

一、实验要求：

- DDL：2021年3月30号 23:59
- 提交的内容：将3个assignment的代码和实验报告放到压缩包中，命名为“lab3-姓名-学号”，并交到课程网站上[\[http://course.dds-sysu.tech/course/3/homework\]](http://course.dds-sysu.tech/course/3/homework)
- 材料的Example的代码放置在 src 目录下。

1. 实验不限语言，C/C++/Rust都可以。
2. 实验不限平台，Windows、Linux和MacOS等都可以。
3. 实验不限CPU，ARM/Intel/Risc-V都可以。

二、实验内容：

Assignment 1

1.1

复现Example 1，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。

1.2

在Example1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，可以参考《于渊：一个操作系统的实现2》P183-184。

Assignment 2

复现Example 2，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”部份。

Assignment 3

改造“Lab2-Assignment 4”为32位代码，即在保护模式后执行自定义的汇编程序。

三、实验过程：

先创建一个文件夹lab3。把该实验用到的代码文件都放在lab3下。

1.1复现Example 1。

先创建一个文件 `bootloader.asm`，然后将一下代码放入 `bootloader.asm` 中，保存。

```
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:

```

然后创建 `mbr.asm` 文件，在 `mbr.asm` 使用LBA模式读取硬盘，然后在MBR中加载bootloader到地址 0x7e00。

- 1、设置起始的逻辑扇区号。逻辑扇区号的第0 ~ 15位存在寄存器ax，逻辑扇区号的16~17位存在寄存器cx。再将逻辑扇区的0 ~ 7位被写入0x1F3端口，8 ~ 15位被写入0x1F4端口，16 ~ 23位被写入0x1F5端口，最后4位被写入0x1F6端口的低4位。
- 2、将要读取的扇区数量写入0x1F2端口。
- 3、向0x1F7端口写入0x20，请求硬盘读。
- 4、等待其他读写操作完成。请求硬盘读的时候，可能硬盘在处理其他操作。因此需要等待其他读写操作完成后才能开始本次读写操作。

```

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:
; 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

    ; 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

times 510 - ($ - $$) db 0
db 0x55, 0xaa

```

运行结果:

```
azhi@azhi-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
azhi@azhi-VirtualBox:~$ cd lab3
azhi@azhi-VirtualBox:~/lab3$ nasm -f bin bootloader.asm -o bootloader.bin
azhi@azhi-VirtualBox:~/lab3$ 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
118 bytes copied, 0.00670981 s, 17.6 kB/s
azhi@azhi-VirtualBox:~/lab3$ nasm -f bin mbr.asm -o mbr.bin
azhi@azhi-VirtualBox:~/lab3$ dd if=mbr.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 bytes copied, 0.0040952 s, 125 kB/s
azhi@azhi-VirtualBox:~/lab3$ 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.
azhi@azhi-VirtualBox:~/lab3$
```

```
QEMU
run bootloaderon 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD0+07ECDDDD0 C980

Booting from Hard Disk...
```

1.2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。

以C、H、S分别表示当前硬盘的柱面号、磁头号、扇区号，CS、HS、SS分别表示起始柱面号、磁头号、扇区号，PS表示每磁道扇区数，PH表示每柱面总的磁道数。DIV是做整除运算，即被除数除以除数所得商的整数部分，MOD运算则是取余数。则有

$$C = \text{LBA} \div (\text{PH} \times \text{PS}) + \text{CS}$$

$$H = (\text{LBA} \div \text{PS}) \bmod \text{PH} + \text{HS}$$

$$S = \text{LBA} \bmod \text{PS} + \text{SS}$$

在CHS模式下，用寄存器al存扇区数，ch存柱面，cl存扇区，dh存磁头。由上面的代码可以知道LBA为1，因此扇区号为2，柱面、磁头均为0。驱动器为硬盘，因此dl为80h。

代码如下：

```
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 ; bootloder的加载地址
load_bootloder:
    call asm_read_hard_disk ; 读取硬盘
    inc ax
    cmp ax, 5
    jle load_bootloder
jmp 0x0000:0x7e00 ; 跳转到bootloder

jmp $ ; 死循环

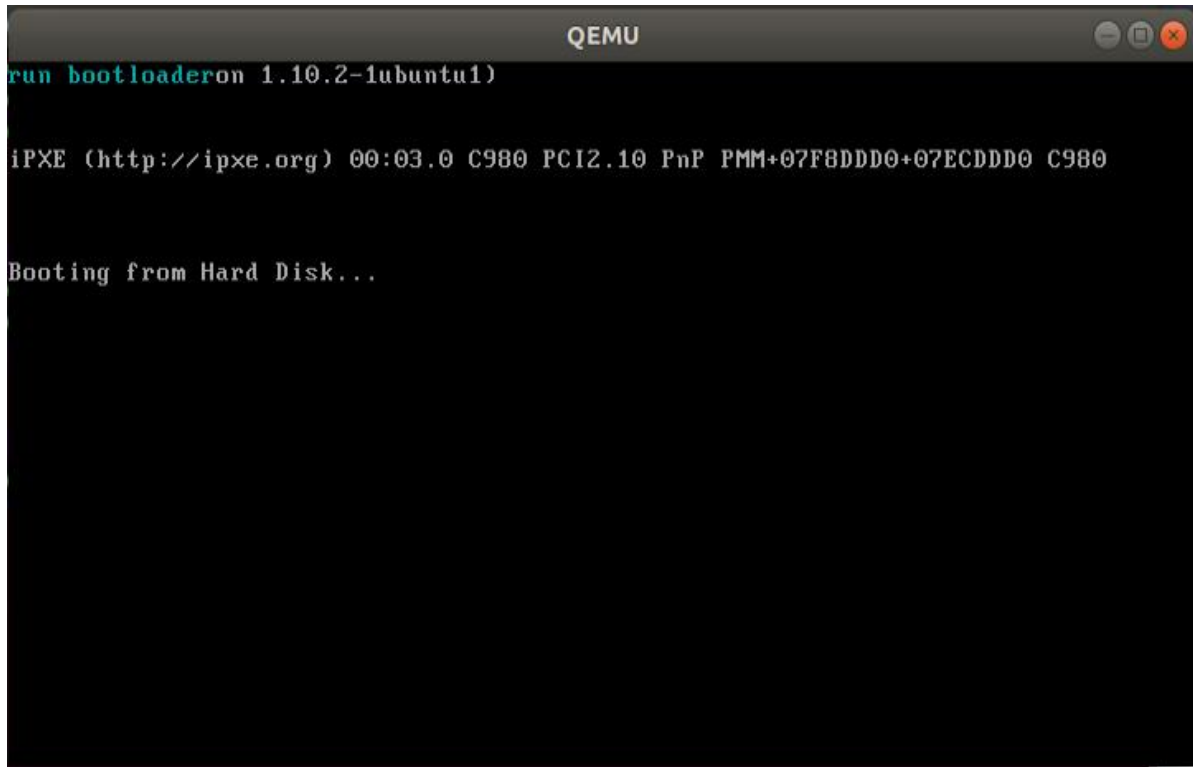
asm_read_hard_disk:
; 参数列表
; 入口参数: AH=02H
; AL=扇区数
; CH=柱面 CL=扇区
; DH=磁头
; DL=驱动器--软盘: 00H~7FH; 硬盘: 80H~0FFH
; ES: BX=缓冲区的地址
; 出口参数:
; CF=0: 操作成功 , AH=00H, AL=传输的扇区数
; CF=1: 操作失败, AH=状态码, 参见功能号01H的说明

push ax
mov cx, 0x0002
mov dx, 0x0080
mov ax, 0x0201
int 13h
; jc fail
pop ax
ret

jmp $ ; 死循环
times 510- ($ - $$) db 0
```

```
db 0x55, 0xaa
```

运行结果:

A screenshot of a QEMU terminal window. The title bar says "QEMU". The terminal output shows the command "run bootloaderon 1.10.2-1ubuntu1)" in green. Below it, the iPXE boot menu is displayed: "iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD0+07ECDDDD0 C980". Then, it says "Booting from Hard Disk...".

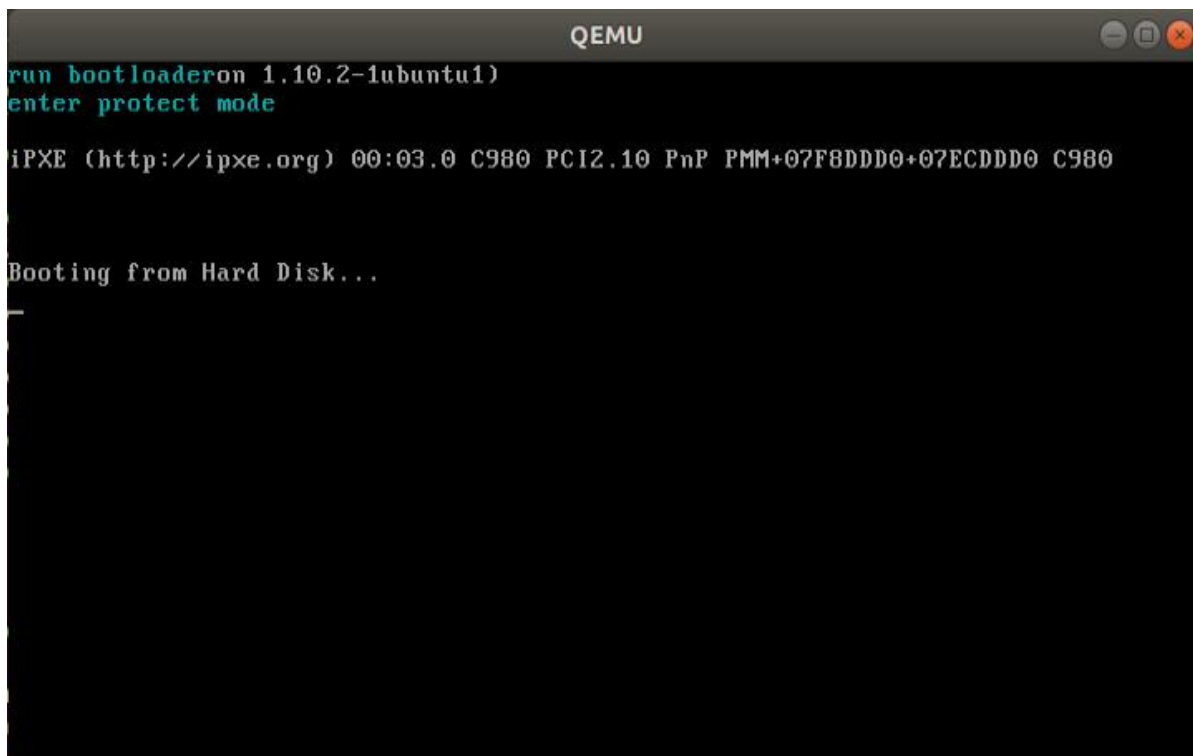
```
QEMU
run bootloaderon 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD0+07ECDDDD0 C980

Booting from Hard Disk...
```

Assignment2

复现代码的运行结果:

A screenshot of a QEMU terminal window, similar to the one above. The title bar says "QEMU". The terminal output shows the command "run bootloaderon 1.10.2-1ubuntu1)" in green, followed by "enter protect mode" in green. Then, the iPXE boot menu is displayed: "iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD0+07ECDDDD0 C980". Then, it says "Booting from Hard Disk...".

```
QEMU
run bootloaderon 1.10.2-1ubuntu1)
enter protect mode

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD0+07ECDDDD0 C980

Booting from Hard Disk...
```

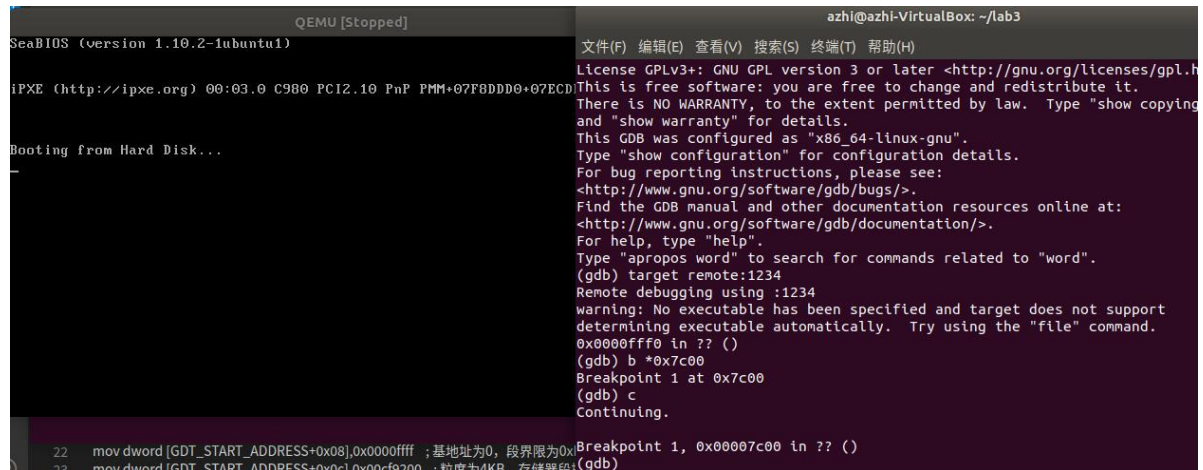
调试基本步骤:

- 1、qemu启动;
- 2、gdb启动;

- 3、加载符号表；
- 4、根据符号表信息设置断点；
- 5、运行至断点处；
- 6、查看寄存器或特定地址的值。

具体步骤：

gdb连上qemu后，设置第一个断点 `b *0x7c00`，然后按 `c` 运行。结果如下：

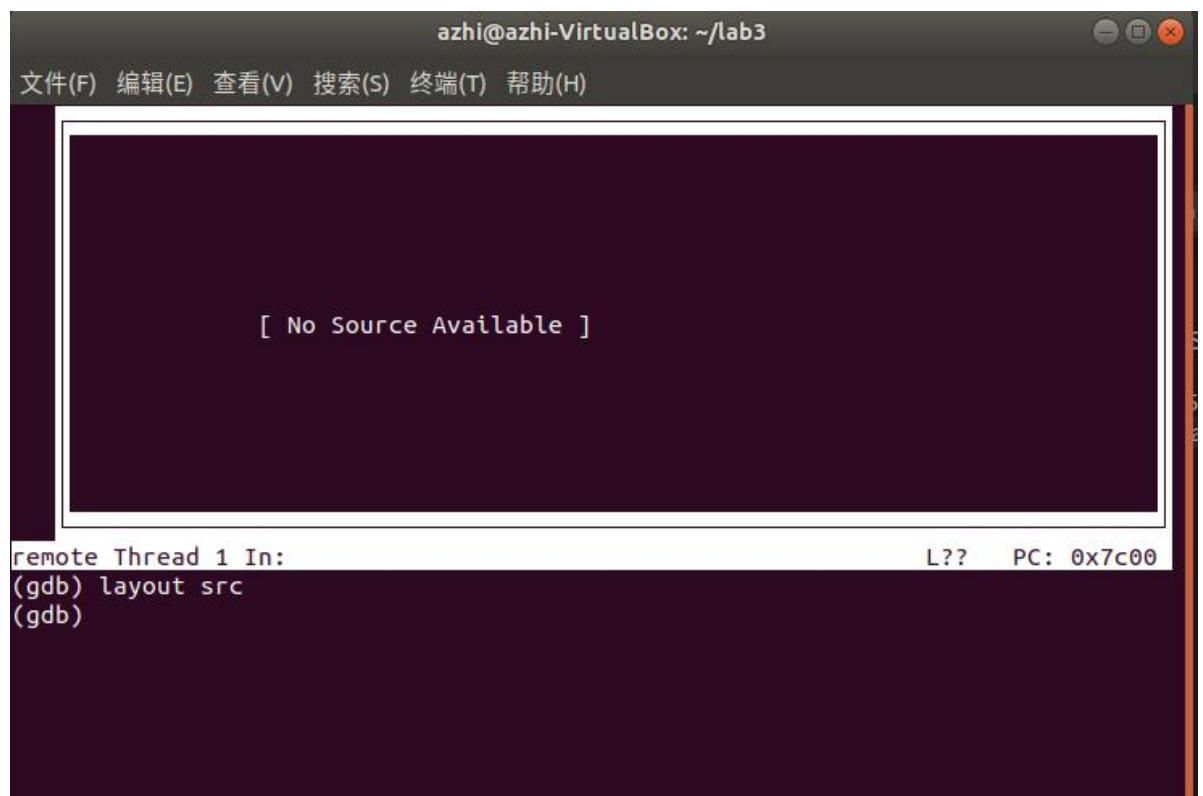


```
QEMU [Stopped]
SeaBIOS (version 1.10.2-1ubuntu1)
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DD0+07ECD
Booting from Hard Disk...

azhi@azhi-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.h
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote:1234
Remote debugging using :1234
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x0000ffff in ?? ()
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.

Breakpoint 1, 0x0007c00 in ?? ()
(gdb)
```

