

lab3 report

实验三报告

信息

- 姓名：李浩辉
- 学号：21307018
- 学院：计算机学院
- 时间：2023.4.1

目录

1. 实验要求
2. 实验过程（含关键代码）
3. 实验问题分析
4. 实验总结

实验要求

1. 复现example 1 以及用CHS模式代替LBA模式读取磁盘复现example 1
2. 复现example 2并用gdb调试
3. 保护模式下执行自定义汇编程序

实验过程

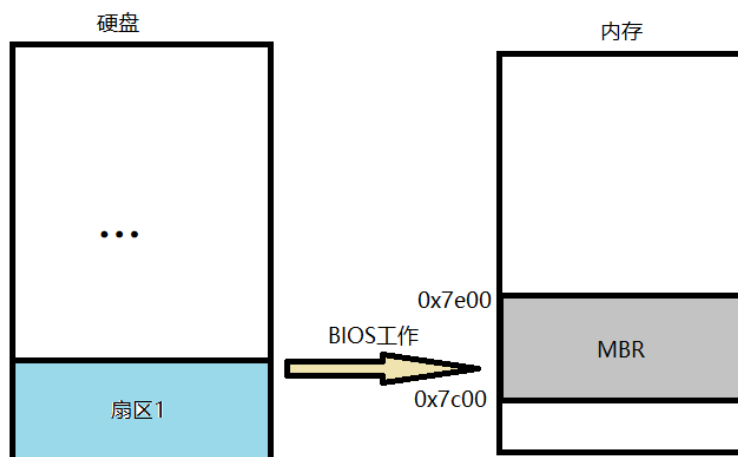
assignment 1

主要内容

复现example1即加载bootloader到内存

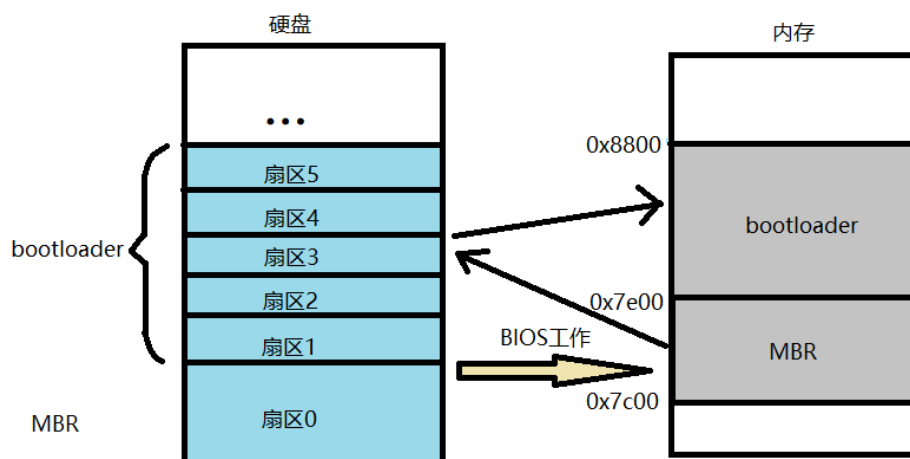
原理

电脑开机，首先CPU指向BIOS，BIOS是个无情的代码搬运工，将启动区（硬盘中的第一扇区,512byte）搬运到了内存0x7c00，同时跳转到对应位置开始执行。



MBR的内容：继续搬运一个文件叫做bootloader (linux0.11里面bootsector) 到内存的另一个后面相邻位置，执行MBR就可以平滑过渡到bootloader继续运行bootloader的内容。为什么要这样做呢？

我的理解是BIOS只能搬运512个字节或者这样更快启动，然后MBR再继续搬运几个扇区内容来内存运行完成初始化。



实验步骤

编写bootloader

bootloader内容提前设置为输出打印字符，用以检验是否成功运行。

```
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:
```

编写mbr

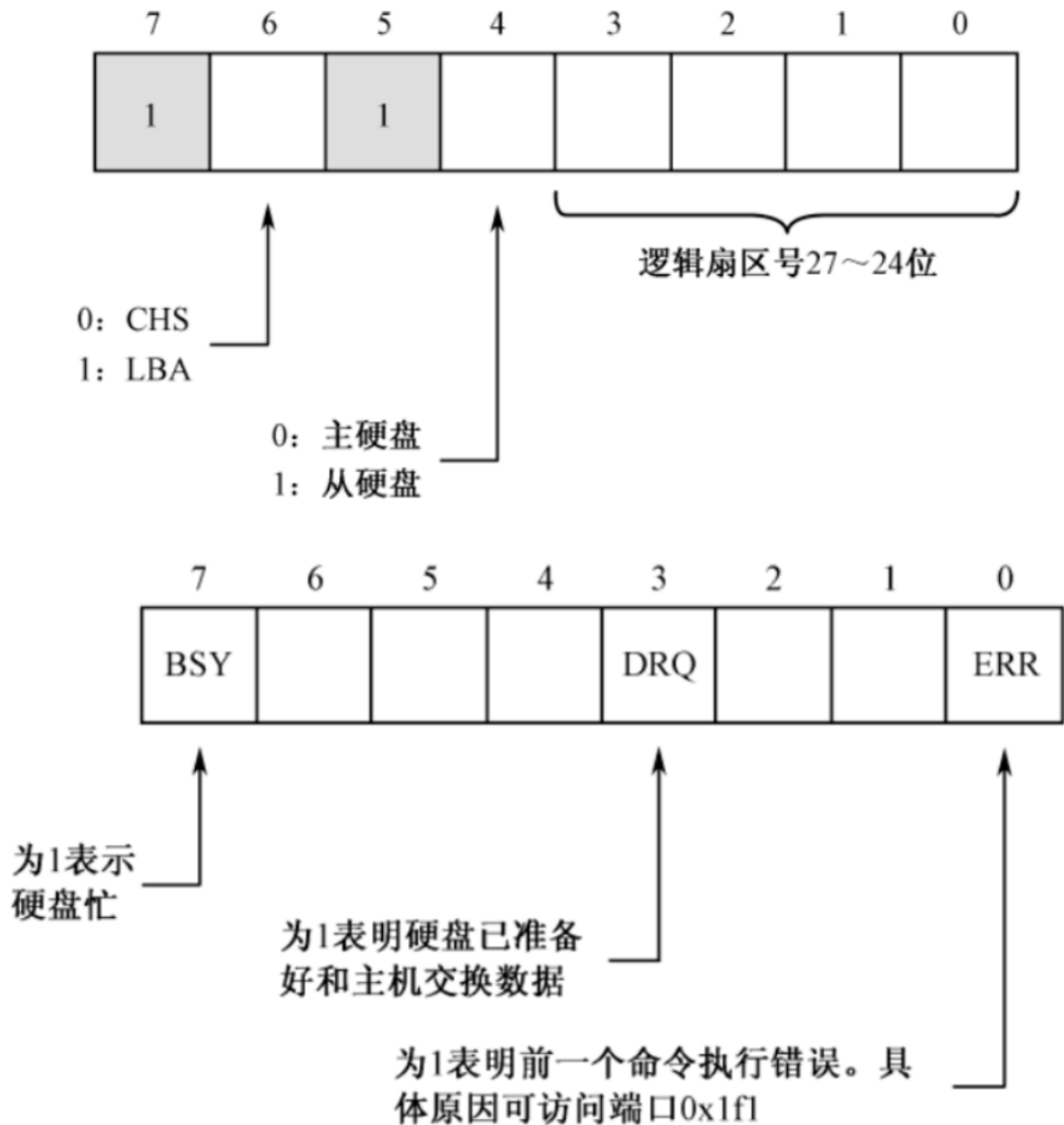
初始化各个寄存器，读取硬盘1-5扇区到0x7e00-0x8800

example1读取内存方式使用LBA模式

LBA读取硬盘规则如下

端口	功能	规则
0×1F0	缓存	暂存硬盘的一个byte
0×1F2	扇区数	
0×1F3	0-7位	
0×1F4	8-15位	
0×1F5	16-23位	
0×1F6	24-27位	低4位放地址，高4位特殊
0×1F7	执行及状态	写入0x20读取，其后表示状态

下图为0x1F6和0x1F7



实现一个读取扇区到指定内存函数

```
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

```

改用CHS模式读写

用到两个知识

1. int 13h功能号02h
2. LBA模式和CHS模式转换

int 13h 读磁盘

寄存器	ah	al	ch	dh	cl	dl	es:bx
内容	2h	扇区数	柱面 (C)	磁头 (H)	扇区 (S)	驱动器	缓冲区地址

LBA \longleftrightarrow CHS

1个柱面有PH磁头，1个磁头有PS扇区

柱面起始编号HS，磁头起始编号HS，扇区起始编号SS

1. CHS \rightarrow LBA

$$LBA = (C-CS) \times PH \times PS + (H-HS) \times PS + (S-SS)$$

2. LBA \rightarrow CHS

$$C = LBA / (PH \times PS) + CS$$

$$H = (LBA / PS) \pmod{PH} + HS$$

$$S = LBA \pmod{PS} + SS$$

将上面读取函数可以改写成这样

```

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(16)
    ; cx=逻辑扇区号16~28位(13)
    ; ds:bx=读取出的数据放入地址

```

; 返回值

; bx=bx+512

```
pushad                                ;keep the origin register
llba dw 0
hlba dw 0
address dw 0
mov [llba],ax                        ;llba get the 0-15 1 word
mov ax,bx
mov [address],ax                    ;address has got the address

mov bx,ds
mov es,bx                            ;change the es=ds

mov ax,cx
mov [hlba],ax                        ;hlba get the 16-28 1 word

mov edx,0
mov eax,0
add ax,[hlba]
shl eax,16
add eax,[llba]                      ;now the eax get the 0~28bit LBA4
mov ebx,1134
idiv ebx                            ;1134=18*63 now the C is stored in eax
cstore dw 0
mov byte[cstore],al

mov edx,0
mov eax,0
mov eax,[hlba]
shl eax,16
add eax,[llba]
mov ebx,63
idiv ebx                            ;now the result was stored in eax: LBA/PS
mov edx,0
mov ebx,18
idiv ebx                            ;now the answer was stored in edx: (LBA/PS)%PH
hstore dw 0
mov al,dl
```

```

mov byte[hstore],al

mov edx,0
mov eax,0
add eax,[hlba]
shl eax,16
add eax,[llba]
mov ebx,63
idiv ebx      ;now the result was stored in edx: LBA%ps
add edx,1
sstore dw 0
mov al,dl
mov byte[sstore],al

mov ch,byte[cstore]      ;c
mov dh,byte[hstore]      ;h
mov cl,byte[sstore]      ;s
mov bx,[address]
mov al,1      ;s number
mov ah,02h    ;function number
mov dl,80h    ;driver number
int 13h      ;es:bx
popad
add bx,512
ret

```

以上代码改了又改，最终才写成。发现几个会导致BUG的行为：

1. int 13h功能号读取磁盘后int 10h输出不会有结果
2. mov [标号], 寄存器：寄存器的选用请慎重
3. db声明一个标号后续起不了作用

[实验问题分析](#)（有的已经有答案有的还没有）

assignment2

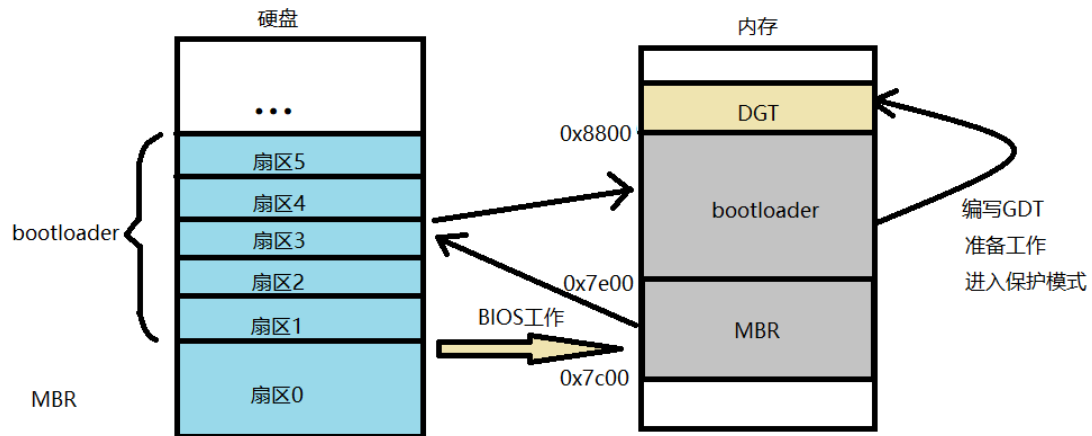
主要内容

复现example2（进入保护模式4步）并用gdb调试

原理

进入保护模式步骤：

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



实验步骤

MBR载入并运行至bootloader

编写bootloader

(其中各种地址已经在boot.inc定义)

```
%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 ; 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1个字节
```

```
;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
```

```
;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
```

```
;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

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:

```

闲言少叙，直接上gdb调试

为了方便，我们可以在makefile 上面写个make debug 方便调试
qemu启动，延时一秒让gdb可以连上并且帮我们加载符号表。

```

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

```

```
sleep 1
gnome-terminal -e "gdb -q -x gdbinit"
```

gdbinit

这里原实验文档有错误，第四句地址要改为0x7e00

```
target remote:1234
set disassembly-flavor intel
add-symbol-file mbr.symbol 0x7c00
add-symbol-file bootloader.symbol 0x7e00
```

简单说下生成mbr符号表过程（其他同理）

注释掉org指令 → 终端中输入

```
nasm -o mbr.o -g -f elf32 mbr.asm
ld -o mbr.symbol -melf_i386 -N mbr.o -Ttext 0x7c00
ld -o mbr.bin -melf_i386 -N mbr.o -Ttext 0x7c00 --oformat binary
```

注意：生成-o文件前要注释掉org指令，但是在写入磁盘的那个bin文件要有org编译来

正式开始gdb调试

1.完成GDT写入，设置好gdt

直接写入GDT的开始地址（这里是0x8800），然后写好地址和界限在gdt

```
终端
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

bootloader.asm
13      inc esi
14      add ebx,2
15      loop output_bootloader_tag
16
B+ 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
22      mov dword [GDT_START_ADDRESS+0x08],0x0000ffff ; 基地址为0，
23      mov dword [GDT_START_ADDRESS+0x0c],0x00cf9200 ; 粒度为4KB，
24
25      ;建立保护模式下的堆栈段描述符

remote Thread 1 In: output_bootloader_tag L18 PC: 0x7e24
$3 = 4
(gdb) p sizeof(long long)
$4 = 8
(gdb) x/5xg 0x8800
0x8800: 0x0000000000000000 0x0000000000000000
0x8810: 0x0000000000000000 0x0000000000000000
0x8820: 0x0000000000000000
(gdb) 
```

```
终端
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

bootloader.asm
29      ;建立保护模式下的显存描述符
30      mov dword [GDT_START_ADDRESS+0x18],0x80007fff ; 基地址为0x00
31      mov dword [GDT_START_ADDRESS+0x1c],0x0040920b ; 粒度为字节
32
33      ;创建保护模式下平坦模式代码段描述符
34      mov dword [GDT_START_ADDRESS+0x20],0x0000ffff ; 基地址为0，
35      mov dword [GDT_START_ADDRESS+0x24],0x00cf9800 ; 粒度为4kb，
36
37      ;初始化描述符表寄存器GDTR
> 38      mov word [pgdt], 39 ;描述符表的界限
39      lgdt [pgdt]
40
41      in al,0x92 ;南桥芯片内的端口

remote Thread 1 In: output_bootloader_tag L38 PC: 0x7e7e
(gdb) n
(gdb) n
(gdb) n
(gdb) x/5xg 0x8800
0x8800: 0x0000000000000000 0x00cf92000000ffff
0x8810: 0x0040960000000000 0x0040920b80007fff
0x8820: 0x00cf98000000ffff
(gdb) 
```

```

终端
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

bootloader.asm
31     mov dword [GDT_START_ADDRESS+0x1c],0x0040920b    ; 粒度为字节
32
33     ;创建保护模式下平坦模式代码段描述符
34     mov dword [GDT_START_ADDRESS+0x20],0x0000ffff    ; 基地址为0,
35     mov dword [GDT_START_ADDRESS+0x24],0x00cf9800    ; 粒度为4kb,
36
37     ;初始化描述符表寄存器GDTR
38     mov word [pgdt], 39    ;描述符表的界限
39     lgdt [pgdt]
40
> 41     in al,0x92    ;南桥芯片内的端口
42     or al,0000_0010B
43     out 0x92,al    ;打开A20

remote Thread 1 In: output_bootloader_tag L41 PC: 0x7e89
$12 = 0x0
(gdb) p/g $scr0
Size letters are meaningless in "print" command.
(gdb) x pgdt
0x7ed8 <pgdt>: 0x27
(gdb) x/6b pgdt
0x7ed8 <pgdt>: 0x27 0x00 0x00 0x88 0x00 0x00
(gdb)

```

31	16 15	0
段基地址 15~0		段界限 15~0

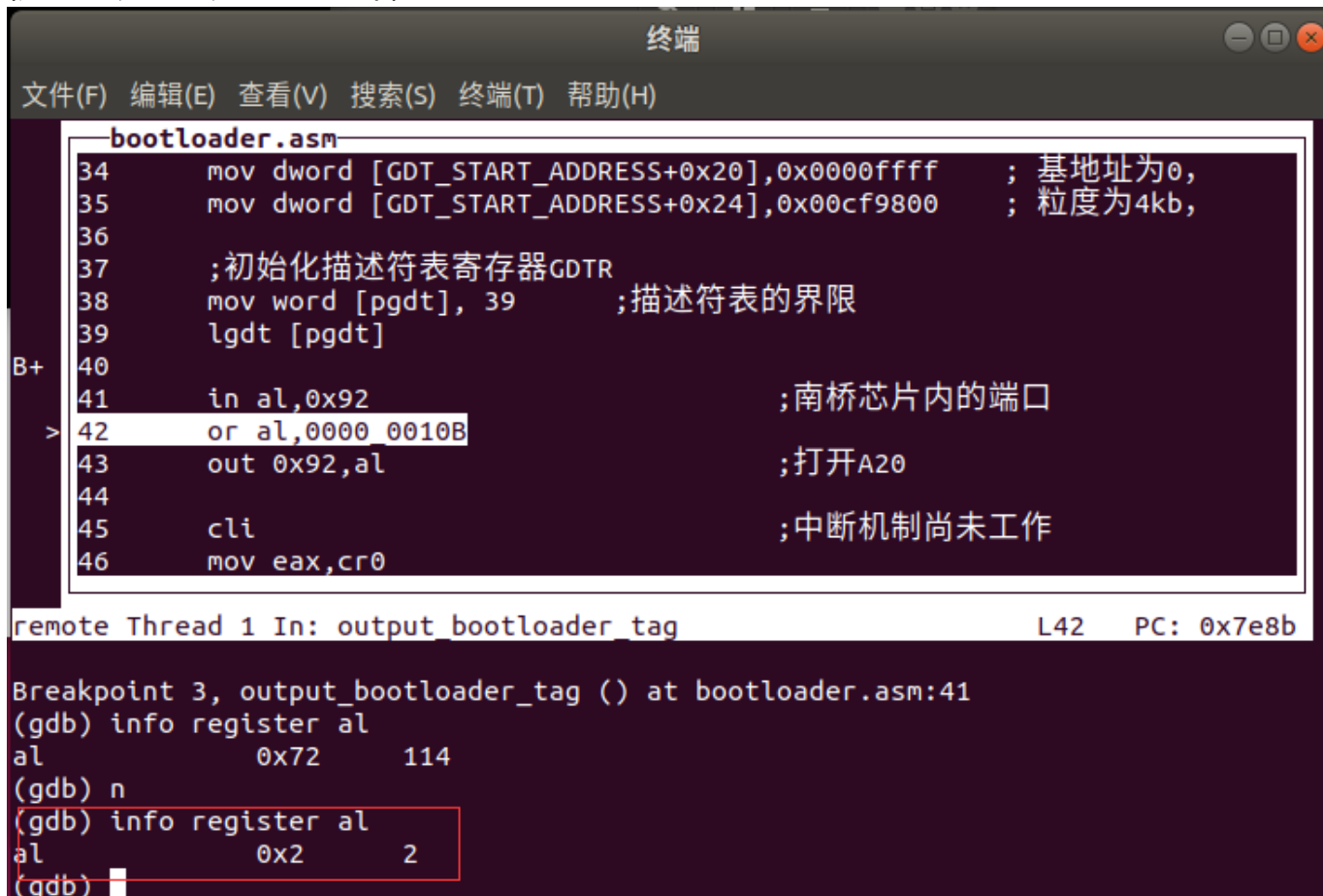
2.打开第21条地址线

设置最低位PE=1，打开地址线

这一步查阅资料以后发现可能**没有必要**，只当是是一个惯例执行了。

将低位数起第二位设为1

(为什么没必要设置在QA4中)



The screenshot shows a terminal window titled "终端" (Terminal). The menu bar includes "文件(F)", "编辑(E)", "查看(V)", "搜索(S)", "终端(T)", and "帮助(H)". The main content area displays assembly code from a file named "bootloader.asm". The code includes comments in Chinese explaining the purpose of each instruction, such as setting the base address, granularity, and initializing the GDTR register. A GDB breakpoint is set at line 41, and the GDB output shows the state of the AL register before and after the breakpoint is hit. The register value changes from 0x72 to 0x2.

```
bootloader.asm
34     mov dword [GDT_START_ADDRESS+0x20],0x0000ffff    ; 基地址为0,
35     mov dword [GDT_START_ADDRESS+0x24],0x00cf9800    ; 粒度为4kb,
36
37     ;初始化描述符表寄存器GDTR
38     mov word [pgdt], 39        ;描述符表的界限
39     lgdt [pgdt]
40
41     in al,0x92                ;南桥芯片内的端口
42     or al,0000 0010B
43     out 0x92,al              ;打开A20
44
45     cli                      ;中断机制尚未工作
46     mov eax,cr0

remote Thread 1 In: output_bootloader_tag L42 PC: 0x7e8b

Breakpoint 3, output_bootloader_tag () at bootloader.asm:41
(gdb) info register al
al                0x72      114
(gdb) n
(gdb) info register al
al                0x2       2
(gdb)
```

3.开启cr0保护模式标志位

将cr0最低位 (PE) 设置为0

终端

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
bootloader.asm
B+ 40
41     in al,0x92                ;南桥芯片内的端口
42     or al,0000_0010B
43     out 0x92,al              ;打开A20
44
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
```

remote Thread 1 In: output bootloader_tag L47 PC: 0x7e93

eax	0x10	16
ecx	0x0	0
edx	0x80	128
ebx	0x1c	28
esp	0x7c00	0x7c00
ebp	0x0	0x0
esi	0x7eec	32492

---Type <return> to continue, or q <return> to quit---

终端

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

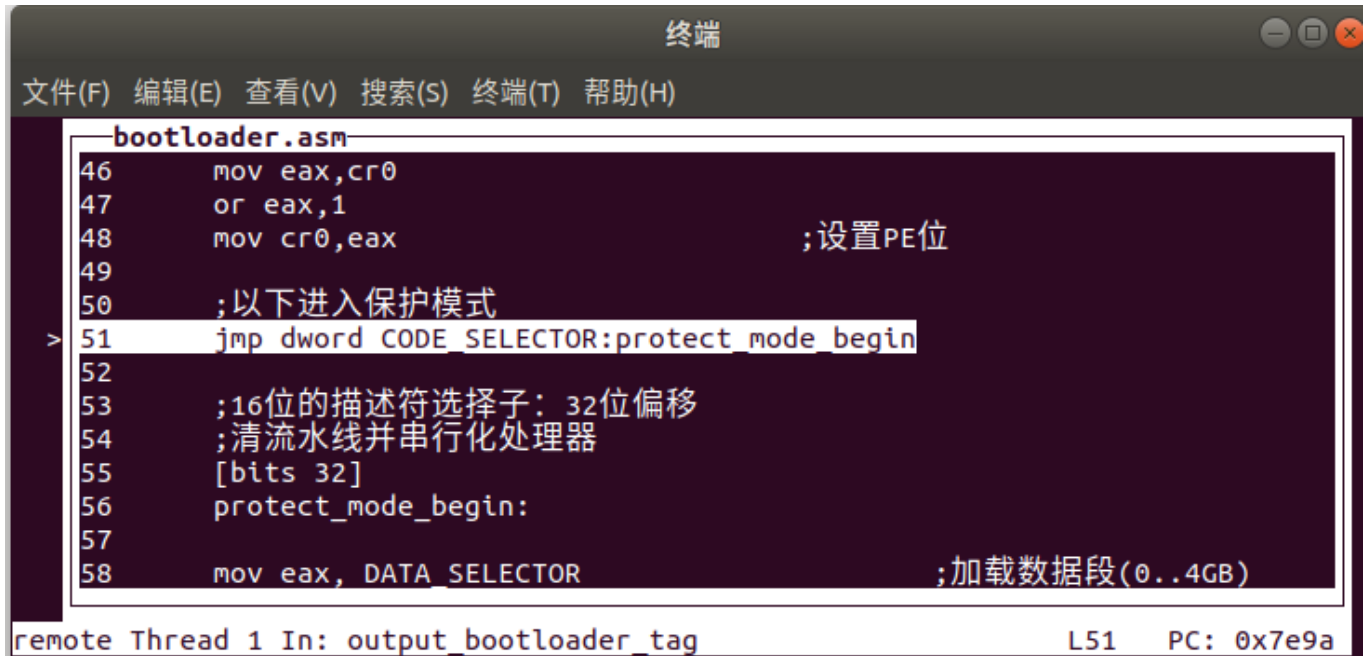
```
bootloader.asm
B+ 40
41     in al,0x92                ;南桥芯片内的端口
42     or al,0000_0010B
43     out 0x92,al              ;打开A20
44
45     cli                      ;中断机制尚未工作
46     mov eax,cr0
47     or eax,1
> 48     mov cr0,eax              ;设置PE位
49
50     ;以下进入保护模式
51     jmp dword CODE_SELECTOR:protect_mode_begin
52
```

remote Thread 1 In: output bootloader_tag L48 PC: 0x7e97

eax	0x11	17
ecx	0x0	0
edx	0x80	128
ebx	0x1c	28
esp	0x7c00	0x7c00
ebp	0x0	0x0
esi	0x7eec	32492

---Type <return> to continue, or q <return> to quit---

4. 执行远跳转



```
终端
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

bootloader.asm
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)

remote Thread 1 In: output_bootloader_tag L51 PC: 0x7e9a
```

assignment 3

主要内容

在保护模式下完成弹射汇编程序

原理

同样的四个方向弹射控制好边界条件转换就好。

区别：

- **关中断**条件下，显示字符**直接写入显存**，不要用Int 10h中断
- **关中断**条件下，实现时间延时可以用**循环**实现，不要用中断

实验步骤

我们直接用example2写下的框架进行

进入保护模式以后，我们写下4个循环代表各个方向来进行字符显示
rd(right down) ,ru (right up) ,ld(left down), lu(left up)

```
rd:
    cmp ebx,2*80*24
    jge ru
    mov edx,0
    mov ecx,eax          ;   eax-->ecx
```

```

mov eax,ebx
idiv word[temp]      ;now the col*2 is in edx
mov eax,ecx
cmp edx,79*2
je ld
add ebx,162
call number
call color
mov word[gs:ebx],ax
call delay            ; time delay
jmp rd

```

时间延时用循环完成

```

delay:
    pushad
    mov ecx, 0xffffffff ;set the time
delay_loop:
    dec ecx
    jnz delay_loop
    popad
    ret

```

实验问题分析

☒ QA1

Q: 使用int 13h读取磁盘以后int 10h中的输出指令还能用吗?

A: 在同一段的内容 (org指令) 下用是不行。

☒ QA2

Q: "mov [标号], 寄存器" 这条指令的寄存器使用有什么不知道的规则吗?

A: 经过测试, ax,cx,al是可以的。具体原因不清楚。

☐ QA3

Q: db声明一个标号, 后续会不起作用。为什么?

A: 等待解决。

☒ QA4

Q：打开第21条地址线是怎样一个过程？为什么说没必要？

A：具体可以看下图，但是其实处理器默认打开A20。

（省略一点内容）

