

# lab3

## 1、实验要求

学习如何从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。

## 2、实验过程+实验结果+关键代码

### Assignment1

#### 1.1 复现Example1: 使用了LBA28的方式来读取硬盘,加载bootloader

在实模式下，`0x0000:0x7e00` 表示段地址为0x0000, 偏移地址为0x7e00。等价于以下过程:

```
1 cs := 0x0000
2 ip := 0x7e00
```

编译 `bootloader.asm`, 写入硬盘起始编号为1的扇区，共有5个扇区。

```
1 nasm -f bin bootloader.asm -o bootloader.bin
2 dd if=bootloader.bin of=hd.img bs=512 count=5 seek=1 conv=notrunc
```

重新编译 `mbr.asm` 并写入硬盘起始编号为0的扇区。

```
1 nasm -f bin mbr.asm -o mbr.bin
2 dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

`bootloader.asm`

```
1 org 0x7e00
2 [bits 16]
3 mov ax, 0xb800
4 mov gs, ax
5 mov ah, 0x01 ;blue
6 mov ecx, bootloader_tag_end - bootloader_tag
7 xor ebx, ebx
8 mov esi, bootloader_tag
9 output_bootloader_tag:
10    mov al, [esi]
11    mov word[gs:bx], ax
12    inc esi
13    add ebx, 2
14    loop output_bootloader_tag
15 jmp $ ; 死循环
16
17 bootloader_tag db 'run bootloader assignment 1.1'
18 bootloader_tag_end:
```

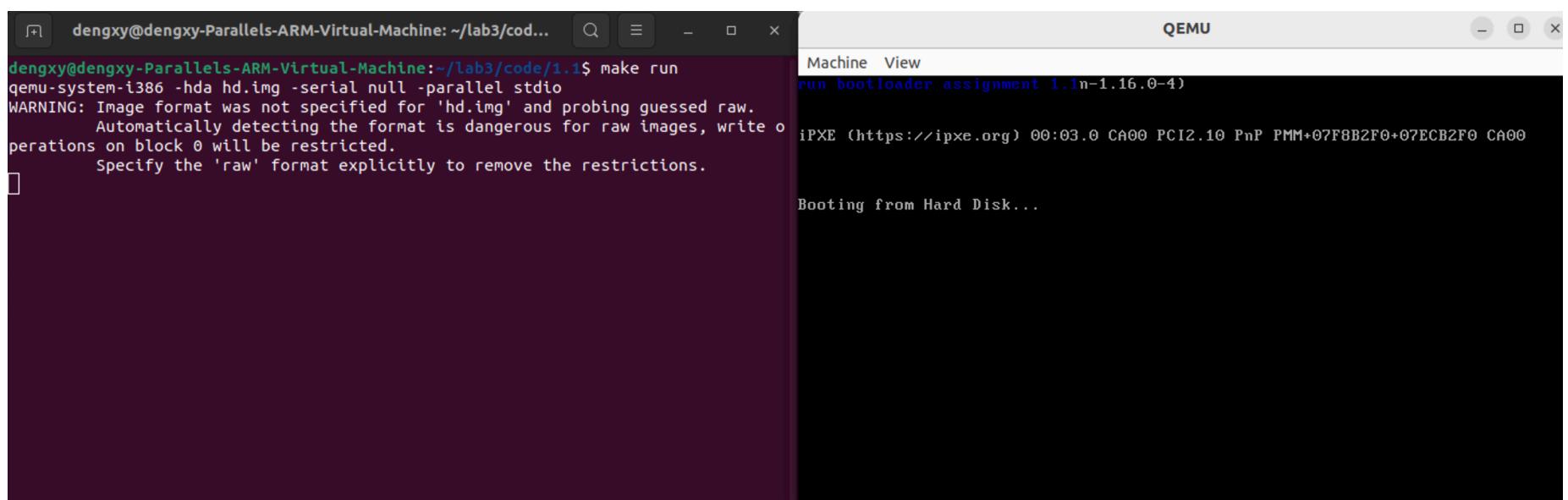
`mbr.asm`

```
1 org 0x7c00
2 [bits 16]
3 xor ax, ax ; eax = 0
4 ; 初始化段寄存器，段地址全部设为0
5 mov ds, ax
6 mov ss, ax
7 mov es, ax
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
21 jmp 0x0000:0x7e00    ; 跳转到bootloader
22
23 jmp $ ; 死循环
24
25 asm_read_hard_disk:
26 ; 从硬盘读取一个逻辑扇区
27
28 ; 参数列表
29 ; ax=逻辑扇区号0~15位
30 ; cx=逻辑扇区号16~28位
31 ; ds:bx=读取出的数据放入地址
32
33 ; 返回值
34 ; bx=bx+512
35
36     mov dx, 0x1f3
37     out dx, al      ; LBA地址7~0
38
39     inc dx          ; 0x1f4
40     mov al, ah
41     out dx, al      ; LBA地址15~8
42
43     mov ax, cx
44
45     inc dx          ; 0x1f5
46     out dx, al      ; LBA地址23~16
47
48     inc dx          ; 0x1f6
49     mov al, ah
50     and al, 0x0f
51     or al, 0xe0      ; LBA地址27~24
52     out dx, al
53
54     mov dx, 0x1f2
55     mov al, 1
```

```

56     out dx, al      ; 读取1个扇区
57
58     mov dx, 0x1f7    ; 0x1f7
59     mov al, 0x20     ; 读命令
60     out dx,al
61
62     ; 等待处理其他操作
63 .waits:
64     in al, dx        ; dx = 0x1f7
65     and al,0x88
66     cmp al,0x08
67     jnz .waits
68
69
70     ; 读取512字节到地址ds:bx
71     mov cx, 256      ; 每次读取一个字, 2个字节, 因此读取256次即可
72     mov dx, 0x1f0
73 .readw:
74     in ax, dx
75     mov [bx], ax
76     add bx, 2
77     loop .readw
78     ret
79
80 times 510 - ($ - $$) db 0
81 db 0x55, 0xaa

```



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。

### 转换公式

Reference: [LBA向CHS模式的转换](#)

CHS与LBA的转换规则：

硬盘系统在写入数据时，是按照从柱面到柱面的方式进行的，在上一个柱面写满数据后才移动磁头到下一个柱面，并从柱面的第一个磁头的第一个扇区开始写入，从而使硬盘性能最优。在对物理扇区进行线性编址时，也会按照这种方式进行。

CHS地址			LBA编号
柱面	磁头	扇区	
0	0	1	0
0	0	2	1
0	0	3~63	2~62
0	1	1	63
0	1	2~63	64~125
0	2	1~63	126~188
0	3	1~63	189~251
1	0	1	252
1	0	2~63	253~314
1	1	1	315

CHS到LBA的转换公式

$$LBA = (C - Cs) \times PH \times PS + (H - Hs) \times Ps + (S - Ss)$$

$C, H, S$ 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 (1)  
 $PS$ 表示每磁道扇区数， $PH$ 表示每柱面总的磁道数  
通常情况： $Cs = 0, Hs = 0, Ss = 1, PS = 63, PH = 255$

LBA到CHS的转换公式：

$$\begin{aligned} C &= LBA \text{ DIV } (PH \times PS) + Cs \\ H &= (LBA \text{ DIV } PS) \text{ MOD } PH + Hs \\ S &= LBA \text{ MOD } PS + Ss \end{aligned}$$

即 (2)

$$\begin{aligned} \text{绝对扇区} &= (\text{逻辑扇区} \text{ mod } \text{每个柱面的总扇区数}) + 1 \\ \text{绝对磁头} &= (\text{逻辑扇区} / \text{每个柱面的总扇区数}) \text{ mod } \text{总磁头数} \\ \text{绝对柱面} &= \text{逻辑扇区} / (\text{每个柱面的总扇区数} * \text{总磁头数}) \end{aligned}$$

或

$$\begin{aligned} C &= LBA \text{ DIV } (PH \times PS) + Cs \\ H &= LBA \text{ DIV } PS - (C - Cs) \times PH + Hs \\ S &= LBA - (C - Cs) \times PH \times PS - (H - Hs) \times Ps + Ss \end{aligned}$$

### INT 13H中断

INT 13H是BIOS中断中的一个子功能，提供了访问和控制磁盘驱动器的功能。通过该中断，可以实现磁盘的读取、写入、格式化、校验等操作。它接受一个参数块作为输入，其中包含了读写操作的具体信息，如扇区数、柱面号、磁头号、扇区大小等。

Reference: [INT 13H](#)

```

1 入口参数:
2 AH=功能
3 AL=扇区数
4 CH=柱面
5 CL=扇区
6 DH=磁头
7 DL=驱动器, 00H~7FH: 软盘; 80H~0FFH: 硬盘
8 ES:BX=缓冲区的地址
9 出口参数:
10 CF=0 操作成功, AH=00H, AL=传输的扇区数, 否则, AH=状态代码, 参见功能号01H中的说明

```

对于本题的一些相关参数

```

1 AH=02H    # 功能: 读磁盘
2 AL=05H    # 5个扇区空间
3 CH=00H
4 DH=00H
5 CL=2      # 读取第二个扇区 (mbr.asm在硬盘起始编号为0的扇区, bootloader.asm在硬盘起始编号为1的扇区)
6 DL=80H    # 硬盘

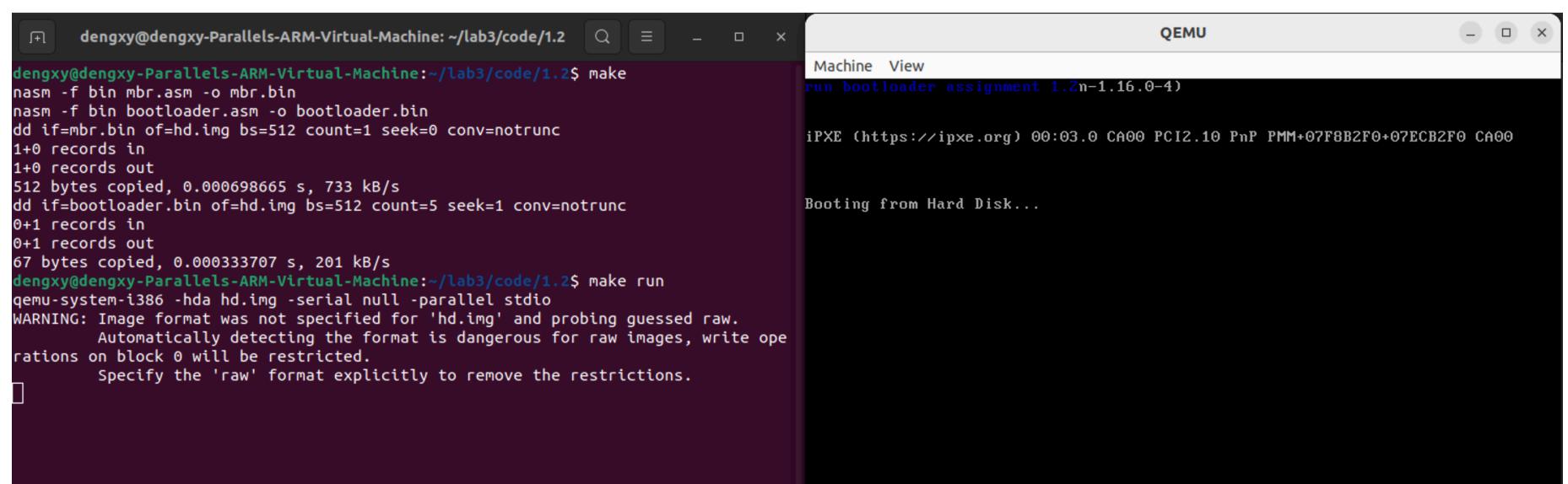
```

在 mbr.asm 中编写 `read_hard_disk` 函数

```

1 read_hard_disk:
2     mov ah, 02h ; 功能: 读磁盘
3     mov al, 05h ; 扇区数
4     mov ch, 00h ; 柱面
5     mov dh, 00h ; 磁头
6     mov cl, 2   ; 扇区
7     mov dl, 80h ; 硬盘
8     int 0x13    ; 中断
9 ret; 子程序返回

```



**Assignment2: 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。**

在bootloader中进入保护模式, 并在进入保护模式后在显示屏上输出“protect mode”

部分常量的定义 `boot.inc`

```

1 ; 常量定义区
2 ; _____ Loader _____
3 ; 加载器扇区数
4 LOADER_SECTOR_COUNT equ 5
5 ; 加载器起始扇区
6 LOADER_START_SECTOR equ 1
7 ; 加载器被加载地址
8 LOADER_START_ADDRESS equ 0x7e00
9 ; _____ GDT _____
10 ; GDT起始位置
11 GDT_START_ADDRESS equ 0x8800

```

`equ`为汇编伪指令，类似于C中的`define`

进入保护模式：

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

Step1:

定义段描述符：代码段描述符、数据段描述符、栈段描述符、视频段描述符（显存所在的内存区域的段描述符）

使用平坦模式：简化地址的访问，让所有的程序运行在同一个段中，这个段的起始地址为0，地址空间大小是4GB，这是全部的地址空间。同时，让代码段描述符、数据段描述符和栈段描述符中的段线性基地址为0，那么偏移地址和线性地址就完全相同，

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（GDT的第一个描述符必须是全0的描述符）

```

1 %include "boot.inc"
2 org 0x7e00
3 [bits 16]
4 ... ; 输出bootloader_tag代码，此处省略
5 ;空描述符
6 mov dword [GDT_START_ADDRESS+0x00],0x00
7 mov dword [GDT_START_ADDRESS+0x04],0x00
8
9 ;创建描述符，这是一个数据段，对应0~4GB的线性地址空间
10 mov dword [GDT_START_ADDRESS+0x08],0x0000ffff ; 基地址为0，段界限为0xFFFFFFF
11 mov dword [GDT_START_ADDRESS+0x0c],0x00cf9200 ; 粒度为4KB，存储器段描述符
12
13 ;建立保护模式下的堆栈段描述符
14 mov dword [GDT_START_ADDRESS+0x10],0x00000000 ; 基地址为0x00000000，界限0x0
15 mov dword [GDT_START_ADDRESS+0x14],0x00409600 ; 粒度为1个字节
16
17 ;建立保护模式下的显存描述符
18 mov dword [GDT_START_ADDRESS+0x18],0x80007fff ; 基地址为0x000B8000，界限0x07FFF
19 mov dword [GDT_START_ADDRESS+0x1c],0x0040920b ; 粒度为字节
20

```

```
21 ;创建保护模式下平坦模式代码段描述符  
22 mov dword [GDT_START_ADDRESS+0x20],0x0000ffff ; 基地址为0, 段界限为0xFFFFF  
23 mov dword [GDT_START_ADDRESS+0x24],0x00cf9800 ; 粒度为4kb, 代码段描述符
```

设置GDTR寄存器：高32位表示GDT的起始地址，低16位表示GDT的界限。（界限=GDT的长度-1）

把GDT信息写入 pgdt，将 pgdt 加载进GDTR

```
1 ;初始化描述符表寄存器GDTR  
2 mov word [pgdt], 39 ;描述符表的界限  
3 lgdt [pgdt]
```

设置段选择子

```
1 ; _____ Selector _____  
2 ;平坦模式数据段选择子  
3 DATA_SELECTOR equ 0x8  
4 ;平坦模式栈段选择子  
5 STACK_SELECTOR equ 0x10  
6 ;平坦模式视频段选择子  
7 VIDEO_SELECTOR equ 0x18  
8 VIDEO_NUM equ 0x18  
9 ;平坦模式代码段选择子  
10 CODE_SELECTOR equ 0x20
```

Step2:

打开第21根地址线

```
1 in al,0x92 ;南桥芯片内的端口  
2 or al,0000_0010B  
3 out 0x92,al ;打开A20
```

Step3:

开启cr0的保护模式标志位（PE位）

```
1 cli ;中断机制尚未工作  
2 mov eax,cr0  
3 or eax,1  
4 mov cr0,eax ;设置PE位
```

Step4:

远跳转进入保护模式

```
1 jmp dword CODE_SELECTOR:protect_mode_begin
```

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

对 mbr.asm 中 asm\_read\_hard\_disk 的修改

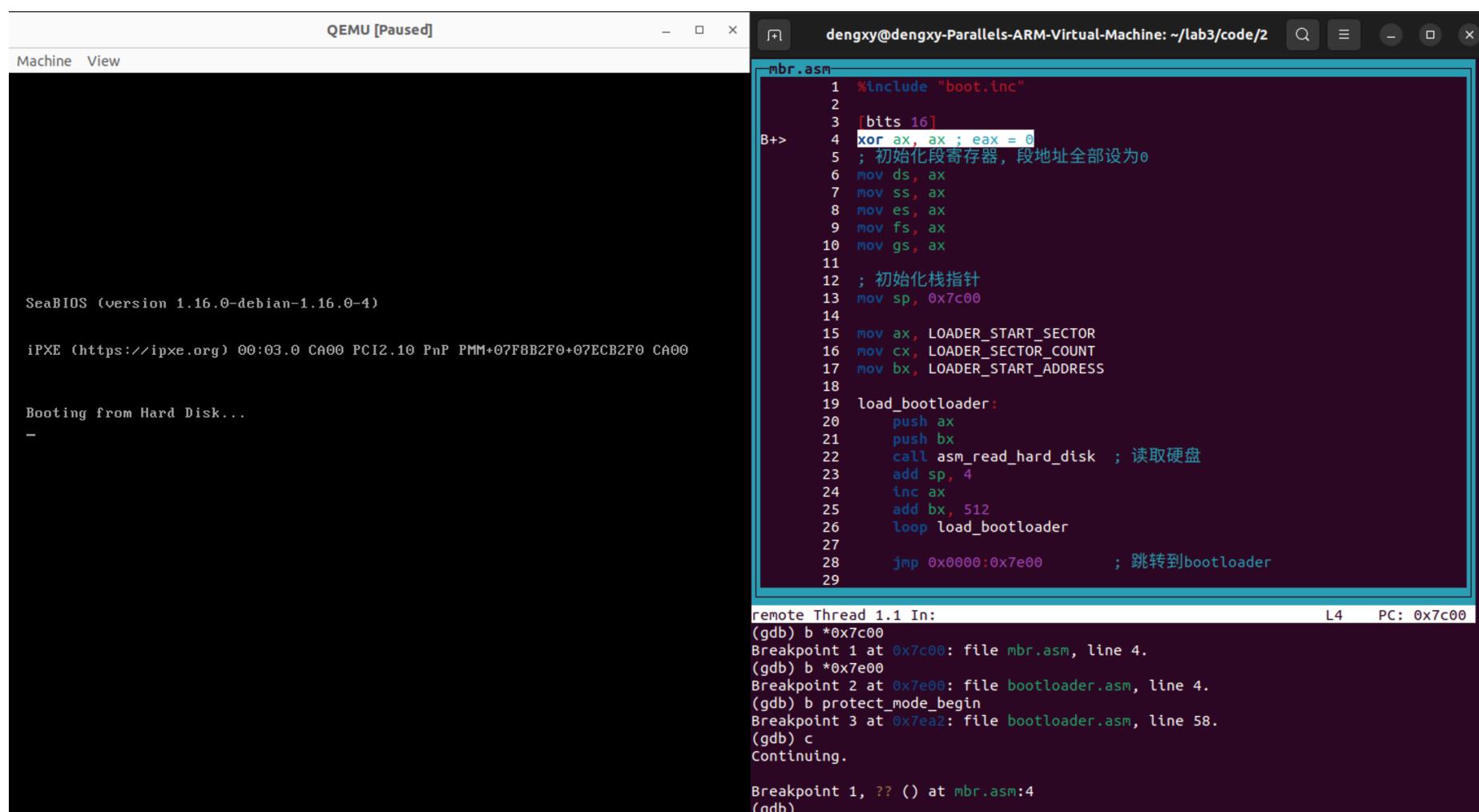
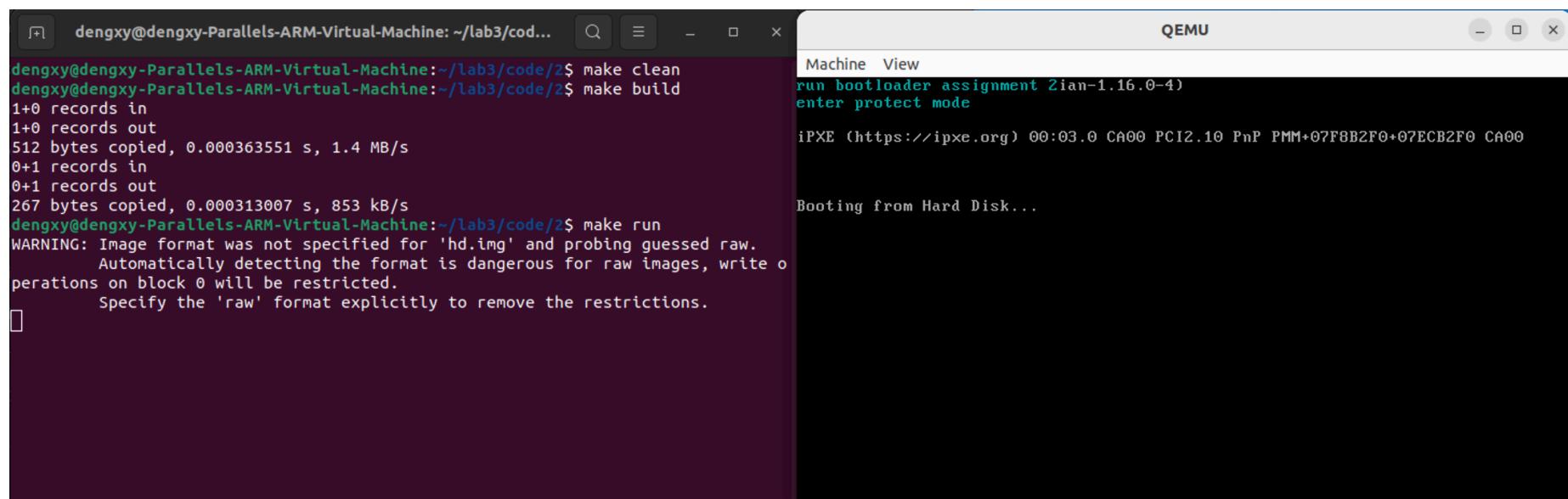
```
1 ; asm_read_hard_disk(memory,block)
```

```
2 ; void asm_read_hard_disk(int memory, int block)
3 ; 加载逻辑扇区号为block的扇区到内存地址memory
4 asm_read_hard_disk:
5     push bp
6     mov bp, sp
7     push ax
8     push bx
9     push cx
10    push dx
11
12    mov ax, [bp + 2 * 3] ; 逻辑扇区低16位
13
14    mov dx, 0x1f3
15    out dx, al ; LBA地址7~0
16
17    inc dx ; 0x1f4
18    mov al, ah
19    out dx, al ; LBA地址15~8
20
21    xor ax, ax
22    inc dx ; 0x1f5
23    out dx, al ; LBA地址23~16 = 0
24
25    inc dx ; 0x1f6
26    mov al, ah
27    and al, 0x0f
28    or al, 0xe0 ; LBA地址27~24 = 0
29    out dx, al
30
31    mov dx, 0x1f2
32    mov al, 1
33    out dx, al ; 读取1个扇区
34
35    mov dx, 0x1f7 ; 0x1f7
36    mov al, 0x20 ; 读命令
37    out dx, al
38
39    ; 等待处理其他操作
40    .waits:
41    in al, dx ; dx = 0x1f7
42    and al, 0x88
43    cmp al, 0x08
44    jnz .waits
45
46    ; 读取512字节到地址ds:bx
47    mov bx, [bp + 2 * 2]
48    mov cx, 256 ; 每次读取一个字, 2个字节, 因此读取256次即可
49    mov dx, 0x1f0
50    .readw:
51    in ax, dx
52    mov [bx], ax
53    add bx, 2
54    loop .readw
55    pop dx
56    pop cx
```

```
57     pop bx
58     pop ax
59     pop bp
60 ret
```

调用代码 `load_bootloader`

```
1 load_bootloader:
2     push ax;逻辑扇区号
3     push bx;加载地址
4     call asm_read_hard_disk ; 读取硬盘
5     add sp, 4;连续pop掉两个参数
6     inc ax
7     add bx, 512
8     loop load_bootloader
9     jmp 0x0000:0x7e00           ; 跳转到bootloader
10    jmp $ ; 死循环
```



QEMU [Paused]

Machine View

```
SeaBIOS (version 1.16.0-debian-1.16.0-4)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B2F0+07ECB2F0 CA00

Booting from Hard Disk...
-
```

dengxy@dengxy-Parallels-ARM-Virtual-Machine: ~/lab3/code/2

```
bootloader.asm
1 %include "boot.inc"
2 ;org 0x7e00
3 [bits 16]
B+> 4 mov ax, 0xb800
5 mov gs, ax
6 mov ah, 0x03 ;青色
7 mov ecx, bootloader_tag_end - bootloader_tag
8 xor ebx, ebx
9 mov esi, bootloader_tag
10 output_bootloader_tag:
11     mov al, [esi]
12     mov word[gs:bx], ax
13     inc esi
14     add ebx, 2
15     loop output_bootloader_tag
16
17 ;空描述符
18 mov dword [GDT_START_ADDRESS+0x00], 0x00
19 mov dword [GDT_START_ADDRESS+0x04], 0x00
20
21 ;创建描述符，这是一个数据段，对应0~4GB的线性地址空间
22 mov dword [GDT_START_ADDRESS+0x08], 0x0000ffff ; 基址为0，段界限为
23 mov dword [GDT_START_ADDRESS+0x0c], 0x00cf9200 ; 粒度为4KB，存储器段
24
25 ;建立保护模式下的堆栈段描述符
26 mov dword [GDT_START_ADDRESS+0x10], 0x00000000 ; 基址为0x00000000
27 mov dword [GDT_START_ADDRESS+0x14], 0x00409600 ; 粒度为1个字节
28
29 ;建立保护模式下的显存描述符

remote Thread 1.1 In: L4 PC: 0x7e00
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00: file mbr.asm, line 4.
(gdb) b *0x7e00
Breakpoint 2 at 0x7e00: file bootloader.asm, line 4.
(gdb) b protect_mode_begin
Breakpoint 3 at 0x7ea2: file bootloader.asm, line 58.
(gdb) c
Continuing.

Breakpoint 1, ?? () at mbr.asm:4
(gdb) c
Continuing.

Breakpoint 2, ?? () at bootloader.asm:4
(gdb)
```

QEMU [Paused]

Machine View

```
run bootloader assignment Zian-1.16.0-4

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B2F0+07ECB2F0 CA00

Booting from Hard Disk...
-
```

dengxy@dengxy-Parallels-ARM-Virtual-Machine: ~/lab3/code/2

```
bootloader.asm
45 cli ;中断机制尚未工作
46 mov eax,cr0
47 or eax,1
48 mov cr0,eax ;设置PE位
49
50 ;以下进入保护模式
51 jmp dword CODE_SELECTOR.protect_mode_begin
52
53 ;16位的描述符选择子：32位偏移
54 ;清流水线并串行化处理器
55 [bits 32]
56 protect_mode_begin:
57
58     mov eax, DATA_SELECTOR ;加载数据段(0..4GB)选择子
59     mov ds, eax
60     mov es, eax
61     mov eax, STACK_SELECTOR
62     mov ss, eax
63     mov eax, VIDEO_SELECTOR
64     mov gs, eax
65
66     mov ecx, protect_mode_tag_end - protect_mode_tag
67     mov ebx, 80 * 2
68     mov esi, protect_mode_tag
69     mov ah, 0x3
70     output_protect_mode_tag:
71         mov al, [esi]
72         mov word[gs:ebx], ax
73         add ebx, 2

remote Thread 1.1 In: protect_mode_begin L58 PC: 0x7ea2
Breakpoint 2 at 0x7e00: file bootloader.asm, line 4.
(gdb) b protect_mode_begin
Breakpoint 3 at 0x7ea2: file bootloader.asm, line 58.
(gdb) c
Continuing.

Breakpoint 1, ?? () at mbr.asm:4
(gdb) c
Continuing.

Breakpoint 2, ?? () at bootloader.asm:4
(gdb) c
Continuing.

Breakpoint 3, protect_mode_begin () at bootloader.asm:58
(gdb)
```

QEMU

Machine View

```
run bootloader assignment Zian-1.16.0-4)
enter protect mode

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B2F0+07ECB2F0 CA00

Booting from Hard Disk...
```

dengxy@dengxy-Parallels-ARM-Virtual-Machine: ~/lab3/code/2

```
bootloader.asm
45 cli ; 中断机制尚未工作
46 mov eax,cr0
47 or eax,1
48 mov cr0,eax ; 设置PE位
49
50 ; 以下进入保护模式
51 jmp dword CODE_SELECTOR:protect_mode_begin
52
53 ; 16位的描述符选择子：32位偏移
54 ; 清流水线并串行化处理器
55 [bits 32]
56 protect_mode_begin:
57
58 mov eax, DATA_SELECTOR ; 加载数据段(0..4GB)选择子
59 mov ds, eax
60 mov es, eax
61 mov eax, STACK_SELECTOR
62 mov ss, eax
63 mov eax, VIDEO_SELECTOR
64 mov gs, eax
65
66 mov ecx, protect_mode_tag_end - protect_mode_tag
67 mov ebx, 80 * 2
68 mov esi, protect_mode_tag
69 mov ah, 0x3
70 output_protect_mode_tag:
71     mov al, [esi]
72     mov word[gs:ebx], ax
73     add ebx, 2

remote Thread 1.1 In: protect_mode_begin
Breakpoint 3 at 0x7ea2: file bootloader.asm, line 58.
(gdb) c
Continuing.

Breakpoint 1, ?? () at mbr.asm:4
(gdb) c
Continuing.

Breakpoint 2, ?? () at bootloader.asm:4
(gdb) c
Continuing.

Breakpoint 3, protect_mode_begin () at bootloader.asm:58
(gdb) c
Continuing.
```

L58 PC: 0x7ea2

info registers

```
(gdb) info registers
eax          0x11          17
ecx          0x0           0
edx          0x80          128
ebx          0x36          54
esp          0x7c00        0x7c00
ebp          0x0           0x0
esi          0x7ef9        32505
edi          0x0           0
eip          0x7ea2        0x7ea2 <protect_mode_begin>
eflags        0x6           [ IOPL=0 PF ]
cs           0x20          32
ss           0x0           0
ds           0x0           0
es           0x0           0
fs           0x0           0
gs           0xb800        47104
fs_base      0x0           0
gs_base      0xb8000       753664
k_gs_base    0x0           0
cr0          0x11          [ ET PE ]
cr2          0x0           0
cr3          0x0           [ PDPR=0 PCID=0 ]
cr4          0x0           [ ]
cr8          0x0           0
efer         0x0           [ ]
xmm0          {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm1          {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm2          {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm3          {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm4          {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm5          {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
```

## Assignment3: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。

结合lab2的assignment4，修改 `bootloader.asm`

在进入保护模式后，运行lab2的assignment4部分的代码

```
1 %include "boot.inc"
2 org 0x7e00;生成符号表时要注释掉
3 [bits 16]
4
5 mov dword [GDT_START_ADDRESS+0x00],0x00
6 mov dword [GDT_START_ADDRESS+0x04],0x00
7
8 mov dword [GDT_START_ADDRESS+0x08],0x0000ffff ; 基址为0, 段界限为0xFFFF
9 mov dword [GDT_START_ADDRESS+0x0c],0x00cf9200 ; 粒度为4KB, 存储器段描述符
10
11 ;建立保护模式下的堆栈段描述符
12 mov dword [GDT_START_ADDRESS+0x10],0x00000000 ; 基址为0x00000000, 界限0x0
13 mov dword [GDT_START_ADDRESS+0x14],0x00409600 ; 粒度为1个字节
14
15 ;建立保护模式下的显存描述符
16 mov dword [GDT_START_ADDRESS+0x18],0x80007fff ; 基址为0x000B8000, 界限0x07FFF
17 mov dword [GDT_START_ADDRESS+0x1c],0x0040920b ; 粒度为字节
18
19 ;创建保护模式下平坦模式代码段描述符
20 mov dword [GDT_START_ADDRESS+0x20],0x0000ffff ; 基址为0, 段界限为0xFFFF
21 mov dword [GDT_START_ADDRESS+0x24],0x00cf9800 ; 粒度为4kb, 代码段描述符
22
23 ;初始化描述符表寄存器GDTR
24 mov word [pgdt], 39 ;描述符表的界限
25 lgdt [pgdt]
26
27 in al,0x92 ;南桥芯片内的端口
28 or al,0000_0010B
29 out 0x92,al ;打开A20
30
31 cli ;中断机制尚未工作
32 mov eax,cr0
33 or eax,1
34 mov cr0,eax ;设置PE位
35
36 ;以下进入保护模式
37 jmp dword CODE_SELECTOR:protect_mode_begin
38
39 ;16位的描述符选择子: 32位偏移
40 ;清流水线并串行化处理器
41 [bits 32]
42 protect_mode_begin:
43
44     _DR equ 1
45     _UR equ 2
46     _UL equ 3
```

```
47     _DL equ 4
48     delay equ 300
49     double_delay equ 150
50
51
52 ;初始化
53 START:
54
55 mov eax, DATA_SELECTOR
56 mov ds, eax
57 mov gs, eax
58 mov eax, STACK_SELECTOR
59 mov ss, eax
60 mov eax, VIDEO_SELECTOR
61 mov es, eax
62
63 ; mov ecx,0
64 ; mov eax,0
65 ; _init_:
66 ;     cmp ecx,0x00007FFF
67 ;     je _loop
68 ;         mov dword[gs:ecx], eax
69 ;         add ecx,1
70 ;         jmp _init_
71 ; _loop:
72 ; mov ebx,2
73 ; mov ecx,0
74 ; mov esi,1          ;offset
75 ; mov edi,1          ;offset
76 ; ; dead loop
77 ; ;initializing the start_point
78 ;     mov ax, cs
79 ;     mov es, ax
80 ;     mov ds, ax
81 ;     mov ax, 0b800h
82 ;     mov es, ax
83     mov esi, 0
84     mov edi, 0
85
86 ;mov ecx, protect_mode_tag_end - protect_mode_tag
87 ;mov ebx, 80 * 2
88 ;mov esi, protect_mode_tag
89 ;mov ah, 0x3
90 ;output_protect_mode_tag:
91 ;    mov al, [esi]
92 ;    mov word[gs:ebx], ax
93 ;    add ebx, 2
94 ;    inc esi
95 ;    loop output_protect_mode_tag
96
97 ;jmp $ ; 死循环
98
99 ;lab2的assignment4中部分代码
100 ;寄存器要改成32位寄存器
101
```

```
102 PRINT:  
103     mov ebx, title  
104     mov al, [ebx+esi]  
105     cmp al, 0  
106     jz LOOP1  
107     mov ebx, 52  
108     mov byte[es:ebx+edi], al  
109     mov byte[es:ebx+edi+1], 1  
110     inc esi  
111     add edi, 2  
112     jmp PRINT  
113  
114 ;循环实现延迟  
115 LOOP1:  
116     dec dword[count]  
117     jnz LOOP1  
118  
119     mov dword[count], delay  
120     dec dword[double_count]  
121     jnz LOOP1  
122  
123     mov dword[count], delay  
124     mov dword[double_count], double_delay  
125  
126     mov al, 1  
127     cmp al, byte[RightDownUpLeft]  
128     jz DownRight  
129  
130     mov al, 2  
131     cmp al, byte[RightDownUpLeft]  
132     jz UpRight  
133  
134     mov al, 3  
135     cmp al, byte[RightDownUpLeft]  
136     jz UpLeft  
137  
138     mov al, 4  
139     cmp al, byte[RightDownUpLeft]  
140     jz DownLeft  
141  
142     jmp $  
143  
144 ;往右下移动  
145 DownRight:  
146     inc dword[x]  
147     inc dword[y]  
148     mov ebx, dword[x]  
149     mov eax, 25  
150     sub eax, ebx  
151     jz DownRightToUpRight  
152  
153     mov ebx, dword[y]  
154     mov eax, ebx  
155     jz DownRightToLeft  
156
```

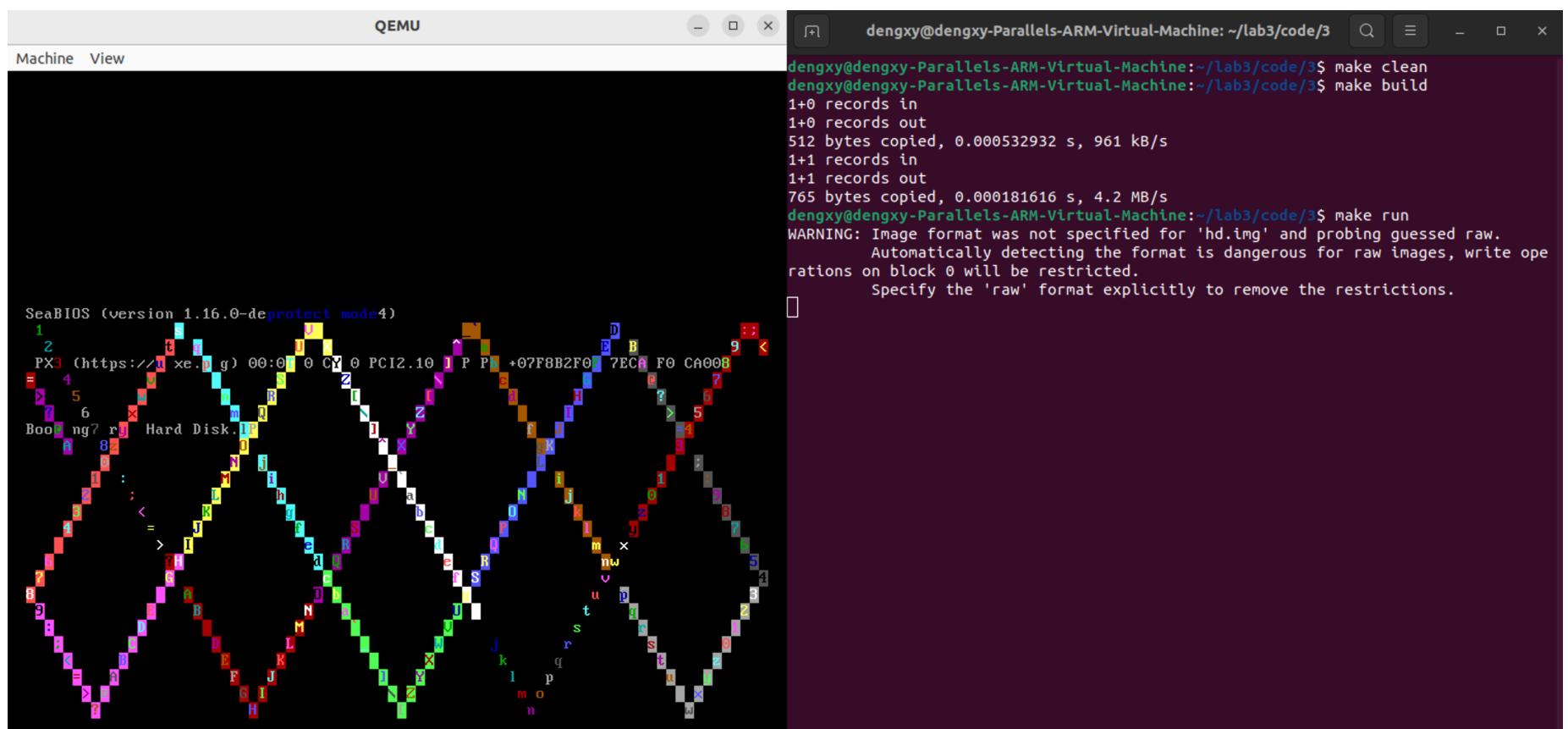
```
157     jmp show
158
159 DownRightToUpRight:
160     mov dword[x], 23
161     mov byte[RightDownUpLeft], _UR
162     jmp show
163 DownRightToLeft:
164     mov dword[y], 78
165     mov byte[RightDownUpLeft], _DL
166     jmp show
167
168 ;往右上移动
169 UpRight:
170     dec dword[x]
171     inc dword[y]
172     mov ebx, dword[y]
173     mov eax, 80
174     sub eax, ebx
175     jz UpRightToUpLeft
176
177     mov ebx, dword[x]
178     mov eax, 0
179     sub eax, ebx
180     jz UpRightToLeft
181
182     jmp show
183
184 UpRightToUpLeft:
185     mov dword[y], 78
186     mov byte[RightDownUpLeft], _UL
187     jmp show
188 UpRightToLeft:
189     mov dword[x], 1
190     mov byte[RightDownUpLeft], _DR
191     jmp show
192
193 ;往左上移动
194 UpLeft:
195     dec dword[x]
196     dec dword[y]
197     mov ebx, dword[x]
198     mov eax, 0
199     sub eax, ebx
200     jz UpLeftToUpRight
201
202     mov ebx, dword[y]
203     mov eax, -1
204     sub eax, ebx
205     jz UpLeftToLeft
206
207     jmp show
208
209 UpLeftToLeft:
210     mov dword[x], 1
211     mov byte[RightDownUpLeft], _DL
```

```
212         jmp show
213 UpLeftToUpRight:
214     mov dword[y], 1
215     mov byte[RightDownUpLeft], _UR
216     jmp show
217
218 ;往左下移动
219 DownLeft:
220     inc dword[x]
221     dec dword[y]
222     mov ebx, dword[y]
223     mov eax, -1
224     sub eax, ebx
225     jz DownLeftToDownRight
226
227     mov ebx, dword[x]
228     mov eax, 25
229     sub eax, ebx
230     jz DownLeftToUpLeft
231
232     jmp show
233
234 DownLeftToDownRight:
235     mov dword[y], 1
236     mov byte[RightDownUpLeft], _DR
237     jmp show
238 DownLeftToUpLeft:
239     mov dword[x], 23
240     mov byte[RightDownUpLeft], _UL
241     jmp show
242
243 ;在屏幕上显示字符
244 show:
245     xor eax, eax
246     mov eax, dword[x]
247     mov ebx, 80
248     mul ebx
249     add eax, dword[y]
250     mov ebx, 2
251     mul ebx
252     mov ebx, eax
253     mov ah, byte[color]
254     mov al, byte[char]
255     mov [es:ebx], eax
256
257     inc byte[char]
258     cmp byte[char], 'z'+1
259     jnz keep
260     mov byte[char], '0'
261
262 keep:
263     inc byte[color]
264     cmp byte[color], 0x10
265     jnz LOOP1
266     mov byte[color], 0x40 ;
```

```

267     jmp LOOP1
268
269 end:
270     jmp $
271
272     count dd delay
273     double_count dd double_delay
274     RightDownUpLeft db _DR
275     color db 0x02
276     x dd 0
277     y dd 0
278     char db '1'
279     title db 'protect mode', 0
280
281 ; times 510-($-$) db 0
282 ; dw 0aa55h
283
284 pgdt dw 0
285     dd GDT_START_ADDRESS
286
287 ;bootloader_tag db 'run bootloader'
288 ;bootloader_tag_end:
289
290 ;protect_mode_tag db 'enter protect mode'
291 ;protect_mode_tag_end:

```



### 3、总结

- 学习了如何从16位的实模式跳转到32位的保护模式
- 学习了LBA和CHS两种读取磁盘的方式
- 对汇编语言有了进一步的理解
- 更加熟练运用 `gdb` 等debug工具来对程序进行调试