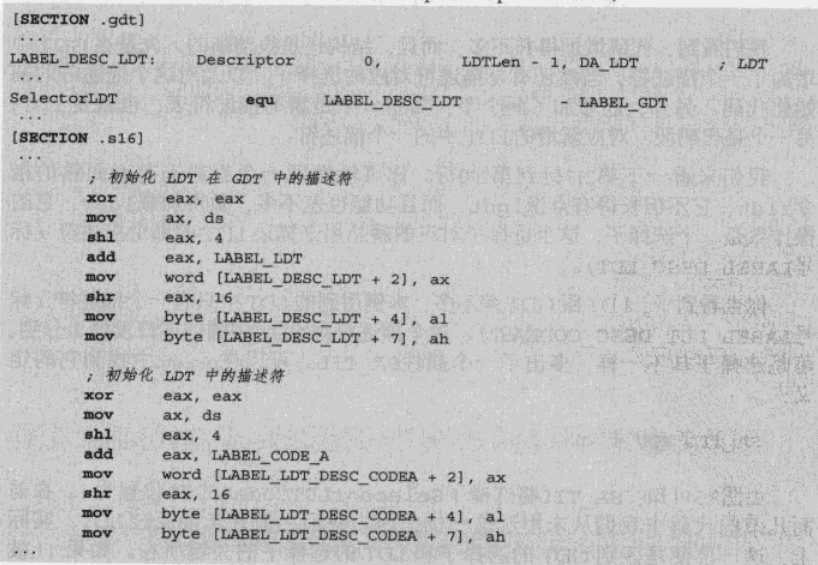
1. 在虚拟机上切换到b盘，执行pmtest3.com。

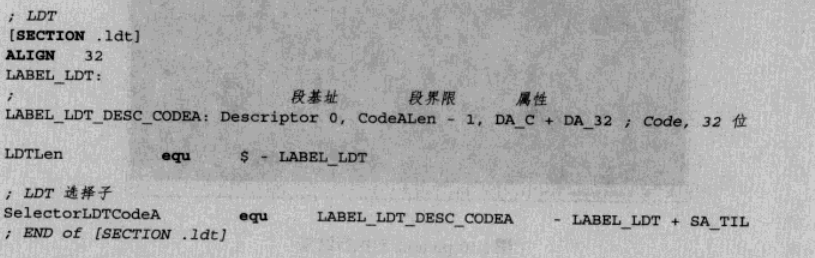


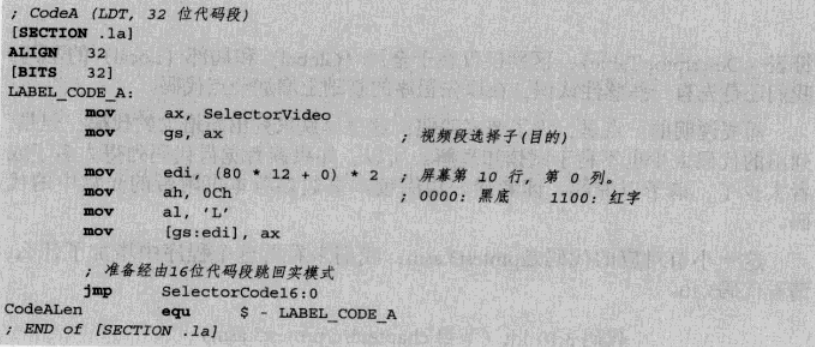
在[SECTION.s32]中打印完“In Protect Mode now”字符串后，LDT添加的代码段还将打印出“L”字符，说明我们正确完成了实验。

实验原理解释：

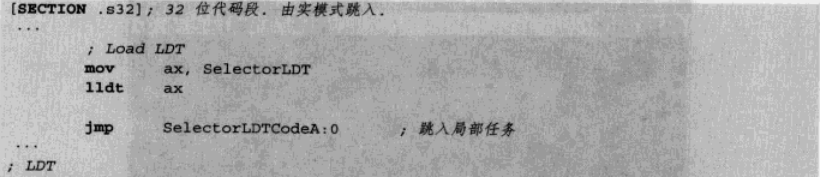
相较于GDT，LDT增加的代码段并不多，先是在GDT中增加了一个描述符以及与其相对应的选择子和对这个描述符的初始化代码：



另外增加了两节内容，一为新的描述符表，也就是LDT；另一为一个代码段，对应新增LDT中的一个描述符 LABEL\_LDT\_DESC\_CODEA



以及一个加载LDT的段落



本次实验用到的LDT的描述符LABEL\_LDT\_DESC\_CODEA，跟GDT的描述符没有什么区别，但是选择子却多了一个属性SA\_TLL;SA\_TLL将SelectorLDTCodeA的TI位置设为1。这是GDT的选择子和LDT选择子不同的关键所在，如果TI被置位，那么系统将从LDT中寻找相应描述符。





## 实验解决问题与课后动手改

### 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？

1. 从实模式到保护模式的关键步骤
   1. 准备 GDT
   2. 用lgdt加载gdtr
   3. 关中断
   4. 打开A20地址线
   5. 置cro的PE位为1
   6. 跳转，进入保护模式
2. 为什么要关中断？
   1. 因为保护模式下中断处理的机制是不同的，不关中断就会出现错误
3. 为什么要打开A20地址线？
   1. 8086只有20位的地址总线，只能寻址到1MB，当访问超过1MB的地址时，系统会回卷，重新从地址零开始寻址。但80286在访问超过寻址范围时不会回卷，这就造成了向上不兼容。
   2. 为了保证百分之百兼容，为了突破1M寻址的局限性、访问更大的内存空间，我们就需要打开A20地址线（开机时它是默认关闭的），如果不被打开，第20个地址位将会总是零。
4. 从保护模式切换回实模式的关键步骤
   1. 从保护模式下的32位代码段跳转到16位代码段
   2. 用对应选择字SelectorNormal初始化段寄存器，使其符合实模式的要求
   3. 置cr0的PE位为0
   4. 跳转，返回到实模式
   5. 程序重设各段寄存器的值，恢复实模式下的SP
   6. 关闭A20地址线
   7. 开中断
   8. 返回DOS