然后试着打开可以显示源代码的窗口。在 `gdb` 中输入命令 `layout src`，如下图



```
azhi@azhi-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

[ No Source Available ]

remote Thread 1 In: L?? PC: 0x7c00
(gdb) layout src
(gdb)
```

结果显示并没有加载符号表。那是因为还没有任何debug的信息显示源代码。

然后，加载MBR对应的符号表。输入命令

```
add-symbol-file mbr.symbol 0x7c00
```



```
azhi@azhi-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

mbr.asm
8      mov fs, ax
9      mov gs, ax
10
11     ; 初始化栈指针
12     mov sp, 0x7c00
13     mov ax, 1                ; 逻辑扇区号第0~15位
14     mov cx, 0                ; 逻辑扇区号第16~31位
15     mov bx, 0x7e00           ; bootloader的加载地址
16     load_bootloader:
17         call asm_read_hard_disk ; 读取硬盘
18         inc ax
19         cmp ax, 5
20         jle load_bootloader

remote Thread 1 In: L3 PC: 0x7c00
(gdb) layout src
(gdb) add-symbol-file mbr.symbol 0x7c00
add symbol table from file "mbr.symbol" at
        .text_addr = 0x7c00
(y or n) y
Reading symbols from mbr.symbol...done.
(gdb) 4 in /home/azhi/lab3/mbr.asm
(gdb) █
```

再次输入 layout src 时有此显示信息。

```
azhi@azhi-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

mbr.asm
10
11     ; 初始化栈指针
12     mov sp, 0x7c00
13     mov ax, 1                ; 逻辑扇区号第0~15位
14     mov cx, 0                ; 逻辑扇区号第16~31位
15     mov bx, 0x7e00           ; bootloader的加载地址
16     load_bootloader:
17         call asm_read_hard_disk ; 读取硬盘
18         inc ax
19         cmp ax, 5
20         jle load_bootloader
21     jmp 0x0000:0x7e00         ; 跳转到bootloader
22

remote Thread 1 In: L3 PC: 0x7c00
ds      0x0      0
es      0x0      0
---Type <return> to continue, or q <return> to quit---
fs      0x0      0
gs      0x0      0
(gdb) fs cmd
Focus set to cmd window.
(gdb) █
```



```
azhi@azhi-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

mbr.asm
10
11 ; 初始化栈指针
12 mov sp, 0x7c00
13 mov ax, 1 ; 逻辑扇区号第0~15位
14 mov cx, 0 ; 逻辑扇区号第16~31位
15 mov bx, 0x7e00 ; bootloader的加载地址
16 load_bootloader:
17 call asm_read_hard_disk ; 读取硬盘
18 inc ax
19 cmp ax, 5
20 jle load_bootloader
21 jmp 0x0000:0x7e00 ; 跳转到bootloader
22

remote Thread 1 In: L3 PC: 0x7c00
gs 0x0 0
(gdb) fs cmd
Focus set to cmd window.
(gdb) b *0x7e00
Breakpoint 2 at 0x7e00
(gdb) c
Continuing.
```

然后再次设断点在0x7e00处。结果continue时运行了很久，需要按ctr C让程序继续运行。

```
azhi@azhi-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

loader.asm
> 77 jmp $ ; 死循环
78
79 pgdt dw 0
80 dd GDT_START_ADDRESS
81
82 bootloader_tag db 'run bootloader'
83 bootloader_tag_end:
84
85 protect_mode_tag db 'enter protect mode'
86 protect_mode_tag_end:
87
88
89

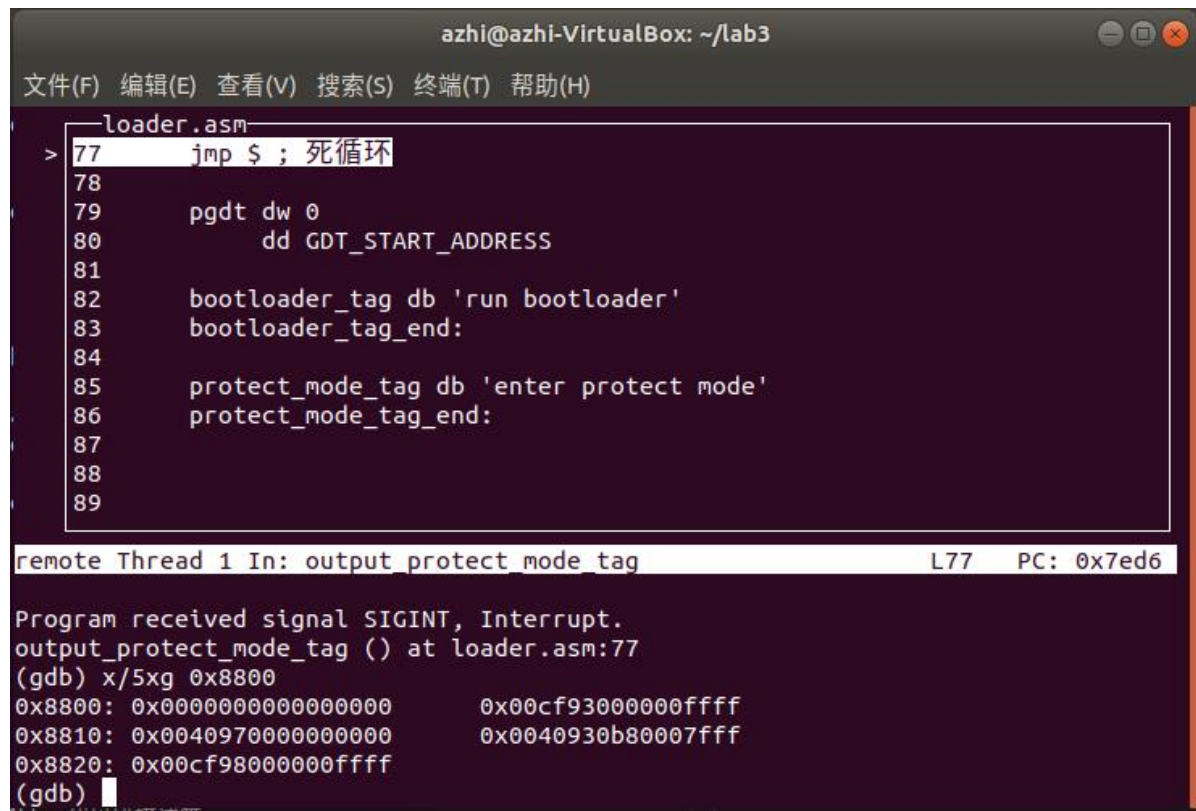
remote Thread 1 In: output protect mode tag L77 PC: 0x7ed6
Program received signal SIGINT, Interrupt.
0x00007ed6 in ?? ()
(gdb) add-symbol-file loader.symbol 0x7e00
add symbol table from file "loader.symbol" at
.text_addr = 0x7e00
(y or n) y
Reading symbols from loader.symbol...done.
(gdb)
```

这时需要像前面一样加载loader的符号表。

```
remote Thread 1 In: output_protect_mode_tag L77
.text_addr = 0x7e00
(y or n) y
Reading symbols from loader.symbol...done.
(gdb) b protect_mode_begin
Breakpoint 3 at 0x7ea2: file loader.asm, line 58.
(gdb) c
Continuing.
```

ctr c继续进行。

然后用x/5xg 0x8800查看GDT的5个段描述符的内容。



```
azhi@azhi-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

loader.asm
> 77 jmp $ ; 死循环
78
79 pgdt dw 0
80 dd GDT_START_ADDRESS
81
82 bootloader_tag db 'run bootloader'
83 bootloader_tag_end:
84
85 protect_mode_tag db 'enter protect mode'
86 protect_mode_tag_end:
87
88
89

remote Thread 1 In: output_protect_mode_tag L77 PC: 0x7ed6

Program received signal SIGINT, Interrupt.
output_protect_mode_tag () at loader.asm:77
(gdb) x/5xg 0x8800
0x8800: 0x0000000000000000 0x00cf93000000ffff
0x8810: 0x0040970000000000 0x0040930b80007fff
0x8820: 0x00cf98000000ffff
(gdb)
```

结果如上图所示，这和代码的设置一致。至此调试完毕。

Assignment3

进入保护模式后，加载数据段选择子，然后用寄存器esi存字符地址，寄存器ebx记载输出位置的偏移量。ah设置输出的前景和背景颜色。其他逻辑和lab2逻辑一致。将下面的代码替换 bootloader.asm 保护模式下的32位代码。

```
;以下进入保护模式
jmp dword CODE_SELECTOR:assi4

;16位的描述符选择子：32位偏移
;清流水线并串行化处理
[bits 32]
assi4:

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 ah, 3fh ;青色
mov esi, BootMessage
mov ebx, 0

loop1:
    dec word[count]          ; 递减计数变量
    jnz loop1                ; >0: 跳转;
    mov word[count], delay
    dec word[dcount]         ; 递减计数变量
    jnz loop1
    mov word[count], delay
    mov word[dcount], ddelay
    ; 以上是用一个二重循环实现时延50000*580个单位时间

    jmp Entrance            ;进行一个周期的工作
    jmp $                   ;halt

```

```

Entrance:
    jmp BoundaryCheckx

```

```

Set:
    mov esi, BootMessage
    xor ecx, ecx
    mov cl, byte[x]
    imul ecx, 2
    add ebx, ecx
    sub ebx, 4
    xor ecx, ecx
    mov cl, byte[y]
    imul ecx, 80
    add ebx, ecx
    ;xor ecx, ecx
    ;mov ecx, [x]

```

```

DispStr:
    ; mov esi, BootMessage
    ; mov ax, BootMessage ;打印字符串
    mov al, [esi]
    mov word[gs:ebx], ax
    add ebx, 2
    inc esi
    mov ecx, esi
    sub ecx, BootMessage
    cmp ecx, 3
    jl DispStr
    ;jmp Updatexy

```

```

Updatexy:
    mov al, byte[x]
    add al, byte[vx]
    mov byte[x], al
    mov al, byte[y]

```

```
    add     al, byte[vy]
    mov     byte[y], al
```

changeCol:

```
    mov     dh, ah
    cmp     dh, 1eh
    jz      col1
    mov     dh, ah
    cmp     dh, 5ah
    jz      col2
    mov     dh, ah
    cmp     dh, 6bh
    jz      col3
    mov     dh, ah
    cmp     dh, 25h
    jz      col4
    mov     dh, ah
    cmp     dh, 42h
    jz      col5
    mov     dh, ah
    cmp     dh, 3fh
    jz      col6
    mov     dh, ah
    cmp     dh, 7ch
    jz      col7
```

```
    ; jmp     loop1          ; 无限循环
```

BoundaryCheckx:

```
    mov     al, byte[x]
    add     al, byte[vx]    ; 预测下一刻的x
    cmp     al, byte[upper] ; 如果x小于上边界
    jl      Changevx       ; 更新vx
    cmp     al, byte[lower] ; 如果x大于下边界
    jg      Changevx       ; 更新vx
```

BoundaryChecky:

```
    mov     al, byte[y]
    add     al, byte[vy]
    cmp     al, byte[left]  ; 如果y小于左边界
    jl      Changevy       ; 更新vy
    add     al, byte[strlen] ; 预测下一刻的yr=y+字符串长
    cmp     al, byte[right] ; 如果yr大于下边界
    jg      Changevy       ; 更新vy
    jmp     Set             ; 如果不需要更新vx vy就继续打印流程
```

Changevx:

```
    neg     byte[vx]
    jmp     BoundaryChecky
```

Changevy:

```
    neg     byte[vy]
    jmp     Set
```

col1:

```
    mov     ah, 3fh
```

```

        jmp            loop1

col2:
    mov ah,7ch
    jmp            loop1

col3:
    mov ah,1eh
    jmp            loop1

col4:
    mov ah,5ah
    jmp            loop1

col5:
    mov ah,6bh
    jmp            loop1

col6:
    mov ah,25h
    jmp            loop1

col7:
    mov     ah,42h
    jmp            loop1

```

```

BootMessage      db      "zhi"
Strlen           db      3
delay            equ     50
ddelay           equ     500
count            dw      delay
dcount           dw      ddelay

x                db      0
y                db      0
vx              db      1
vy              db      1
left            db      0
upper           db      0
right           db      79
lower           db      24

boundary         db      10

pgdt dw 0
    dd GDT_START_ADDRESS

times 512*5- ($ - $$) db 0
db 0x55, 0xaa

```

运行, 查看结果:

