

**本科生实验报告**

实验课程：操作系统原理实验

任课教师：刘宁

实验题目：从实模式到保护模式

专业名称：计算机科学与技术

学生姓名：

学生学号：

实验地点：实验中心B202

实验时间：2025.3.26

**Section 1 实验概述**

在一次实验中中，我们学习了有关于x86汇编的相关内容，同时了解计算机的启动过程并在16位的实模式环境下编写了程序。

本次实验，将会学习到从16位的实模式下跳转到32位的保护模式，然后在平坦模式下运行自己编写的32位汇编程序程序。同时，将学习读取磁盘的两种方式，LBA和CHS的方式，学习到如何使用I/O端口和硬件交互，将磁盘中的程序导入到我们内存当中，为后面保护模式编程打下基础。

**Section 2 预备知识与实验环境**

* 预备知识：x86汇编语言、IA-32处理器体系结构、LBA方式读写硬盘、CHS方式读写硬盘、保护模式
* 实验环境：
  + 虚拟机版本/处理器型号：Virtualbox7.0.6、Ubuntu18.04
  + 代码编辑环境：gedit文本编辑器、gdb调试器、qemu
  + 代码编译工具：nasm

**Section 3 实验任务**

* 实验任务1：复现“加载bootloader”这一节
* 实验任务2：将LBA28读取硬盘的方式换成CHS读取，同时给出逻辑扇区号向CHS的转换公式。
* 实验任务3：复现“进入保护模式”一节，使用gdb工具在进入保护模式的4个重要步骤上设置断点，并结合代码、寄存器的内容等来分析这4个步骤。
* 实验任务4：在进入保护模式后，按照如下要求，编写并执行一个自己定义的32位汇编程序，使用两种不同的自定义颜色和一个自定义的起始位置(x,y)，使得bootloader加载后，在显示屏坐标(x,y)处开始输出自己的学号+姓名拼音首字母缩写，要求相邻字符前景色和背景色必须是相互对调的。

**Section 4 实验步骤与实验结果**

------------------------- **实验任务1** -------------------------

* 任务要求：复现加载bootloader这一节，说说怎么做的并提供结果截图
* 思路分析：

①首先将输出‘run bootloader’的代码放入到bootloader.asm中，在bootloader.asm中的起始地址为0x7e00，

②然后在MBR中我们使用LBA的方式以及in/out的方式从磁盘的扇区中加载bootloader到内存，并跳转到bootloader的起始地址0x7e00执行

内存结构如下表：

| name | start | length | end |
| --- | --- | --- | --- |
| MBR | 0x7c00 | 0x200(512B) | 0x7e00 |
| bootloader | 0x7e00 | 0xa00(512B \* 5) | 0x8800 |

* 实验步骤：

**Step1：编写bootloader.asm文件实现run bootloader的输出**，这段代码我们需要放在0x7e00的地址处编写

org 0x7e00

[bits 16]

mov ax, 0xb800

mov gs, ax

mov ah, 0x03 ;青色

mov ecx, bootloader\_tag\_end - bootloader\_tag

xor ebx, ebx

mov esi, bootloader\_tag

output\_bootloader\_tag:

mov al, [esi]

mov word[gs:bx], ax

inc esi

add ebx,2

loop output\_bootloader\_tag

jmp $ ; 死循环

bootloader\_tag db 'run bootloader'

bootloader\_tag\_end:

**Step2：在mbr中使用LBA的方式读写磁盘的第1~5号扇区到我们的内存**，LBA读取硬盘的规则如下表所示：

|  |  |  |
| --- | --- | --- |
| 端口 | 功能 | 作用 |
| 0x1F0 | 缓存 | 从该端口中读取硬盘的一个字节 |
| 0x1F2 | 扇区数 | 将要读取的扇区数量写入0x1F2端口 |
| 0x1F3 | LBA 0~7位 | 逻辑扇区号0~7位 |
| 0x1F4 | LBA 8~15位 | 逻辑扇区号8~15位 |
| 0x1F5 | LBA 16~23位 | 逻辑扇区号16~23位 |
| 0x1F6 | LBA 24~27位 | 低4位逻辑扇区24~27位，高4位特殊 |
| 0x1F7 | 执行状态 | 写入0x20读取，其后表示状态 |

所以我们只需要根据上表将LBA对应的位数传入到对应的端口，这里需要注意，in/out命令只能够使用dx和al寄存器，然后写0x20到0x1F7端口开始读写就可，最后我们从0x1F0端口读出512个字节即可实现读取整个扇区，具体代码如下：

asm\_read\_hard\_disk:

; 从硬盘读取一个逻辑扇区

; 参数列表

; ax=逻辑扇区号0~15位

; cx=逻辑扇区号16~28位

; ds:bx=读取出的数据放入地址

; 返回值

; bx=bx+512

mov dx, 0x1f3

out dx, al ; LBA地址7~0

inc dx ; 0x1f4

mov al, ah

out dx, al ; LBA地址15~8

mov ax, cx

inc dx ; 0x1f5

out dx, al ; LBA地址23~16

inc dx ; 0x1f6

mov al, ah

and al, 0x0f

or al, 0xe0 ; LBA地址27~24

out dx, al

mov dx, 0x1f2

mov al, 1

out dx, al ; 读取1个扇区

mov dx, 0x1f7 ; 0x1f7

mov al, 0x20 ;读命令

out dx,al

读取512个字节：

; 等待处理其他操作

.waits:

in al, dx ; dx = 0x1f7

and al,0x88

cmp al,0x08

jnz .waits

; 读取512字节到地址ds:bx

mov cx, 256 ; 每次读取一个字，2个字节，因此读取256次即可

mov dx, 0x1f0

.readw:

in ax, dx

mov [bx], ax

add bx, 2

loop .readw

ret

* 实验结果展示：通过执行前述代码，可得下图结果。



------------------------- **实验任务2** -------------------------

* 任务要求：将LBA28读取硬盘的方式换成CHS读取，同时给出逻辑扇区号向CHS的转换公式。
* 思路分析：

①首先我们需要知道LBA逻辑扇区号与CHS之间的转换关系，这里我们把CHS看成磁盘中的扇区的物理坐标，那么这时候逻辑扇区号LBA可以看成是磁盘中扇区坐标的一维映射，对于这种三维到一维的映射，我们可以使用整除和求余数这两种运算从一维映射回三维。

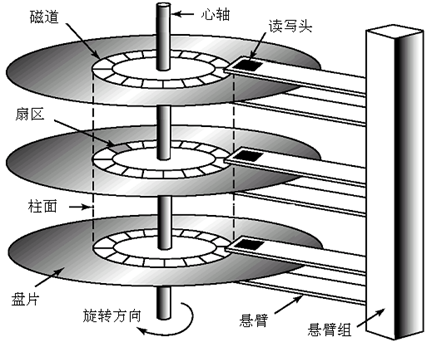
②在获得了从LBA到CHS的转换关系后，我们只需要要将将实验任务1的代码中的asm\_read\_hard\_disk函数中的内容进行修改就行了，LBA是通过in，out的指令进行I/O操作读写磁盘，如果我们知道了LBA对应的CHS号，我们就可以使用BIOS中的int 13H中断服务来读写磁盘中对应扇区的内容。

* 实验步骤：

磁盘的物理结构如下，可以看到一个扇区可以由柱面号、磁头号和扇区号三个参数唯一确定，每个柱面的磁头数为18，每个磁道有63个扇区，这里我们假设PS为每个磁道的扇区数，PH为每个柱面的磁道数，我们就可以得到：

CHS到LBA的转换公式：(LBA从0开始)

LBA到CHS的转换公式：(//为整除 mod为取余数)



**Setp1：使用上述的转换公式将LBA转换到对应的CHS**，具体代码如下：

asm\_read\_hard\_disk:

;扇区数63 磁道数18 一个柱面有63\*18个扇区

pushad

lba dw 0 ;LBA

c dw 0 ;柱面号c

h dw 0 ;磁头号h

s dw 0 ;磁区号s

mov [lba], ax

;计算柱面号 c= LBA // (18\*63)

mov ax,[lba]

mov cx,1134 ;18\*63=1134

xor dx,dx ;被除数dx:ax

div cx ;ax为商

mov [c],al

;计算磁头号 h= (LBA // 63) % 18

mov ax,[lba]

mov cx,63

xor dx,dx

div cx ;ax为LBA/63的商即LBA整除63的结果

mov cx,18

xor dx,dx

div cx ;除以18,dx为(LBA//63)mod 18的结果

mov [h],dl

;计算扇区号 s= LBA % 63

mov ax,[lba]

mov cx,63

xor dx,dx

div cx ;LBA mod 63 结果存放在dx即余数

inc dx ;扇区号从1开始所以要加1

mov [s],dl

**Step2：使用BIOS中int 13H实现磁盘扇区的读取**

int 13H的各个参数如下表所示：

|  |  |
| --- | --- |
| 寄存器 | 含义 |
| ah | 2H |
| al | 扇区数 |
| ch | 柱面号 |
| ah | 磁头号 |
| cl | 扇区号 |
| dl | 驱动器号 |
| es:bx | 缓冲区地址 |

我们根据上表将各个参数填入对应的寄存器中，然后使用int 13H启动，具体代码如下：

;调用BIOS

mov ch,[c] ;c

mov dh,[h] ;h

mov cl,[s] ;s

mov al,5 ;读连续5块扇区

mov ah,02H ;功能号02H

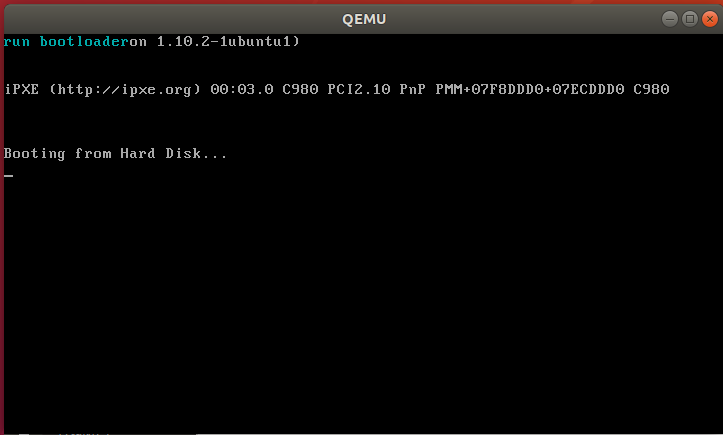
mov dl,80H

int 13H

popad

ret

* 实验结果展示：通过执行前述代码，可得下图结果。



------------------------- **实验任务3** -------------------------

* 任务要求：复现“进入保护模式”一节，使用gdb工具在进入保护模式的4个重要步骤上设置断点，并结合代码、寄存器的内容等来分析这4个步骤，最后附上结果截图。
* 思路分析：

①进入保护模式的四个步骤：

1.准备GDT，用lgdt指令加载GDTR信息。

2.打开第21根地址线。

3.开启cr0的保护模式标志位。

4.远跳转，进入保护模式。

根据保护模式的四个步骤一步一步的构建bootloader.asm的代码进入，32位地址的保护模式，

②为了检查我们的设计是否正确，需要使用gdb工具在这四步前后都设置断点，然后查看寄存器中的值，观察是否发生了预期的变化来判断我们的代码是否正确的执行了的方式来对这四步进行调试，观察是否成功。

* 实验步骤：

**一、进入保护模式**

**第一步：准备GDT，用lgdt指令加载GDTR信息。**

**Step1：**首先，我们需要**定义代码段描述符、数据段描述符、栈段描述符和视频段描述符这四个段描述符**，然后在GDT中依次放入0描述符，数据段描述符、堆栈段描述符、显存段描述符和代码段描述符，注意，GDT的第0个描述符必须是全0的描述符代码如下。

段描述符的由64位构成，分别含义如下，我们主要来看高32位，数据段描述符的高32位如下，00000000 1 0 1 0 1111 1 00 1 0010 00000000

G表示粒度 G=1表示段界限以4KB为单位。

D/B位是默认操作数的大小或默认堆栈指针的大小

L位是 64 位代码段标志

AVL位是保留位。

P位是段存在位， P=1表示段存在， P=0表示段不存在。

DPL指明访问该段必须有的最低优先级，优先级从0-3依次降低，即0拥有最高优先级，3拥有最低优先级。

S位是描述符类型。S=0表示是系统段，S=1表示该段位代码段或数据段。

TYPE指示代码段或数据段的类型



;空描述符

mov dword [GDT\_START\_ADDRESS+0x00],0x00

mov dword [GDT\_START\_ADDRESS+0x04],0x00

;创建描述符，这是一个数据段，对应0~4GB的线性地址空间

mov dword [GDT\_START\_ADDRESS+0x08],0x0000ffff

; 基地址为0，段界限为0xFFFFF

mov dword [GDT\_START\_ADDRESS+0x0c],0x00cf9200

; 粒度为4KB，存储器段描述符

;建立保护模式下的堆栈段描述符

mov dword [GDT\_START\_ADDRESS+0x10],0x00000000

; 基地址为0x00000000，界限0x0

mov dword [GDT\_START\_ADDRESS+0x14],0x00409600

; 粒度为字节

;建立保护模式下的显存描述符

mov dword [GDT\_START\_ADDRESS+0x18],0x80007fff

; 基地址为0x000B8000，界限0x07FFF

mov dword [GDT\_START\_ADDRESS+0x1c],0x0040920b

; 粒度为字节

;创建保护模式下平坦模式代码段描述符

mov dword [GDT\_START\_ADDRESS+0x20],0x0000ffff

; 基地址为0，段界限为0xFFFFF

mov dword [GDT\_START\_ADDRESS+0x24],0x00cf9800

; 粒度为4kb，代码段描述符

**Step2：创建一个48位的变量存放GDTR的信息**，由于没有48位的寄存器我们使用创建变量的方式来创建我们的GDTR，GDTR的前16位存放我们的界限，这里的界限就是描述符所占的字节数减1，即4\*8-1=39

pgdt dw 0

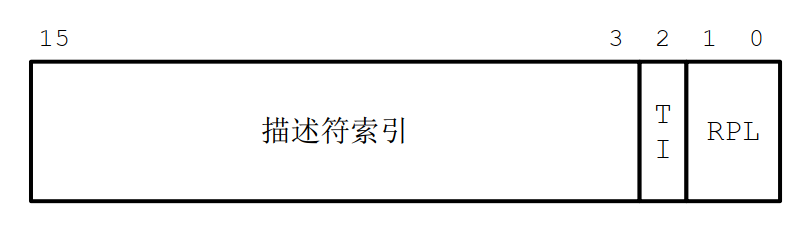
dd GDT\_START\_ADDRESS

;初始化描述符表寄存器GDTR

mov word [pgdt], 39 ;描述符表的界限

lgdt [pgdt]

**Step3：根据段描述符的内容设置段选择子**，段选择子总共有16位，其含义如下，比如数据段描述符是GDT中第1个描述符，因此高13位为0000 0000 0000 0001，第2位TI=0表示GDT，最后两位RPL设为最高特权级00这样数据段选择子就是0000 0000 0000 0001000=0x08.剩下的以此类推。



; \_\_\_\_\_\_\_\_\_\_\_\_\_Selector\_\_\_\_\_\_\_\_\_\_\_\_\_

;平坦模式数据段选择子

DATA\_SELECTOR equ 0x8

;平坦模式栈段选择子

STACK\_SELECTOR equ 0x10

;平坦模式视频段选择子

VIDEO\_SELECTOR equ 0x18

VIDEO\_NUM equ 0x18

;平坦模式代码段选择子

CODE\_SELECTOR equ 0x20

**第二步：打开第21根地址线，**第21根地址线的开关位于南桥芯片的端口A20，使用 in，out 指令可以对主板端口进行读、写操作

Step1：打开第21根地址线

in al,0x92 ;南桥芯片内的端口

or al,0000\_0010B

out 0x92,al ;打开A20

**第三步：开启cr0的保护模式标志位，** CR0 是 32 位的寄存器，包含了一系列用于控制处理器操作模式和运行状态的标志位，其第0位是保护模式的开关位，称为PE（protect mode enable）位，CPU 进入保护模式

cli ;中断机制尚未工作

mov eax,cr0

or eax,1

mov cr0,eax ;设置PE位

**第四步：远跳转，进入保护模式，**此时，jmp指令将CODE\_SELECTOR送入cs，将protect\_mode\_begin + LOADER\_START\_ADDRESS送入eip，进入保护模式。

jmp dword CODE\_SELECTOR:protect\_mode\_begin

**第五步：在保护模式下，输出输出“enter protect mode”**。

**Step1：将选择子放入对应的段寄存器**

;16位的描述符选择子：32位偏移

;清流水线并串行化处理器

[bits 32]

protect\_mode\_begin:

mov eax, DATA\_SELECTOR ;加载数据段(0..4GB)选择子

mov ds, eax

mov es, eax

mov eax, STACK\_SELECTOR

mov ss, eax

mov eax, VIDEO\_SELECTOR

mov gs, eax

**Step2：32位下将enter protect mode字符串依次送入0xB8000即VIDEO\_SELECTOR对应的gs段寄存器。**

mov ecx, protect\_mode\_tag\_end - protect\_mode\_tag

mov ebx, 80 \* 2

mov esi, protect\_mode\_tag

mov ah, 0x3

output\_protect\_mode\_tag:

mov al, [esi]

mov word[gs:ebx], ax

add ebx, 2

inc esi

loop output\_protect\_mode\_tag

jmp $ ; 死循环

protect\_mode\_tag db 'enter protect mode'

protect\_mode\_tag\_end:

**二、使用gdb对上述代码进行调试**

我们按照之前的方法进入到gdb进行调试，这里我们简化了之前实验进行调试的步骤，将他们都打包成一个makefile文件，使用更加方便的make指令来进行我们的调试

**Step1：编写我们的makefile文件**，这个文件将我们在启动gdb时所需要的所有指令都打包到了一起，我们只需要在终端中输入make+对应的标号就可以一次性运行这个标号下的所有指令，比如我们使用make debug就可以很快的启动我们的gdb.

run:

@qemu-system-i386 -hda hd.img -serial null -parallel stdio

debug:

@qemu-system-i386 -s -S -hda hd.img -serial null -parallel stdio &

@sleep 1

@gnome-terminal -e "gdb -q -x gdbinit"

build:

@nasm -g -f elf32 mbr.asm -o mbr.o

@ld -o mbr.symbol -melf\_i386 -N mbr.o -Ttext 0x7c00

@ld -o mbr.bin -melf\_i386 -N mbr.o -Ttext 0x7c00 --oformat binary

@nasm -g -f elf32 bootloader.asm -o bootloader.o

@ld -o bootloader.symbol -melf\_i386 -N bootloader.o -Ttext 0x7e00

@ld -o bootloader.bin -melf\_i386 -N bootloader.o -Ttext 0x7e00 --oformat binary

@dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc

@dd if=bootloader.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc

clean:

@rm -fr \*.bin \*.o

**Step2：编写gdbinit，**同理gdb中的指令也可以打包成一个文件夹，这里我们将gdb中连接qemu加载代码符号表以及在进入保护模式的四个步骤前后都设置了断点这些命令都打包好，方便我们测试进入保护模式的四个步骤

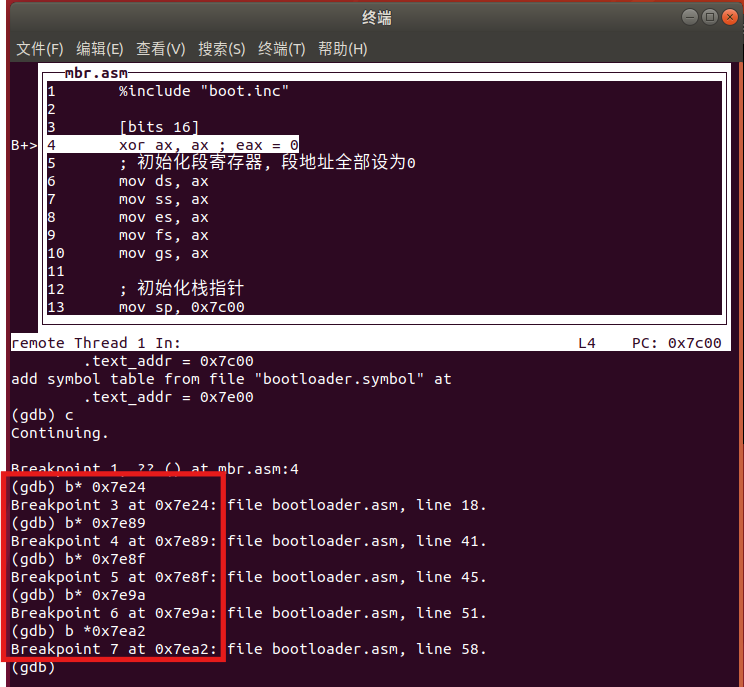


图 1 设置断点

target remote:1234

set disassembly-flavor intel

b \*0x7c00

b \*0x7e00

b \*0x7e24

b \*0x7e89

b \*0x7e8f

b \*0x7e9a

b \*0x7ea2

layout src

add-symbol-file mbr.symbol 0x7c00

add-symbol-file bootloader.symbol 0x7e00

**Step3：查看GDT中的值以及gdtr中的值**

在设置GDT之前我们0x8800中的五个值都为0，在运行到赋值结束之后，我们再次查看0x8800地址中的值以及pgdt地址中的值，可以看到我们的段描述符和描述符寄存器GDTR已经设置好了，如下图所示

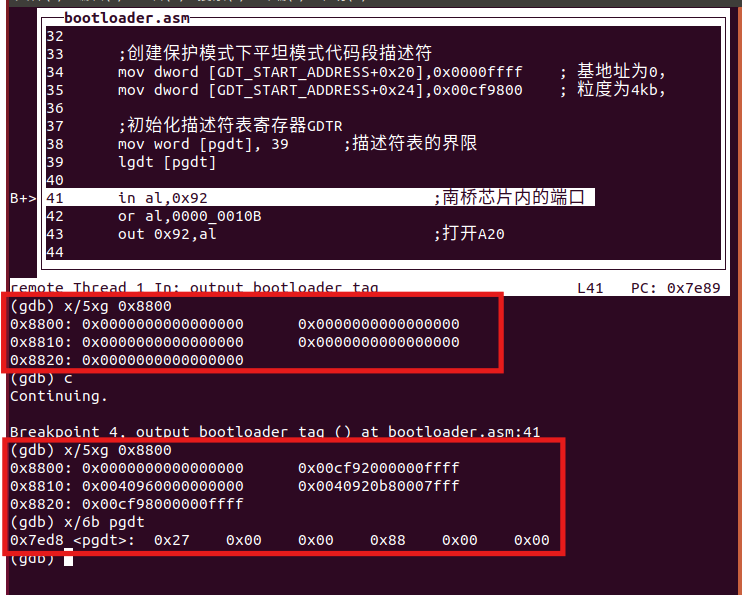


图 2 段描述符以及描述符表寄存器

**Step3：打开第21条地址线，**我们使用in/out指令将南桥芯片的第21条地址线的端口取出，然后使用或运算将第二位设置为1，然后在返回到端口中，下图展示了al寄存器的变成了0010b，可以看到第2位成功设置为了1

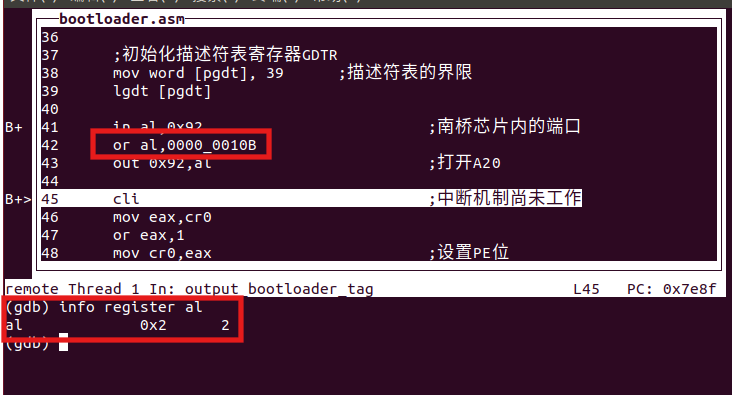


图 3 打开第21条地址线

**Step4：开启cr0保护模式标志位，**为了打开保护模式我们需要将cr0的第0位设置为1，这里我们可以看到cr0的值从0x10变成了0x11，表示我们设置成功了

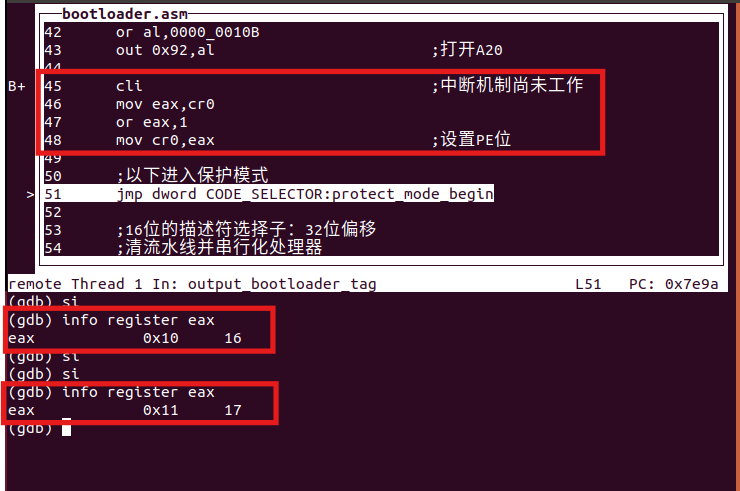


图 4 开启cr0保护模式标志位

**Step5：执行远跳转，**可以看到我们按下c继续执行远跳转之后，qemu出现了enter protect mode的字样，表示我们成功进入到了保护模式中

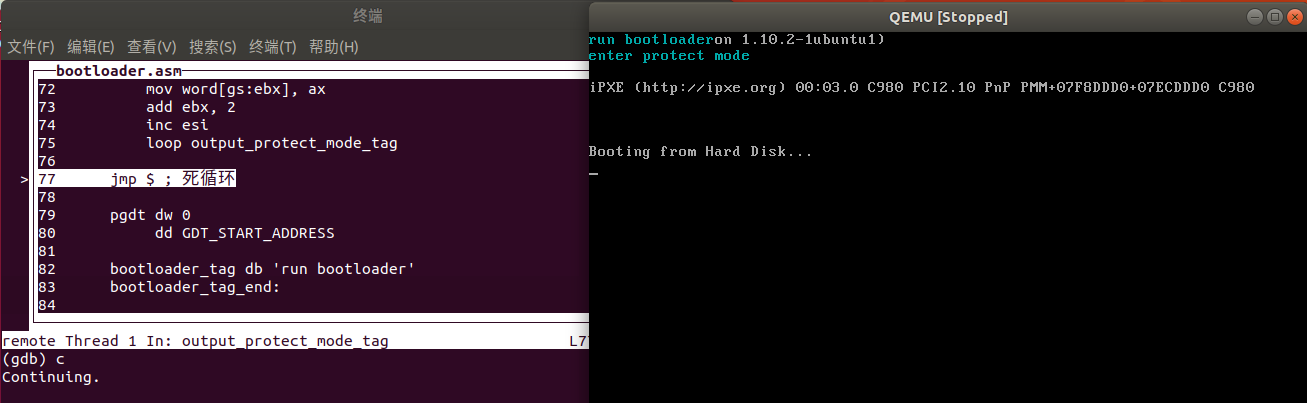
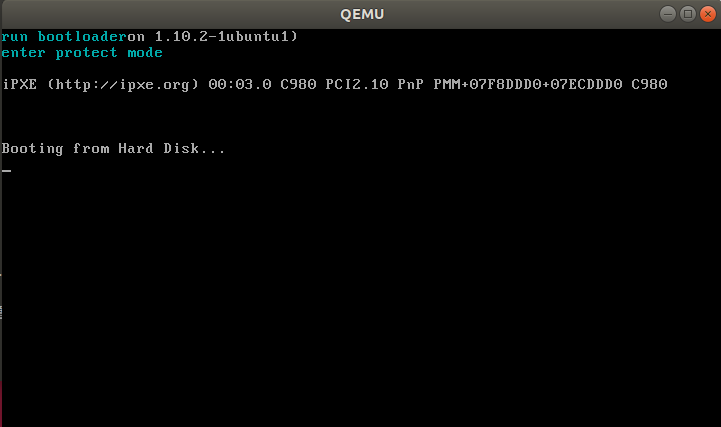


图 5 执行远跳转

* 实验结果展示：通过执行前述代码，可得下图结果，



------------------------- **实验任务4** -------------------------

* 任务要求：在进入保护模式后，按照如下要求，编写并执行一个自己定义的32位汇编程序，使用两种不同的自定义颜色和一个自定义的起始位置(x,y)，使得bootloader加载后，在显示屏坐标(x,y)处开始输出自己的学号+姓名拼音首字母缩写，要求相邻字符前景色和背景色必须是相互对调的。最后附上结果截图。
* 思路分析：

①我们要在保护模式下编写32位的汇编程序，从实验2可以知道我们显示字符的原理，就是将对应的字符和对应的颜色属性都送入到显存中0xB8000~0xBFFFF处就能够在显示屏上显示字符了，由于要求我们在保护模式下编写程序，从实验任务3知道，保护模式下的视频段描述符也就是显存段就是我们要存入字符数据的目标，所以我们只需要将我们的学号和名字字符依次放入显存段即可实现字符显示

②对于前景色和背景色的交替显示，我们可以使用移位和或命令来以及两个中间寄存器来实现，前四位数和后四位数的转换，同时我们使用字符串的方式来储存我们的字符，方便我们使用循环的方式打印字符，简化代码的长度。

* 实验步骤：

**Step1：设置字符串变量以及颜色变量和初始的位置坐标**，这里我们将初始的坐标设置为(12，12)，对应的初始显存地址为(80\*12+12) \*2.实现代码如下：

mov ecx, name\_and\_student\_number\_end - name\_and\_student\_number

mov ebx, (80\*12+12) \* 2 ;起始位置(12,12)

mov esi, name\_and\_student\_number ;字符串首地址

color db 0x73 ;0111\_0011白色和青色

name\_and\_student\_number db '23320104ltx';学号和名字

name\_and\_student\_number\_end:

**Step2：实现前景色和背景色的互换**，首先我们使用与操作分别提取出颜色的后4位和前4位，分别储存到al，bl，然后使al左移4位移动到高位，使bl右移4位移动到低位，然后使用或操作将al，bl连接，实现互换。

output\_loop:

pushad

mov al,[color]

and al,0x0F ;低4位

shl al,4

mov bl,[color]

and bl,0xF0 ;高4位

shr bl,4

or al,bl

mov [color],al

popad

**Step3：将颜色和字符写入到对应的显存位置**，将颜色存入到ax的高8位将字符存入到ax低8位，同时使得ebx增加到下一个字符的内存位置，使得esi自加1，指向下一个字符，代码实现如下

mov ah,[color]

mov al,[esi]

mov word[gs:ebx],ax

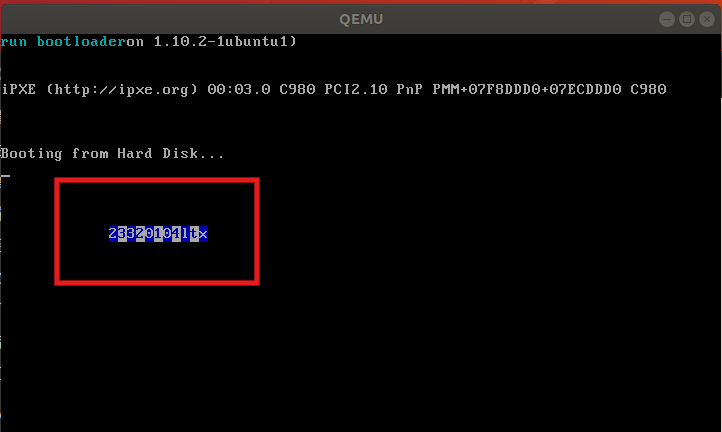
add ebx,2

inc esi

loop output\_loop

jmp $ ; 死循环

* 实验结果展示：通过执行前述代码，可得下图结果，



**Section 5 实验总结与心得体会**

通过本次 “从实模式到保护模式” 的实验中学习到了磁盘的I/O操作，同时进一步体会了计算机的启动过程，以及进一步提高了使用gdb以及qemu对所写的程序进行调试的技巧的方法，同时学习到了如何突破512B的限制，加载我们自己所写的bootloader和如何从16位的实模式进入到32位的保护模式下。

在加载bootloader的实验中，使我理解了计算机如何与外部存储设备交互数据，深入学习和探究了LBA和CHS两种磁盘读写的方式，同时还通过硬盘结构推到出了LBA和CHS的转换公式，同时通过编写程序的方式验证了转换公式的正确性。

在进入保护模式代码的编写中，学习到了段描述符、段选择子等等很多新的概念，同时还以进入保护模式的四个步骤为对象，使用gdb对这四个步骤进行了调试，通过设置断点、查看代码运行前后的寄存器、变量的值、单步运行调试等命令，不仅提高了代码调试能力，同时进一个加深了进入保护模式的步骤和个步代码的作用。同时，在调试的过程中，为了方便调试学习到了makefile文件的使用，通过编写makefile文件和使用make指令，大大提高了调试代码、运行代码的效率。

总之，通过本次实验我亲手编写和调试从实模式切换到保护模式的代码，提高了对x86汇编语言的运用能力，了解了硬盘的I/O操作的CHS模式和LBA模式，理解了计算机如何与外部存储设备交互数据，同时进一步学习debug技巧，提高了代码调试能力。

**Section 6 附录：参考资料清单**

1. [C/H/S与LBA的转换关系\_lba柱面换算-CSDN博客](https://blog.csdn.net/G_Spider/article/details/6906184)
2. [INT 13H磁盘服务详解-CSDN博客](https://blog.csdn.net/brainkick/article/details/7583727)
3. [Makefile教程：Makefile文件编写1天入门 - C语言中文网](https://c.biancheng.net/makefile/)

**Section 7 附录：代码清单**

完整的代码文件夹在Code文件夹中，为了篇幅简单，这里只放本次实验报告中所涉及到的实验要求自己编写的关键代码

【实验任务1】完整的mbr.asm程序如下：

org 0x7c00

[bits 16]

xor ax, ax ; eax = 0

; 初始化段寄存器, 段地址全部设为0

mov ds, ax

mov ss, ax

mov es, ax

mov fs, ax

mov gs, ax

; 初始化栈指针

mov sp, 0x7c00

mov ax, 1                ; 逻辑扇区号第0~15位

mov cx, 0                ; 逻辑扇区号第16~31位

mov bx, 0x7e00           ; bootloader的加载地址

load\_bootloader:

    call asm\_read\_hard\_disk  ; 读取硬盘

    inc ax

    cmp ax, 5

    jle load\_bootloader

jmp 0x0000:0x7e00        ; 跳转到bootloader

jmp $ ; 死循环

asm\_read\_hard\_disk:

    mov dx, 0x1f3

    out dx, al    ; LBA地址7~0

    inc dx        ; 0x1f4

    mov al, ah

    out dx, al    ; LBA地址15~8

    mov ax, cx

    inc dx        ; 0x1f5

    out dx, al    ; LBA地址23~16

    inc dx        ; 0x1f6

    mov al, ah

    and al, 0x0f

    or al, 0xe0   ; LBA地址27~24

    out dx, al

    mov dx, 0x1f2

    mov al, 1

    out dx, al   ; 读取1个扇区

    mov dx, 0x1f7    ; 0x1f7

    mov al, 0x20     ;读命令

    out dx,al

    ; 等待处理其他操作

  .waits:

    in al, dx        ; dx = 0x1f7

    and al,0x88

    cmp al,0x08

    jnz .waits

    ; 读取512字节到地址ds:bx

    mov cx, 256 ;每次读取一个字2个字节因此读取256次

    mov dx, 0x1f0

  .readw:

    in ax, dx

    mov [bx], ax

    add bx, 2

    loop .readw

    ret

times 510 - ($ - $$) db 0

db 0x55, 0xaa

【实验任务1】完整的bootloader.asm程序如下：

org 0x7e00

[bits 16]

mov ax, 0xb800

mov gs, ax

mov ah, 0x03 ;青色

mov ecx, bootloader\_tag\_end - bootloader\_tag

xor ebx, ebx

mov esi, bootloader\_tag

output\_bootloader\_tag:

    mov al, [esi]

    mov word[gs:bx], ax

    inc esi

    add ebx,2

    loop output\_bootloader\_tag

jmp $ ; 死循环

bootloader\_tag db 'run bootloader'

bootloader\_tag\_end:

【实验任务2】完整的读扇区程序其他和任务1一样如下：

asm\_read\_hard\_disk:

    pushad

    lba dw 0    ;LBA

    c dw 0      ;柱面号c

    h dw 0      ;磁头号h

    s dw 0      ;磁区号s

    mov [lba], ax

;计算柱面号 c= LBA // (18\*63)

    mov ax,[lba]

    mov cx,1134 ;18\*63=1134

    xor dx,dx   ;被除数dx:ax

    div cx      ;ax为商

    mov [c],al

;计算磁头号 h= (LBA // 63) % 18

    mov ax,[lba]

    mov cx,63

    xor dx,dx

    div cx      ;ax为LBA/63的商即LBA整除63的结果

    mov cx,18

    xor dx,dx

    div cx      ;除以18,dx为(LBA//63)mod 18的结果

    mov [h],dl

;计算扇区号 s= LBA % 63

    mov ax,[lba]

    mov cx,63

    xor dx,dx

    div cx      ;LBA mod 63 结果存放在dx即余数

    inc dx      ;扇区号从1开始所以要加1

    mov [s],dl

;调用BIOS

    mov ch,[c]  ;c

    mov dh,[h]  ;h

    mov cl,[s]  ;s

    mov al,5    ;读连续5块扇区

    mov ah,02H  ;功能号02H

    mov dl,80H

    int 13H

    popad

    ret

times 510 - ($ - $$) db 0

db 0x55, 0xaa

【实验任务3】完整的bootloader.asm程序如下：

%include "boot.inc"

org 0x7e00

[bits 16]

mov ax, 0xb800

mov gs, ax

mov ah, 0x03 ;青色

mov ecx, bootloader\_tag\_end - bootloader\_tag

xor ebx, ebx

mov esi, bootloader\_tag

output\_bootloader\_tag:

    mov al, [esi]

    mov word[gs:bx], ax

    inc esi

    add ebx,2

    loop output\_bootloader\_tag

;空描述符

mov dword [GDT\_START\_ADDRESS+0x00],0x00

mov dword [GDT\_START\_ADDRESS+0x04],0x00

;创建描述符，这是一个数据段，对应0~4GB的线性地址空间

mov dword [GDT\_START\_ADDRESS+0x08],0x0000ffff

mov dword [GDT\_START\_ADDRESS+0x0c],0x00cf9200

;建立保护模式下的堆栈段描述符

mov dword [GDT\_START\_ADDRESS+0x10],0x00000000

mov dword [GDT\_START\_ADDRESS+0x14],0x00409600

mov dword [GDT\_START\_ADDRESS+0x18],0x80007fff

mov dword [GDT\_START\_ADDRESS+0x1c],0x0040920b

;创建保护模式下平坦模式代码段描述符

mov dword [GDT\_START\_ADDRESS+0x20],0x0000ffff

mov dword [GDT\_START\_ADDRESS+0x24],0x00cf9800

;初始化描述符表寄存器GDTR

mov word [pgdt], 39      ;描述符表的界限

lgdt [pgdt]

in al,0x92                         ;南桥芯片内的端口

or al,0000\_0010B

out 0x92,al                        ;打开A20

cli                                ;中断机制尚未工作

mov eax,cr0

or eax,1

mov cr0,eax                        ;设置PE位

;以下进入保护模式

jmp dword CODE\_SELECTOR:protect\_mode\_begin

[bits 32]

protect\_mode\_begin:

mov eax, DATA\_SELECTOR           ;加载数据段(0..4GB)选择子

mov ds, eax

mov es, eax

mov eax, STACK\_SELECTOR

mov ss, eax

mov eax, VIDEO\_SELECTOR

mov gs, eax

mov ecx, protect\_mode\_tag\_end - protect\_mode\_tag

mov ebx, 80 \* 2

mov esi, protect\_mode\_tag

mov ah, 0x3

output\_protect\_mode\_tag:

    mov al, [esi]

    mov word[gs:ebx], ax

    add ebx, 2

    inc esi

    loop output\_protect\_mode\_tag

jmp $ ; 死循环

pgdt dw 0

     dd GDT\_START\_ADDRESS

bootloader\_tag db 'run bootloader'

bootloader\_tag\_end:

protect\_mode\_tag db 'enter protect mode'

protect\_mode\_tag\_end:

【实验任务4】完整的字符输出代码程序如下：

mov ecx, name\_and\_student\_number\_end - name\_and\_student\_number

mov ebx, (80\*12+12) \* 2             ;起始位置(12,12)

mov esi, name\_and\_student\_number    ;字符串首地址

output\_loop:

    pushad

    mov al,[color]

    and al,0x0F     ;低4位

    shl al,4

    mov bl,[color]

    and bl,0xF0     ;高4位

    shr bl,4

    or al,bl

    mov [color],al

    popad

    mov ah,[color]

    mov al,[esi]

    mov word[gs:ebx],ax

    add ebx,2

    inc esi

    loop output\_loop

jmp $ ; 死循环

color db 0x73                 0111\_0011b白色和青色

name\_and\_student\_number db '23320104ltx'   ;学号和名字 name\_and\_student\_number\_end: