LAB3 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: 从实模式保护模式

专业名称: 计算机科学与技术

学生姓名: 马福泉

学生学号: 23336179

实验地点: 实验中心 B202

实验时间:2025.3.30

Section 1 实验概述

在上一节中，同学们已经学习了 x86 汇编的相关内容、初步了解计算机的启动过程并在 16 位的实模式环境下进行编程。

在本节中，同学们将会学习到如何从 16 位的实模式跳转到 32 位的保护模式，然后在平坦模式下运行 32 位程序。同时，同学们将学习到如何使用 I/O 端口和硬件交互，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：汇编语言基础，计算机体系结构，操作系统基础，调试工具的使用（如 gdb）等。

- 实验环境：
  - 虚拟机版本/处理器型号：VMware-Ubuntu22.04.5/i386（32 位）
  - 代码编辑环境： 编辑器：VSCode
    插件：C/C++插件，汇编插件
  - 代码编译工具： 编译器：gcc
    工具链：binutils、make、qemu、nasm 等
    调试工具：gdb
  - 重要三方库信息：GNU binutils：工具集（如 as、ld）
    gdb：调试工具
    QEMU、Bochs：模拟器用于测试 等

Section 3 实验任务

实验任务 1：课后思考题第 9 题，复现“加载 bootloader”这一节，说说你是怎么做的并提供结果截图，也可以参考 Ucore、Xv6 等系统源码，实现自己的 LBA 方式的磁盘访问。

实验任务 2：课后思考题 10:在"加载 bootloader"一节中，我们使用了 LBA28 的方式来读取硬盘。此时，我们只要给出逻辑扇区号即可，但需要手动去读取 I/O 端口。然而，BIOS 提供了实模式下读取硬盘的中断，其不需要关心具体的 I/O 端口，只需要给出逻辑扇区号对应的磁头（Heads）、扇区（Sectors）和柱面（Cylinder）即可，又被称为 CHS 模式。现在，同学们需要将 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。最后说说你是怎么做的并提供结果截图。

实验任务 3：课后思考题 11 复现“进入保护模式”一节，使用 gdb 或其他 debug 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。gdb 的使用可以参考 appendix 的“debug with gdb and qemu”部分。

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，要求简单说一说你的实现思路，并提供结果截图。 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。
- 思路分析：将 lab2 中输出 Hello World 部份的代码放入到 bootloader 中，然后在 MBR 中加载 bootloader 到内存，并跳转到 bootloader 的起始地址执行。
- 实验步骤：

1. 创建硬盘镜像文件

创建一个虚拟硬盘镜像文件 hd.img，用于模拟硬盘。

```
qemu-img create hd.img 10M
```

2.按实验指导书编写 bootloader.asm ，并编译为二进制文件 bootloader.bin。

```
nasm -f bin bootloader.asm -o bootloader.bin
```

3. 将 Bootloader 写入硬盘镜像文件(bootloader.asm)

将 bootloader.bin 写入硬盘镜像文件的第 2 个扇区（编号为 1），共 5 个扇区。

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

4. 编写 MBR 代码（mbr.asm）

按实验指导书编写 mbr.asm ，并编译为二进制文件 mbr.bin。

```
nasm -f bin mbr.asm -o mbr.bin
```

5. 将 MBR 写入硬盘镜像文件

将 mbr.bin 写入硬盘镜像文件的首扇区。

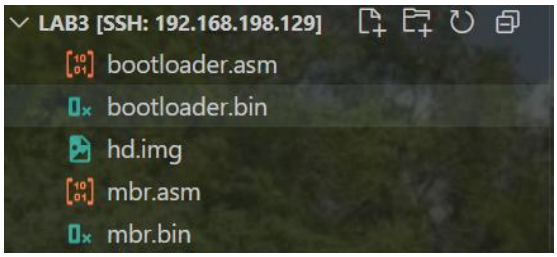
```
dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

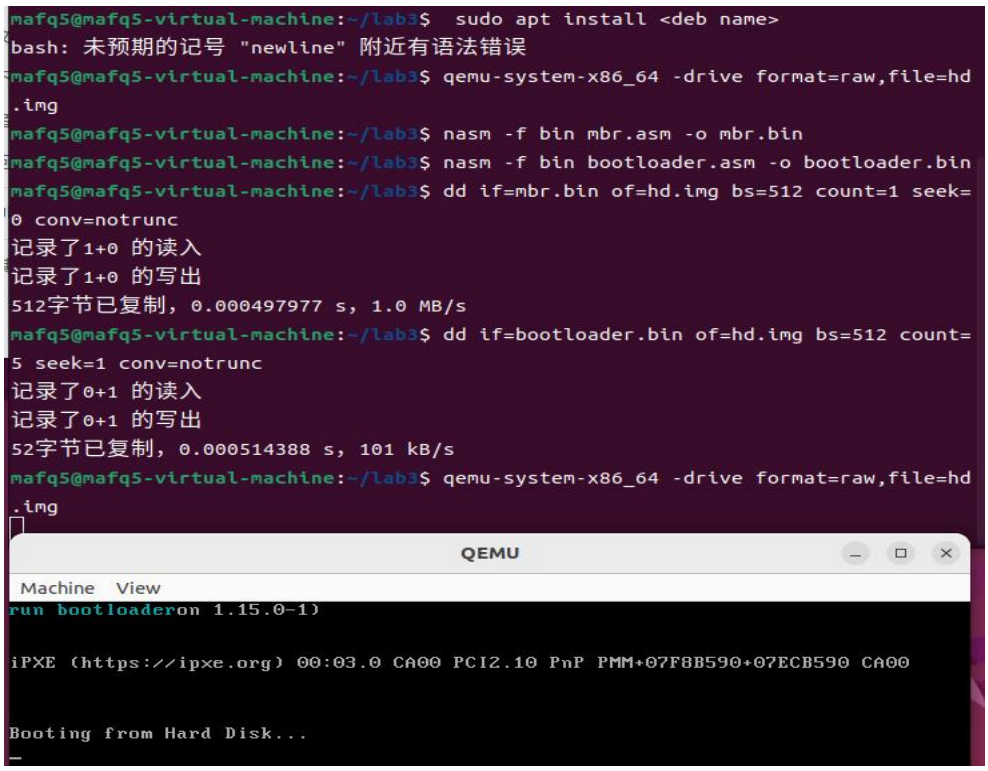
6. 运行虚拟机

使用 QEMU 运行硬盘镜像文件，验证 MBR 和 Bootloader 是否正常工作。

```
qemu-system-x86_64 -drive format=raw,file=hd.img
```

- 实验结果展示：代码与实验指导书一样，已放在 code 文件夹，不再赘述，以下是目录以及运行结果





---------- 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. 将 MBR 中的 LBA28 读取方式替换为 CHS 读取方式。
  2. 使用 BIOS 中断 INT 13H 功能号 02H 来读取硬盘扇区
  3. 和实验任务一一样编译运行
- 实验步骤：

1. LBA 到 CHS 的转换公式

LBA（逻辑块地址）和 CHS（柱面/磁头/扇区）是两种不同的硬盘寻址方式。LBA 是线性寻址，而 CHS 是基于物理结构的寻址。转换公式如下：

(每磁道扇区数（SPT，Sectors Per Track）为 63。

每柱面磁头数（HPC，Heads Per Cylinder）为 18。)

转换公式:

1. **计算柱面（Cylinder）**：

$$\text{Cylinder} = \frac{\text{LBA}}{\text{SPT} \times \text{HPC}}$$

2. **计算磁头（Head）**：

$$\text{Head} = \frac{\text{LBA}}{\text{SPT}} \mod \text{HPC}$$

3. **计算扇区（Sector）**：

$$\text{Sector} = (\text{LBA} \mod \text{SPT}) + 1$$

2.编写的核心的代码如下：（具体见 mbr_CHS.asm）。

将 MBR 中的 LBA28 读取方式替换为 CHS 读取方式。

```
; LBA到CHS的转换子程序
lba_to_chs:
    ; 逻辑扇区号向CHS的转换公式:
    ; 扇区号 S = (逻辑扇区号 L % 63 (每磁道的扇区数 SPT)) + 1
    ; 磁头号 H = (L / 63) % 18 （每柱面的磁头数）
    ; 柱面号 C = (L / 63) / 18

    ; 每磁道扇区数 (SPT) = 63
    ; 每柱面磁头数 (HPC) = 18

    ; 计算扇区号 S
    xor dx, dx                ; 清零 DX
    mov si, 63                ; SPT = 63
    div si                    ; AX / SPT，商在 AX，余数在 DX
    inc dx                    ; 扇区号 S = (L % 63) + 1
    mov byte [sector], dl     ; 保存扇区号

    ; 计算磁头号 H
    xor dx, dx                ; 清零 DX
    mov di, 18                ; HPC = 18
    div di                    ; AX / HPC，商在 AX，余数在 DX
    mov byte [head], dl       ; 磁头号 H = (L / 63) % 18

    ; 计算柱面号 C
    mov byte [cylinder], al   ; 柱面号 C = (L / 63) / 18

    ret
```

使用 BIOS 中断 INT 13H 功能号 02H 来读取硬盘扇区

```
    ; 调用BIOS读取磁盘中断
    int 0x13
    jc load_error             ; 如果进位标志CF=1，说明读取出错

    ; 读取成功，跳转到bootloader
    jmp 0x0000:0x7e00

load_error:
    ; 显示出错信息
    mov si, error_msg
    call print_string
    jmp $                     ; 无限循环
```

3. 编译和写入硬盘镜像

编译 MBR 和 Bootloader，并将它们写入硬盘镜像文件。

*我们可以编写 Makefile,方便实验，如下：

```
build1:
    nasm -f bin mbr.asm -o mbr.bin
    nasm -f bin bootloader.asm -o bootloader.bin
    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
build2:
    nasm -f bin mbr_CHS.asm -o mbr_CHS.bin
    nasm -f bin bootloader.asm -o bootloader.bin
    dd if=mbr_CHS.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
run:
    qemu-system-i386 -hda hd.img -serial null -parallel stdio
clean:
    rm -fr *.bin
```

4. 运行虚拟机

使用 QEMU 运行硬盘镜像文件，验证 MBR 和 Bootloader 是否正常工作。

```
    qemu-system-x86_64 -drive format=raw,file=hd.img
或者直接
    make clean
    make build2
    make run
```

- 实验结果展示：

```
mafq5@mafq5-virtual-machine:~/lab3$ make build2
nasm -f bin CHS.asm -o CHS.bin
nasm -f bin bootloader.asm -o bootloader.bin
dd if=CHS.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
记录了1+0 的读入
记录了1+0 的写出
512字节已复制，0.000660826 s，775 kB/s
dd if=bootloader.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
记录了0+1 的读入
记录了0+1 的写出
57字节已复制，0.000935463 s，60.9 kB/s
mafq5@mafq5-virtual-machine:~/lab3$ make run2
qemu-system-i386 -hda hd.img -serial null -parallel stdio
WARNING: Image format was not specified for 'hd.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
```

```
QEMU - Press Ctrl+Alt+G to release grab
Machine  View
run bootloader(CHS)15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B590+07ECB590 CA00


Booting from Hard Disk...
```

---------- 实验任务 3 ----------

- 任务要求：复现“进入保护模式”一节，使用 gdb 或其他 debug 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。
- 思路分析：准备 GDT（全局描述符表）。打开 A20 地址线。设置 CR0 寄存器的 PE（保护模式使能）位。执行远跳转进入保护模式。在保护模式下输出 "protect mode"
- 实验步骤：

1. 按实验指导书编写 bootloader_protect.asm, boot.inc, mbr_protect.asm.

   编写 makefile,编译运行

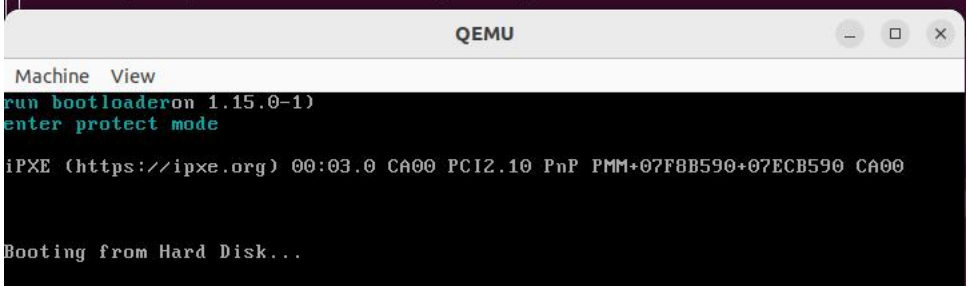
```
build1:
    nasm -f bin mbr.asm -o mbr.bin
    nasm -f bin bootloader.asm -o bootloader.bin
    dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
    dd if=bootloader.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
build2:
    nasm -f bin mbr_CHS.asm -o mbr_CHS.bin
    nasm -f bin bootloader.asm -o bootloader.bin
    dd if=mbr_CHS.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
build3:
    nasm -f bin mbr_protect.asm -o mbr_protect.bin
    nasm -f bin bootloader_protect.asm -o bootloader_protect.bin
    dd if=mbr_protect.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
    dd if=bootloader_protect.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
run:
    qemu-system-i386 -hda hd.img -serial null -parallel stdio
clean:
    rm -fr *.bin
```

```
make clean
make build3
make run
```

运行结果如下：

成功进入保护模式

```
mafq5@mafq5-virtual-machine:~/lab3$ make build3
nasm -f bin mbr_protect.asm -o mbr_protect.bin
nasm -f bin bootloader_protect.asm -o bootloader_protect.bin
dd if=mbr_protect.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
记录了1+0 的读入
记录了1+0 的写出
512字节已复制，0.00168067 s，305 kB/s
dd if=bootloader_protect.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
记录了0+1 的读入
记录了0+1 的写出
254字节已复制，0.00035171 s，722 kB/s
mafq5@mafq5-virtual-machine:~/lab3$ make run
qemu-system-i386 -hda hd.img -serial null -parallel stdio
WARNING: Image format was not specified for 'hd.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
```



2. 使用 gdb 进行 debug：

```
qemu-system-x86_64 -drive format=raw,file=hd.img
新窗口输入 gdb
```

也可以创建 makefile 文件和 gdbinit 文件，把命令整合，如下：

task3 > makefile

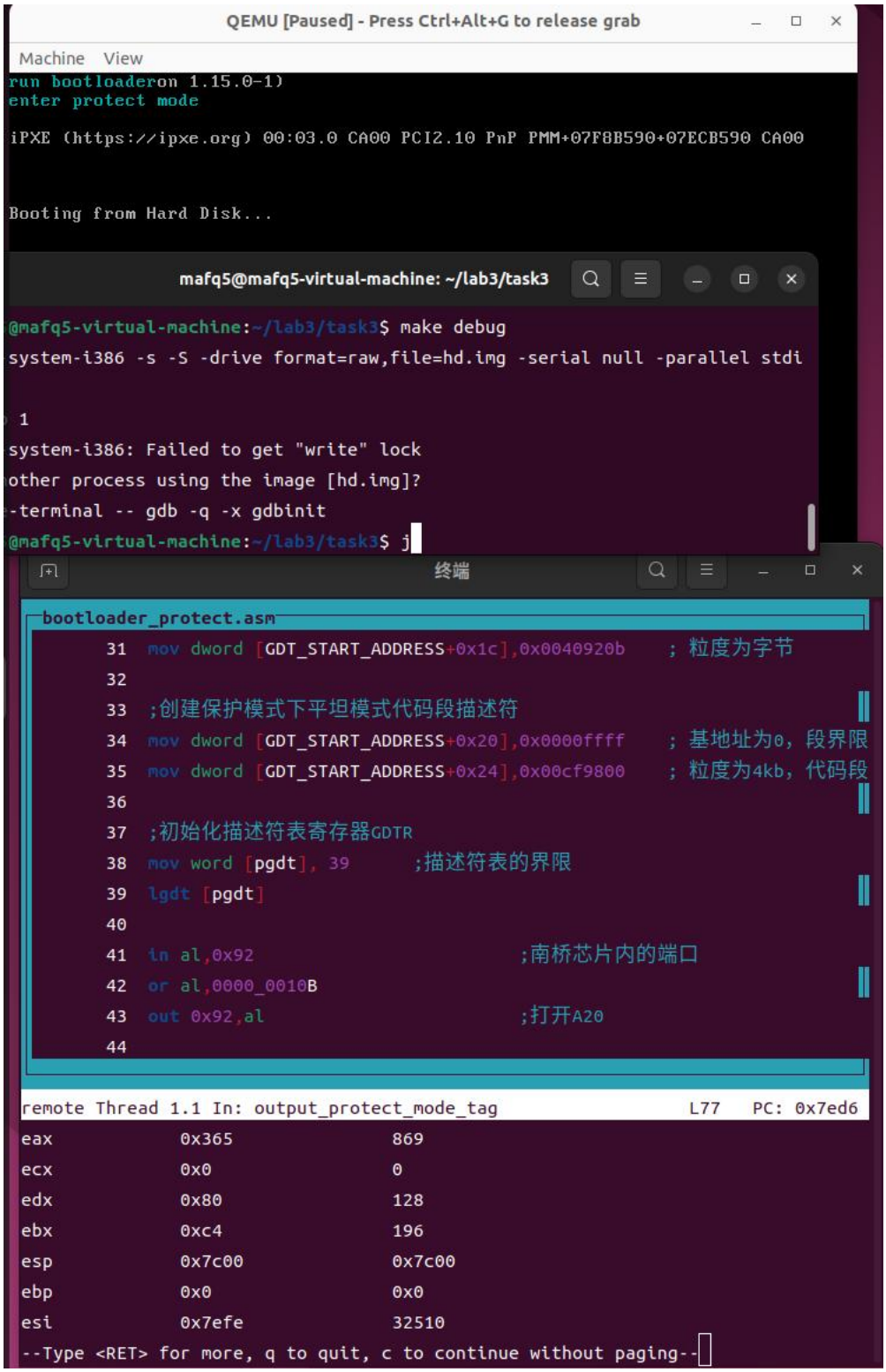
```
1   debug:
2       qemu-system-i386 -s -S -drive format=raw,file=hd.img -serial null -parallel stdio &
3       sleep 1
4       gnome-terminal -- gdb -q -x gdbinit
5   build:
6       nasm -g -f elf32 mbr_protect.asm -o mbr_protect.o
7       ld -o mbr_protect.symbol -melf_i386 -N mbr_protect.o -Ttext 0x7c00
8       ld -o mbr_protect.bin -melf_i386 -N mbr_protect.o -Ttext 0x7c00 --oformat binary
9       nasm -g -f elf32 bootloader_protect.asm -o bootloader_protect.o
10      ld -o bootloader_protect.symbol -melf_i386 -N bootloader_protect.o -Ttext 0x7e00
11      ld -o bootloader_protect.bin -melf_i386 -N bootloader_protect.o -Ttext 0x7e00 --oformat binary
12      dd if=mbr_protect.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
13      dd if=bootloader_protect.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
14  clean:
15      rm -fr *.bin *.o
```

task3 > gdbinit