### 控制寄存器cr0的PE位为0表示cpu处于实模式，为1表示处于保护模式。通过你的实验验证这一点。

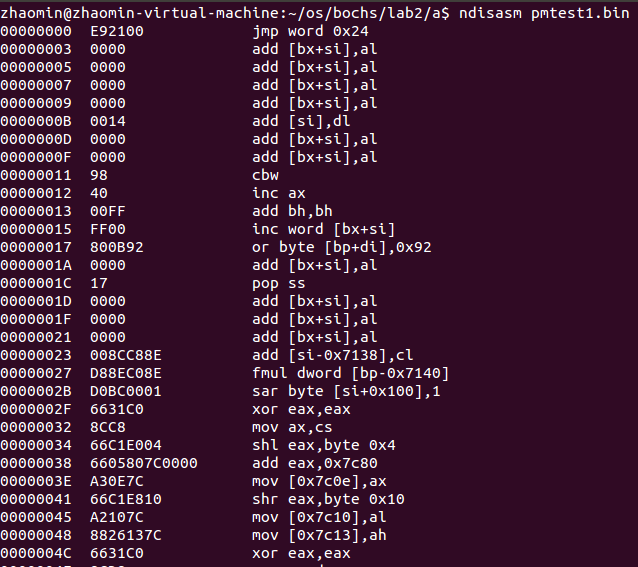
实验步骤：首先执行以下代码做好运行准备

nasm pmtest1.asm -o pmtest1.bin

dd if=pmtest1.bin of=a.img bs=512 count=1 conv=notrunc

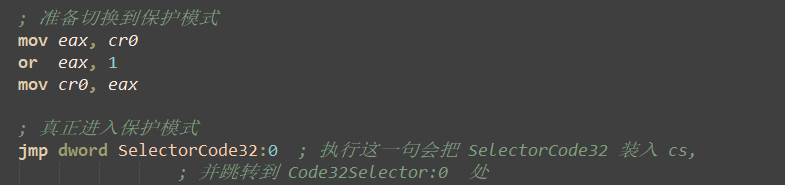
然后执行 ndisasm pmtest1.bin

反汇编pmtest1.bin可得





仔细查看反汇编代码可知，在程序的0072处将执行指令mov cr0，eax

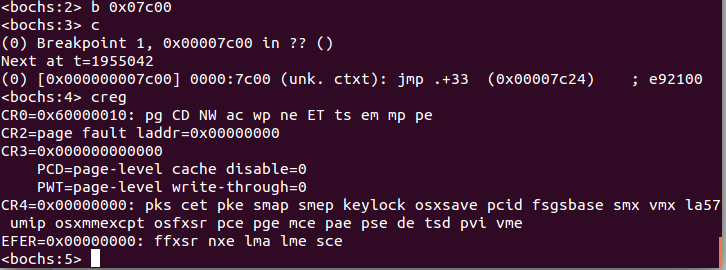


根据pmtest1.asm代码可知，在执行mov cr0，eax之后，cr0的PE位将被置1，至此进入保护模式的所有准备都已经做好，下一条指令 jmp dword SelectorCode32:0 将真正进入保护模式

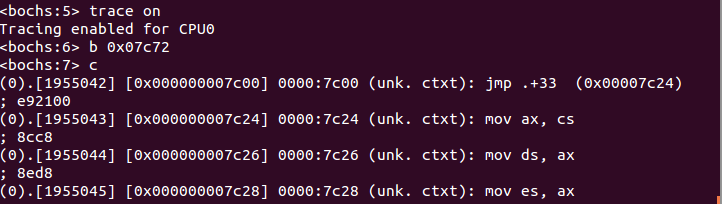
然后执行：

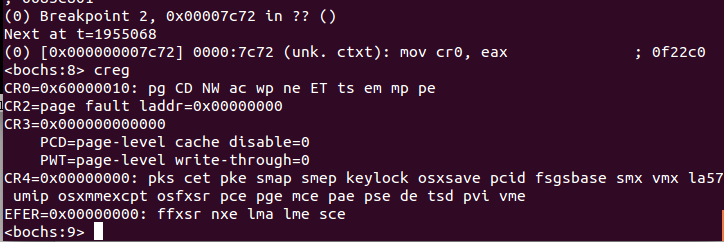
bochs

启动bochs调试，首先在07c00h处设置断点运行，并查看寄存器内容。可知此时cr0的PE位为0，处于实模式下

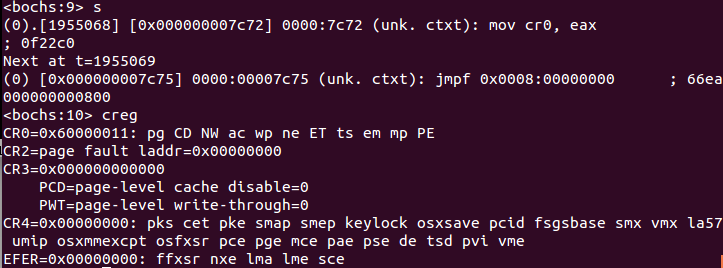


然后在0x07c72处设置断点并执行到此，可知此时cr0的PE位依旧为0，处于实模式下

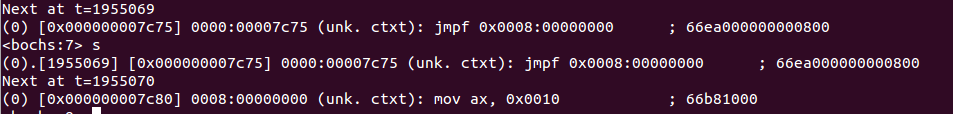




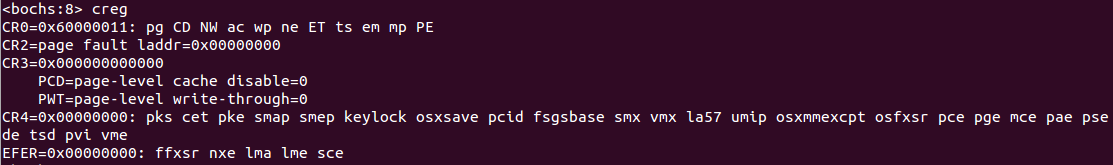
再次单步执行程序，使其执行mov cr0，eax，查看寄存器信息可知cr0的PE位变为1，已经为进入保护模式做好最后准备，下一条指令就将是 jmp dword SelectorCode32:0



再次单步执行jmp dword SelectorCode32:0



并查看寄存器值：

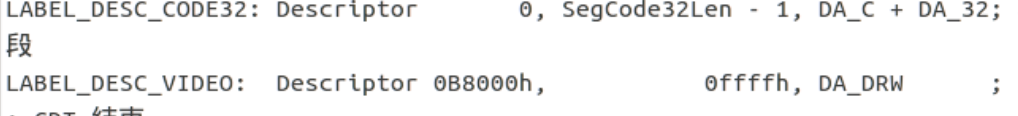


此时处于保护模式下，cr0的PE位将一直为1。

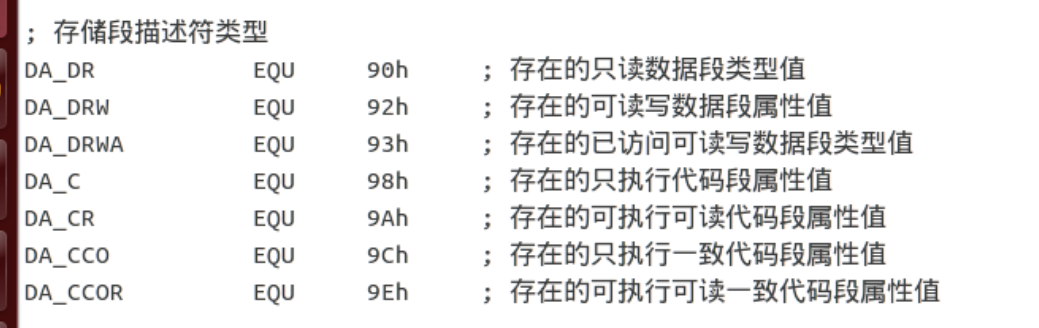
### 分析代码pmtest1.asm 中video段的属性，写出其完整的描述符。

视频段的完整描述符定义为：

LABEL\_DESC\_VIDEO: Descriptor 0B8000h, 0ffffh, DA\_DRW



其中，0B8000h 是段基址，这个描述符正好指向了显存位置，0ffffh 是段界限，DA\_DRW 是描述符属性。DA\_DRW对应pm.inc中的92h表明Video段是存在的可读写数据段。