```
1   target remote:1234
2   set disassembly-flavor intel
3   add-symbol-file mbr_protect.symbol 0x7c00
4   add-symbol-file bootloader_protect.symbol 0x7e00
```

```
make build
make debug
layout src
```

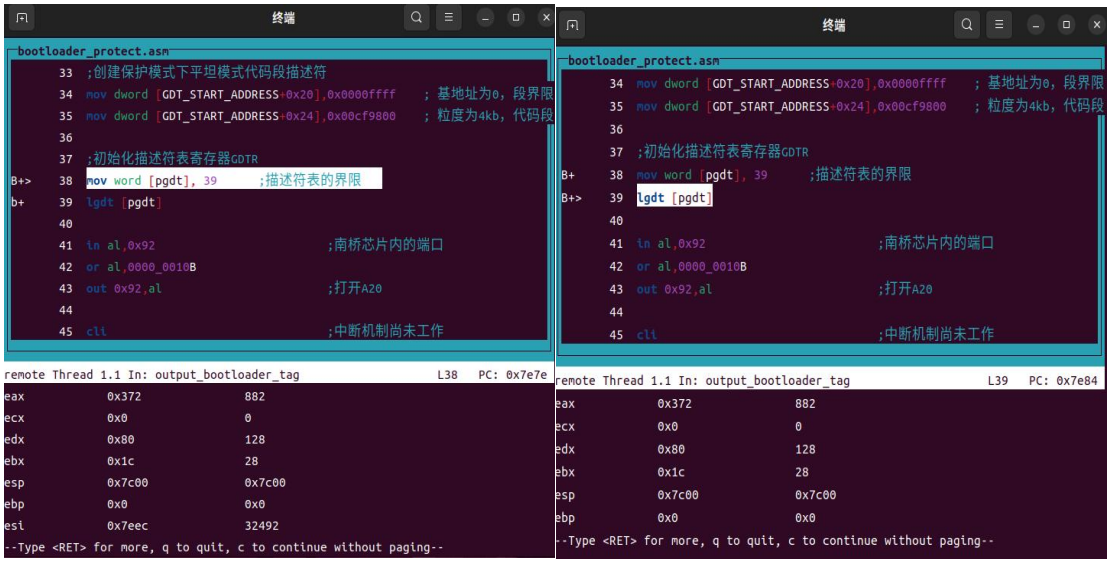
```
list
info registers
```



3. 接下来我们将分别检验保护模式的四大步骤：（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

（1）用 lgdt 加载 GDTR 信息指令与前后的寄存器数据如下：

```
make build
make debug
b *0x7e00
layout src
list
c
b 38
b 39
c
info registers
C
Info registers
```

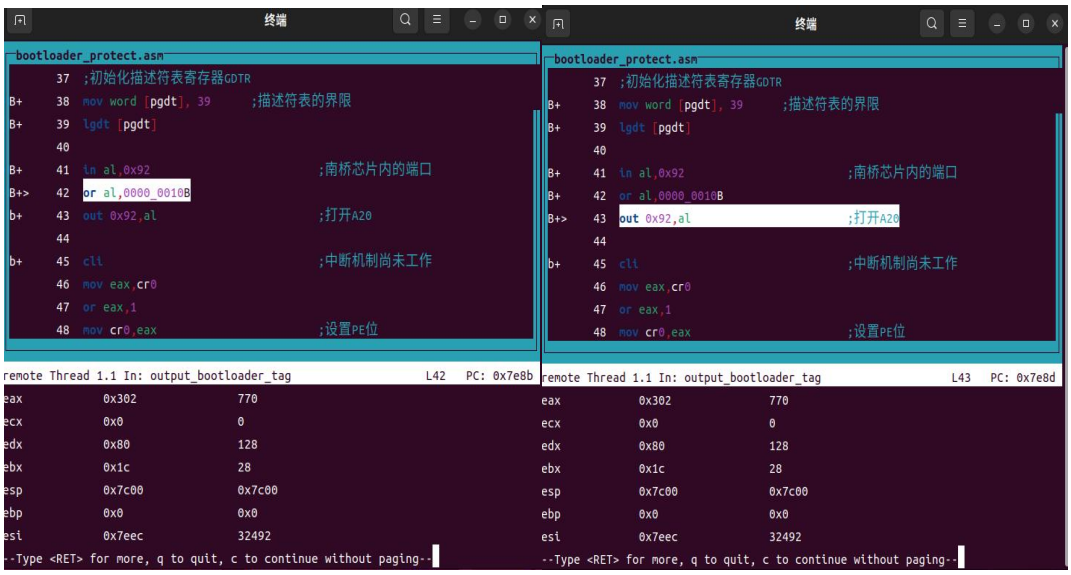


各个寄存器的值并非发生变化。

(2) 打开第 21 根地址线：这一步骤一共有三条指令，其中前两天指令执行后的寄存器数据分别如下：（指令就是断点，运行，查看寄存器，不再赘述）

在执行 in al,0x92 之后，查看 AL 寄存器的值为 0x02(eax 低八位）。此值是从端口 0x92 读取的原始数据。

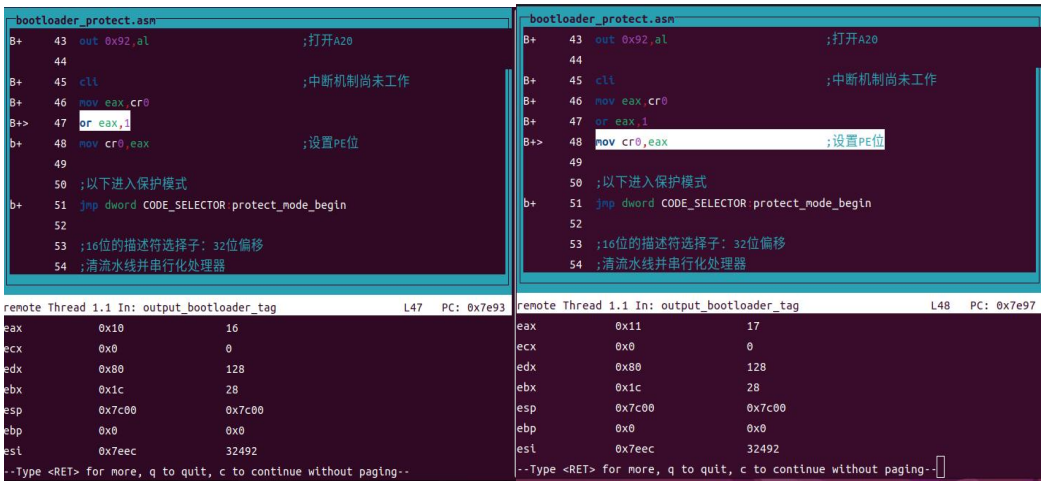
执行 or al,0000_0010B 之后，再次查看 AL 寄存器的值为 0x02(eax 低八位），确认第 1 位（从右往左数，从 0 开始）被置为 1



(3) 开启 cr0 的保护模式标志位，如下图

在执行 mov eax,cr0 之后，查看 EAX 寄存器的值，记录初始的 CR0 寄存器值。

执行 or eax,1 之后，再次查看 EAX 寄存器的值，确认最低位（PE 位）被置 1



(4) 远跳转：远跳转后，系统会进入保护模式，这里我们选择保护模式下的其中一条指令设置断点，查看几个关键寄存器：

CR0 寄存器

检查 CR0 的 PE 位是否被设置为 1，以确认系统已经进入保护模式。

EIP 寄存器

应该指向保护模式下的代码地址（例如 protect_mode_start）。

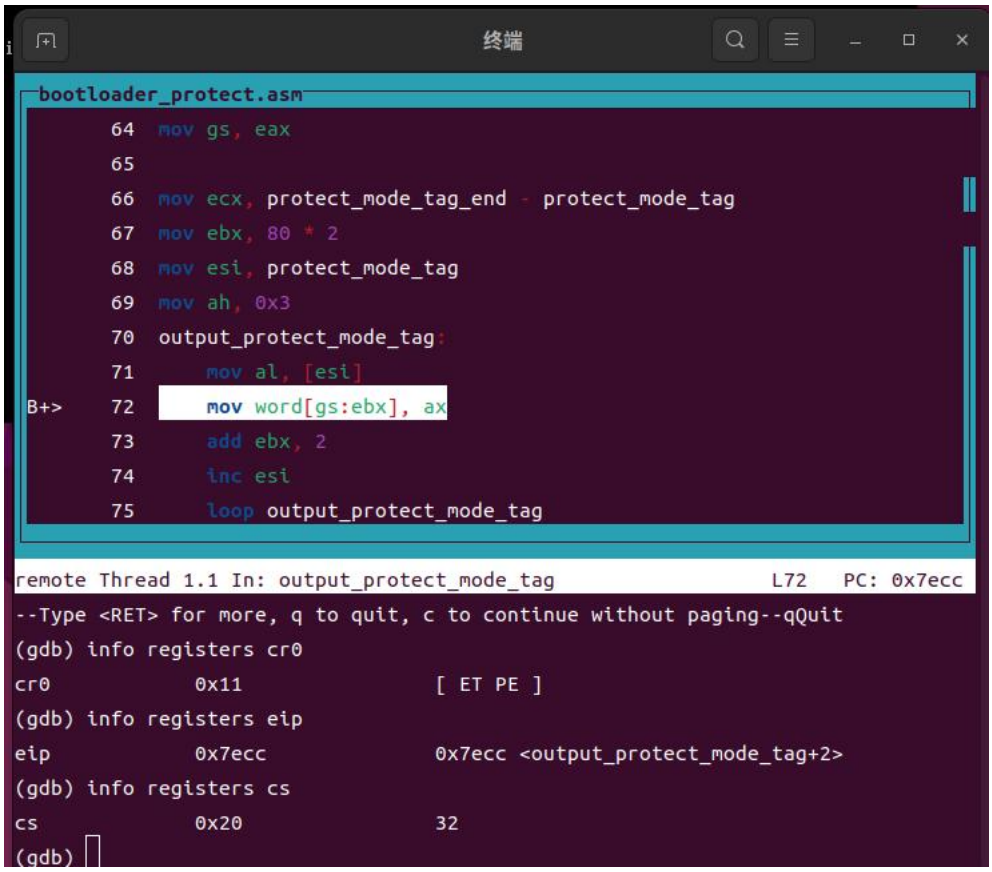
EIP 的值与设置的断点地址一致。

CS 寄存器

应该是保护模式下的代码段选择子（例如 0x08）。

CS 的值与远跳转指令中的选择子一致。

如下图所示：



- 实验结果展示：所有结果已经展示在对应步骤中

---------- 实验任务 4 ----------

- 任务要求：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，要求简单说一说你的实现思路，并提供结果截图。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。
- 思路分析：参考实验 2 和实验 3 的 MBR 程序框架，编写 32bits.asm, mbr_32bits.asm, makefile, 然后编译运行。
- 实验步骤：

编写 32bits.asm, 与 mbr_32bits.asm,核心代码思路如下：（仅展示 32bits.asm),(mbr 代码思路差别不大)

1. 实模式阶段 (16 位)

从 0x7e00 地址加载,显示初始消息, GDT 初始化

2. 实模式到保护模式的切换

切换过程的三个关键步骤：

打开 A20 地址线：通过端口 0x92 操作

```
in al,0x92
or al,0000_0010B
out 0x92,al
```

关闭中断并设置 CR0 寄存器 PE 位：

```
cli
mov eax,cr0
or eax,1
mov cr0,eax
```

远跳转到保护模式代码段：

jmp dword CODE_SELECTOR:protect_mode_begin

3. 保护模式阶段 (32 位)

初始化段寄存器：加载选择子到各个段寄存器

显示进入保护模式的消息：显示"Now in Protected Mode"

显示学号和姓名：用交替的颜色（青色底白色字和白色底青色字）输出"23336179MFQ"

```
82  ; 显示学号和姓名（交替颜色）
83  ; 起始位置设置在屏幕中央，第10行，第30列
84  mov ebx, (10 * 80 + 30) * 2  ; 计算起始位置
85  mov esi, student_info
86  mov ecx, student_info_end - student_info
87  mov ah, 0x3F    ; 青色底白色字(第一种颜色 - 0x3为青色背景，0xF为白色文字)
88
89  output_student_info:
90      mov al, [esi]
91      mov word[gs:ebx], ax
92      xor ah, 0xCC ; 将颜色从青色底白色字(0x3F)切换到白色底青色字(0xF3)
93      add ebx, 2
94      inc esi
95      loop output_student_info
96
97  jmp $ ; 死循环
```

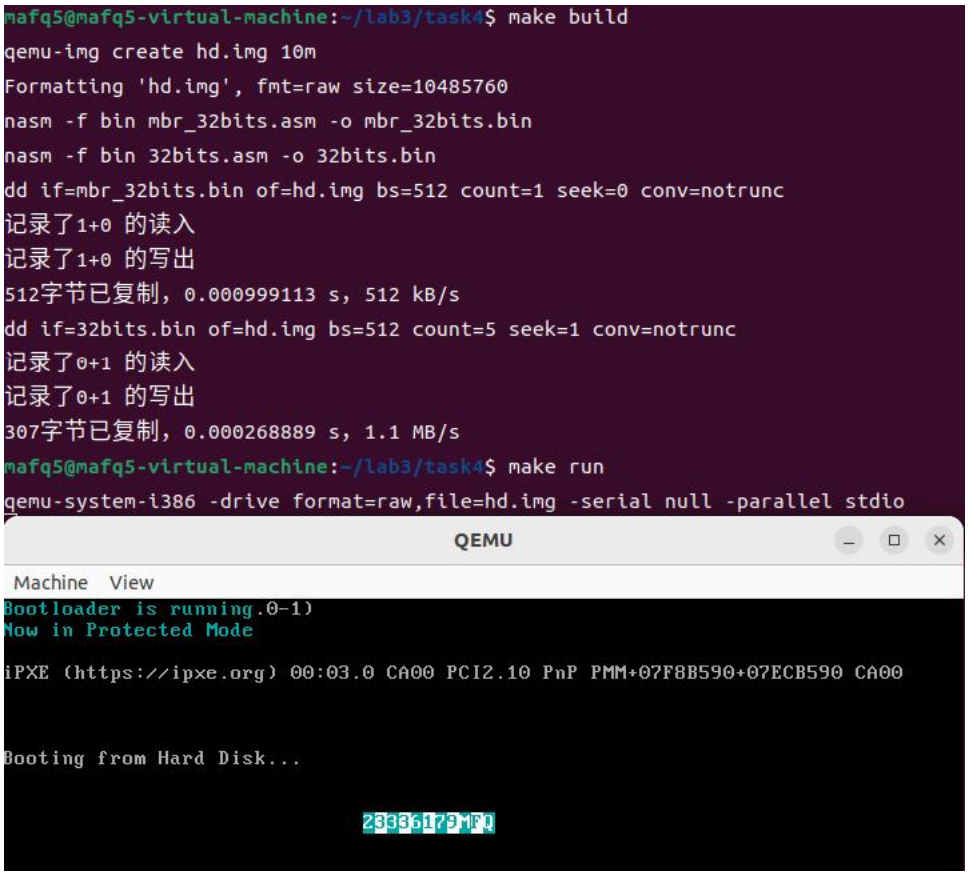
编写 makefile，如下

```
build:
    qemu-img create hd.img 10m
    nasm -f bin mbr_32bits.asm -o mbr_32bits.bin
    nasm -f bin 32bits.asm -o 32bits.bin
    dd if=mbr_32bits.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
    dd if=32bits.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
run:
    qemu-system-i386 -drive format=raw,file=hd.img -serial null -parallel stdio
clean:
    rm -fr *.bin
```

编译运行

```
make build
make run
```

- 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

1. 注意 vscode 编码问题

确保文件编码为 UTF-8 无 BOM。在设置处更改。

或者可以使用文本编辑器（如 vim、nano 或其他支持无 BOM 的编辑器）重新保存文件。

2. LBA 与 CHS 对比

1） 简化编程复杂性

直接操作硬件：在 LBA 模式下，程序需要直接与硬盘的 I/O 端口交互，计算 LBA 地址并将其拆分为多个部分写入不同的端口。这涉及到对硬盘硬件的直接操作，编程复杂且容易出错。

使用 BIOS 中断：BIOS 提供的 INT 13H 中断服务是一个标准的磁盘操作接口，功能号 02H 用于读取硬盘扇区。通过使用 INT 13H，程序只需要提供柱面（Cylinder）、磁头（Head）和扇区（Sector）号，BIOS 会自动处理底层的硬件操作。这种方式大大简化了编程复杂性。

2） 兼容性

旧系统支持：许多旧的系统和工具仍然使用 CHS 模式进行硬盘操作。使用 INT 13H 功能号 02H 可以确保在这些旧系统上的兼容性。

标准接口：BIOS 提供的 INT 13H 中断服务是标准的磁盘操作接口，广泛用于实模式下的磁盘操作。使用这些标准接口可以确保代码在不同系统上的可移植性。

3） 实模式下的限制

实模式：在实模式下，程序可以直接访问 BIOS 提供的中断服务。INT 13H 功能号 02H 是 BIOS 提供的标准磁盘读取服务，非常适合在实模式下使用。

保护模式：在保护模式下，程序通常需要直接操作硬件或使用操作系统提供的磁盘操作接口。但在实模式下，使用 BIOS 中断是最简单和最直接的方法。

3. 注意：之前就犯过的错：为了使得 gdb 能够找到 debug 信息，我们需要生成符号表。注意，生成符号表的时候需要将 mbr.asm 开头的 org 伪指令删去。

4. 收获总结：

1） 理解实模式和保护模式的区别

实模式：16 位模式，地址空间受限（最大 1MB），直接操作硬件，适合简单的启动代码。

保护模式：32 位模式，地址空间扩展到 4GB，提供内存保护和分段机制，适合复杂的操作系统和应用程序。

2） 掌握 GDT 的设置和使用：

全局描述符表（GDT）：定义了段描述符，每个描述符包含段的基地址、界限和访问权限。

段描述符：用于定义代码段、数据段、栈段和显存段等。

平坦模式：通过设置基地址为 0，段界限为 4GB，简化了内存访问，使得整个 4GB 内存空间可以被直接访问。

3） 学会切换到保护模式

设置 GDTR：加载 GDT 的起始地址和大小。

打开 A20 地址线：允许访问超过 1MB 的内存。

设置 CR0 的 PE 位：启用保护模式。

远跳转：切换到保护模式下的代码。

4） 实践调试技巧

使用 QEMU 和 GDB：调试启动代码和保护模式下的代码。

查看寄存器状态：使用 info registers 查看寄存器的值。

设置断点：在关键位置设置断点，逐步调试程序。

Section 6 附录：参考资料清单

1. 部分代码 debug 与指令除了参考实验指导书，还参考了大语言模型

2. 参考文章，博客部分如下：q

[https://blog.csdn.net/brainkick/article/details/7583727]
[https://blog.csdn.net/G_Spider/article/details/6906184]
INT13H 磁盘操作详解-CSDN 博客