92h，对应的二进制为：10010010

**其中：**

P位是1，表示这个段在内存中存在

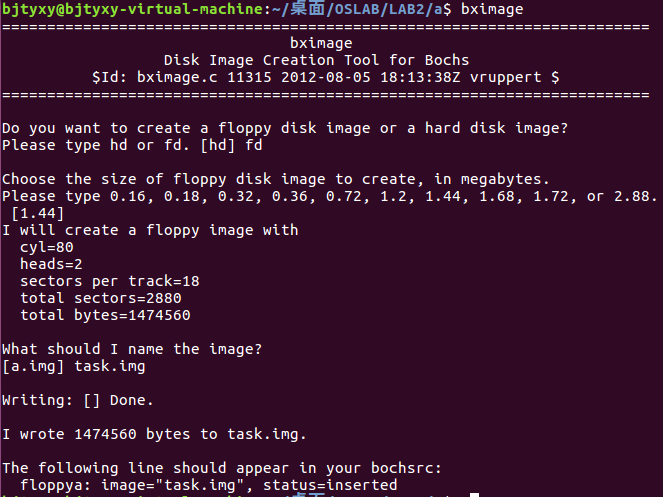
S位是1，表明这个段是数据段或者代码段（此处为数据段）

Type值为2，即0010，表明这个段是可读写的

DPL为0，即00，表明其特权级（数字越小，特权级越大）

### 课后动手改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(如你的姓名和学号)，然后LDT段内代码段功能为读取并打印该GDT的内容；

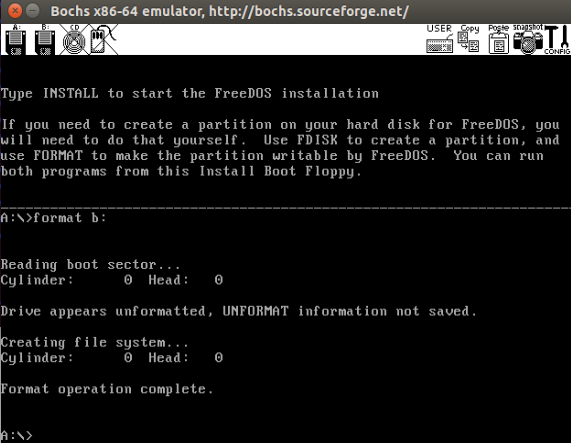
1. 用bximage生成一个软盘映像，命名为task.img



1. 修改bochsrc如下



1. 使用命令bochs -f bochsrc启动bochs，进入freedos，格式化B:盘



1. 编译代码：nasm task.asm -o task.com
2. 将task.com复制到虚拟软盘task.img上

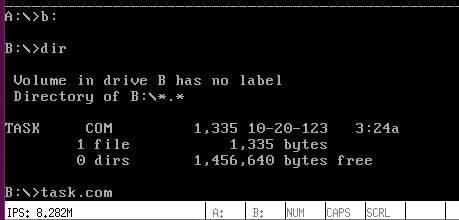
sudo mount -o loop task.img /mnt/floppy

sudo cp task.com /mnt/floppy/

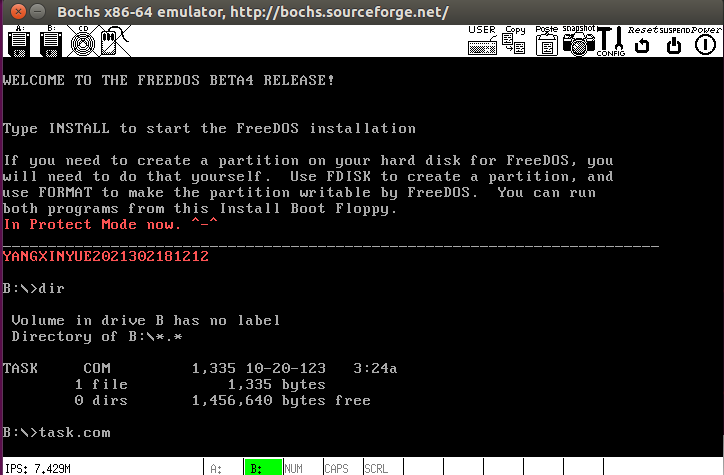
sudo umount /mnt/floppy



1. 启动bochs，进入freedos，执行命令B:\task.com



1. 运行结果如下

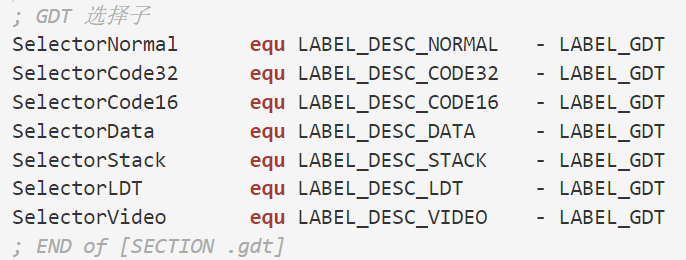


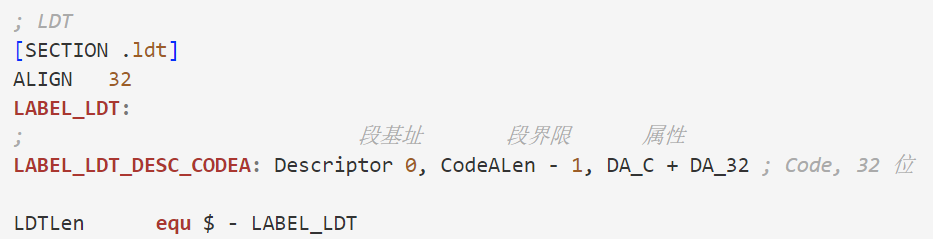
1. 关键代码如下

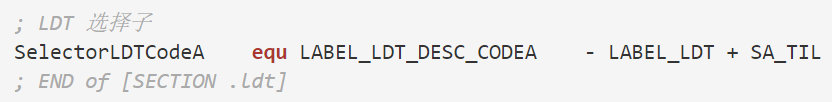
* 定义GDT代码段



* 定义GDT选择子



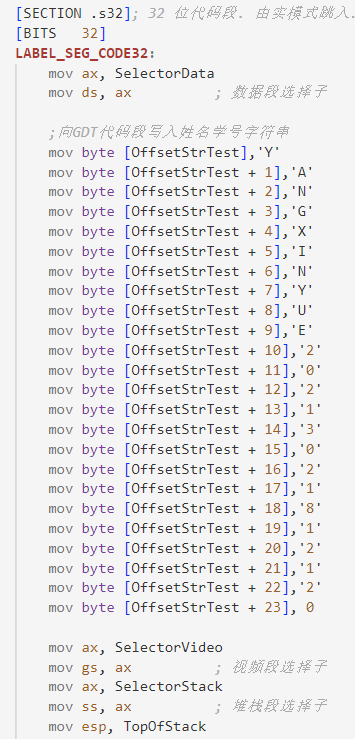
* 定义LDT代码段
* 定义LDT选择子



* 定义数据段，即GDT段内定义的内存数据结构LABEL\_DATA



* 向GDT代码段写入字符串

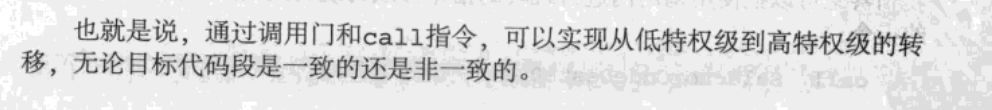


* 读取并打印该GDT的内容



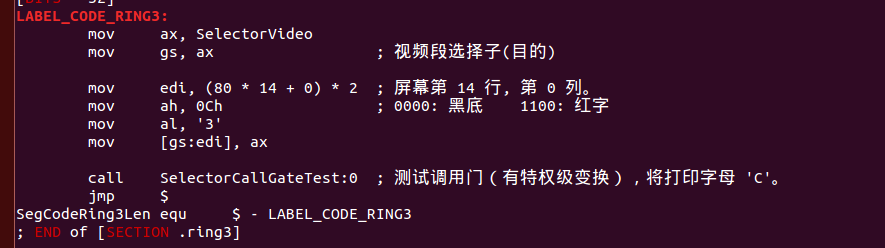
### 课后动手改2：自定义2个GDT代码段A、B，分属于不同特权级，功能自定义，要求实现A-->B的跳转，以及B-->A的跳转。

通过调用门来实现特权级的交换，实现打印两个字符



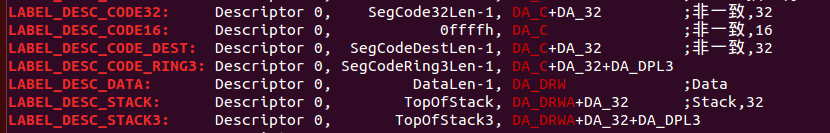
使用调用门的过程实际分为了两个部分，一部分是从低特权级到高特权级，通过调用门和call指令来实现；另一部分则是从高特权级到低特权级，通过ret指令来实现。

首先，使用call指令调用将要建立的调用门，所以还需要通过retf指令从子程序返回

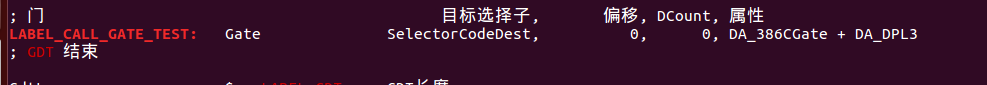




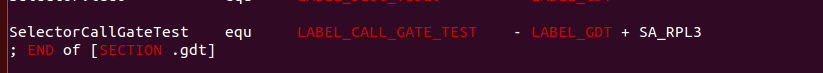
加入相应代码段的描述、选择子：



添加调用门：



其中的属性表明其是一个调用门。里面指定的选择子是SelectorCodeDest，表明目标代码段是刚刚新添加的代码段。下面是其选择子的定义

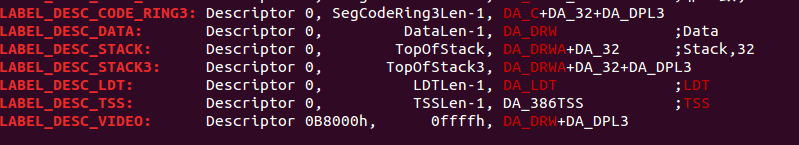


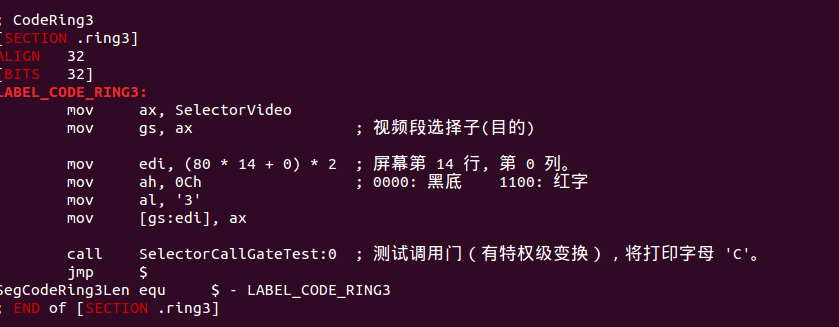
从低特权级到高特权级，还需要用到tss，所以添加初始化tss描述符的代码，并在特权级转换前加载



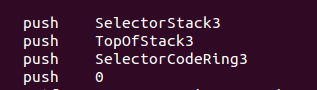
从高特权级到低特权级，通过ret指令来实现

在ret指令执行前，堆栈中需要准备好目标代码段的cs、eip以及ss和esp。ret执行后，即转到了低特权级代码中了。首先添加代码段和堆栈段





这段代码，最终会实现打印一个字符“C”。在准备好代码段和堆栈段后，将cs、eip以及ss和esp压栈，并执行retf指令返回



最后，让我们的程序可以返回实模式，跳入局部任务，并打印出字符L，并返回实模式。



最后的效果如下：



# 实验结果总结

（对实验结果进行分析。并理论联系实际，思考并列出本实验对应的OS原理的知识点，并说明本实验中的实现部分如何对应和体现了原理中的基本概念和关键知识点。）

在本次实验过程中，我们掌握了什么是实模式，什么是保护模式，了解到了实模式和保护模式之间的关系，实模式和保护模式的区别和特点。并在本次的实验过程中，通过编写相关的汇编程序，实现了从实模式到保护模式，保护模式再到实模式的转换的基本方法。掌握了从实模式到保护模式，包括设置GDT表、加载GDTR寄存器、打开A20地址线等关键步骤的切换流程。此外，我们还了解到了全局描述符表（GDT）、GDTR、descriptor的结构，及其之间的关系，并在实验中创建了GDT，了解到了gdt的构造，知道了GDT中的描述符所包含的段基址、段界限、特权级、类型等关键信息，而这些信息决定了处理器对段的访问权限。我们还通过实验体会到了如何使用GDT描述符来定义数据段和代码段，并理解了不同类型的描述符的含义和区别。

此外，通过实验我们进一步加深了对操作系统原理的理解，特别是关于内存管理、特权级管理、中断处理等方面的知识。我们学习到了GDT表的作用，GDTR寄存器的作用（存储GDT表的地址和大小信息），还学习到了如何通过设置控制寄存器CR0的PE位来控制处理器的工作模式。总之，整个实验过程帮助我们加深了对操作系统课程的理解，包括内存管理和特权级管理机制、中断处理机制等方面；也加强了我们对实模式和保护模式的区别和特点的认识。

# 个人分工及心得体会

（每个人分别填写自己在本次实验中的分工，并总结实验的心得体会。）

## 张子航

完成本次实验解决问题3，以及课后动手改2，并撰写了相应的实验报告内容，以及随后的实验内容总结

本次实验，让我可以更加熟练的使用bochs进行操作，也学到了有关实模式与保护模式的相关知识，了解到了两种模式之间的相互转换。于此同时，本次实验还帮助我回顾了有关操作系统以及汇编语言的相关知识。在实践中，我学习到了如何编写汇编程序来实现实模式到保护模式的切换，以及保护模式再切换回实模式的方法。这个过程使我对操作系统内核的运行机制有了更清晰的认识。

总之，本次实验帮助我更好地理解了操作系统底层的工作原理，也加深了我对计算机系统整体架构的认识。我相信这些知识和经验会对我的进一步学习和研究产生积极的影响。

## 辜汝曦

完成本次实验内容步骤的1-3题，并撰写报告中实验环境与实验步骤、以及本次实验内容步骤1-3题的部分。

这次实验让我更加熟悉nasm和bochs的操作，也了解了远古时代操作系统的内存管理方式。不仅如此，它还让我回顾了操作系统中有关内存的知识点，并通过汇编语言将其具象化：之前汇编语言学习CS、DS、ES这些有关段的知识时，总是不能理解，只能硬背代码；在操作系统理论课上学习OS的段式管理时，自己也没有将这两者联系起来。这次实验将这两者联系起来，让我对操作系统的分段机制以及内存访问有了更深刻的了解，也在用汇编语言操作内存中数据段、代码段这一方面更为熟练。

## 杨馨悦

完成本次实验解决问题与课后动手改的1、4题，撰写报告中与之相对应的部分，并完成实验报告排版。

通过本次实验，我更加熟悉了使用bochs调试汇编程序的基本步骤及Bochs的调试基本命令。通过阅读和编写代码，不仅回顾了汇编语言的基础知识，加深了对寻址操作、jmp跳转指令、段寄存器等知识点的印象，还认识了实模式与保护模式，学习了GDT、LDT的基本概念。与此同时，本次实验让我能够在实践与调试中不断深化对教材内容的理解、增强动手能力，更让我明白了操作系统底层的代码逻辑与思维，对我今后系统性地学习操作系统有很大帮助。

## 赵敏

完成本次实验的1~4题，并撰写LDT切换部分的实验报告，以及课后手改部分的第2题。

在本次实验中，再次熟悉了bochs的运行调试过程，学会经过断点调试灵活获取自己所需要的信息，查看程序执行效果以及修改代码。学习到实模式和保护模式的切换方式，以及GDT、LDT方法的相同与不同。在阅读理解课本内容的同时，也对过往学习的汇编语言、操作系统的知识有了更深的感悟和理解，能够更好的融会贯通过去的知识来为所要完成的实验服务。期待能在后续课程中更深入了解操作系统的知识。

# 思考题目（课上抽查）

1. 控制寄存器cr0的PE位为0表示cpu处于实模式，为1表示处于保护模式。通过你的实验验证这一点。

2. 分析代码pmtest1.asm 中video段的属性，写出其完整的描述符。

3. 完成参考资料1第47页下面1-6要求的任务，在新增的代码段中完成在屏幕上打印你的名字和学号的任务。

4. 总结不同特权级代码转移规则，并自己设计实验进行验证。