



# 本科生实验报告

实验课程:操作系统

实验名称:lab3 从实模式到保护模式

专业名称:计算机科学与技术(人工智能与大数据)

学生姓名:卜海涛

学生学号:22336016

实验地点:实验楼 E 栋

实验成绩:

报告时间:2024 年 3 月 31 日

## 1. 实验要求

学习如何使用 I/O 端口与硬件进行交互，利用 LBA 模式以及硬盘中断两种方法利用 mbr 将 bootloader 程序从硬盘中读入内存；在 bootloader 程序中编写段描述符以及进入保护模式的四个步骤，最后进入保护模式

## 2. 实验过程

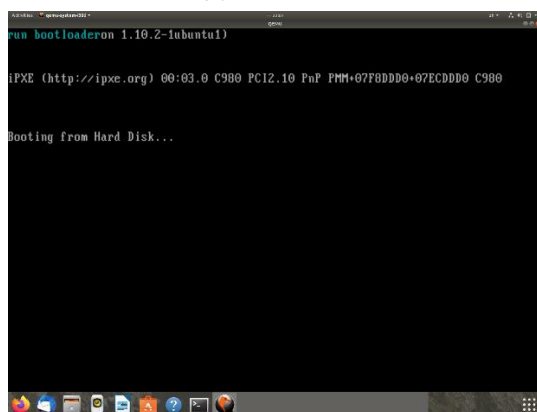
### 任务 1

#### 1.1

首先编写一个 bootloader 程序，这个程序在硬盘占 mbr 后的 5 个扇区，能向屏幕输出一个字符串

编写 mbr 程序，这个程序利用 LBA 方式从硬盘中读取 bootloader 所在的 5 个扇区到内存，读取完之后程序计数器跳转到 bootloader 所在地址开始执行 bootloader

运行 mbr 程序得到结果



#### 1.2

bootloader 程序无需更改

在 mbr 文件中将 LBA 模式更改为 CHS 模式来读取磁盘内容，此时不再需要手动指定端口，但是需要调用 int13 中断中的 02h 中断，手动指定柱面、扇区以及磁道来读取。

其中 LBA 到 CHS 的转换公式为  $LBA = (\text{柱面} * 18 + \text{磁道}) * 63 + \text{扇区} - 1$

```

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 dx, ax
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax

; 初始化栈指针
mov sp, 0x7c00

mov bx, 0x7c00
; 确定读入内容的初始地址

interrupt:
mov ah, 2
mov al, 1
mov ch, 0
mov cl, 2
mov dh, 0
mov di, 80h
int 13h
; 一次只能读一个扇区, 因此要写个循环

inc di
add bx, 0x200
; 每读取一个扇区要更新写入地址, 即加512byte

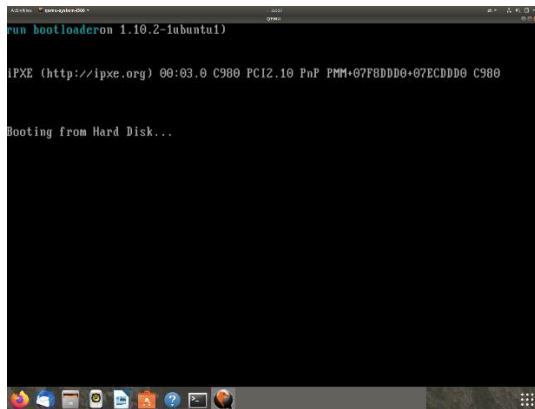
loop:
cmp cl, 6
je interrupt

times 510 - ($ - $$) db 0
db 0x55, 0xaa
; 从当前位置到第510个字节用0填充, 在最后两个字节写入相应字符, mbr要占够512B的空间

jmp 0x0000:0x7c00
; 跳转到bootloader开始运行

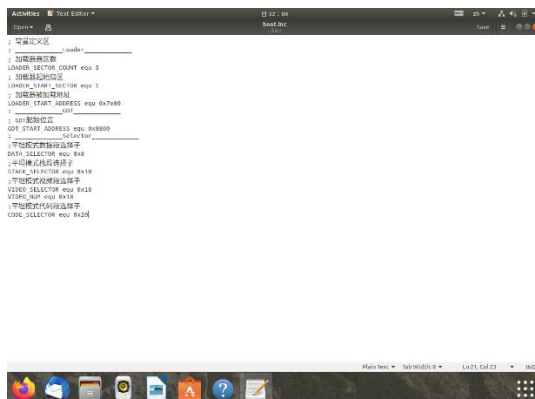
```

## 最后运行结果为



## 任务 2

首先将需要用到的常量放到一个独立文件, 提高代码可读性



在 bootloader 程序中定义段描述符组成段描述符表, 使 cpu 能够利用段选择子查找相应段的位置

```

;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0xFFFF
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0x00000000，界限0x7FFF
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0xFFFF
mov dword [GDT_START_ADDRESS+0x24],0x00cf9800 ; 粒度为4kb，代码段描述符

```

将 gdt 的大小和内容保存在 48 位寄存器 gdtr 中，CPU 通过 gdtr 寄存器访问 gdt



```

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

```

打开第 21 位地址线，由于实模式只有 20 位地址总线，第 21 位要置 0 以防止地址溢出，现在进入保护模式要将其打开

```

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

```

接着将保护模式真正的开关 CR0 寄存器的首位置 1，表示 CPU 进入保护模式

```

cli ; 保护模式下中断机制尚未建立，应禁止中断
mov eax, cr0
or eax, 1
mov cr0, eax ; 设置 PE 位

```

要想执行保护模式的代码，我们应该把代码段的选择子放到 cs 寄存器中，但是 cs 无法使用 mov 进行修改（其他段寄存器都可以），因此只能用跳转指令跳转到保护模式代码段的首地址

```

jmp dword CODE_SELECTOR:protect_mode_begin

```

(地址是 32 位，要用 dword，两个字，四个字节。32 位)

将其余段选择子用 mov 放入相应寄存器

```

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

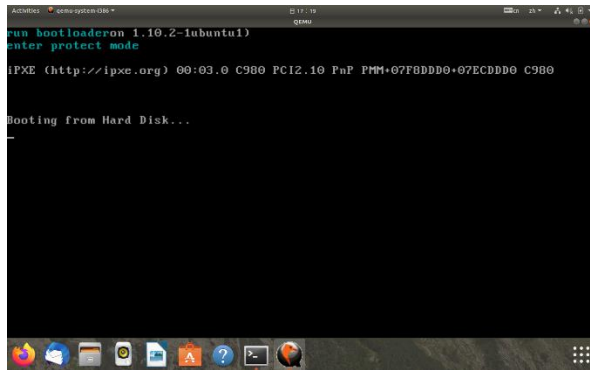
```

最后在保护模式代码段编写一个字符显示程序用来判断代码是否被执行

```
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
```

更改任务 1 的 mbr 程序，在调用读取硬盘函数时先把寄存器的值 push 到栈中，待子函数执行完毕再 pop 恢复，防止发生错误

观察程序运行结果



## GDB 调试

首先生成 mbr 和 bootloader 两个程序的符号表，即 .symbol 文件（.bin 文件中是不含符号表的），方便我们在调试时能够看到源码

```
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
（bootloader 类似）
```

将两个程序 bin 文件写入虚拟硬盘中

```
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
```

使用 qemu 加载 hd.img 运行

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

接着在另一个终端中启动 gdb（记住一定要在 lab3 目录下启动，否则后续会报错找不到 symbol 文件），连接 qemu 并设置断点

```
gdb
target remote:1234
```

b \*0x7c00

c

接着打开源码显示窗口，加载符号表后便可正式开始调试

layout src

add-symbol-file mbr.symbol 0x7c00

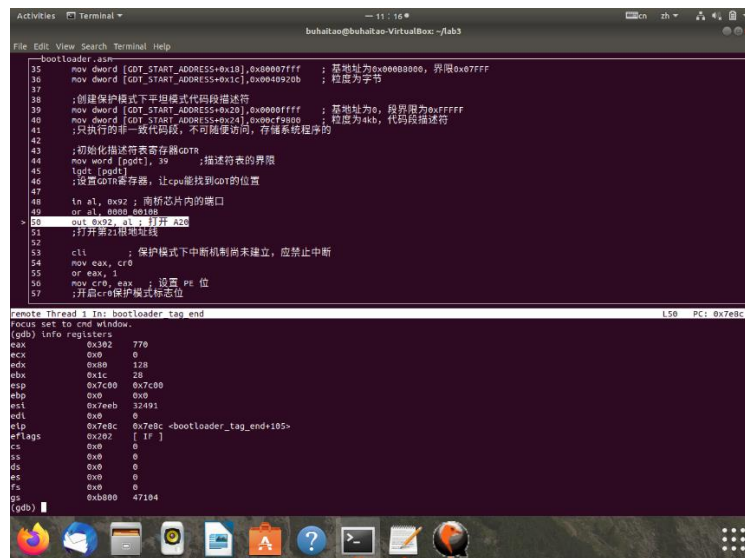
## 查看段描述符表是否在内存中

利用 x/5xg 0x8800 指令，表示从 0x8800 地址开始（段描述符是从这个地址开始存的），往后查看 5 个 g（8byte）的内存（即 5 个段描述符），观察到结果

```
remote Thread 1 In: protect_mode_begin L66 PC: 0x7ea1
Continuing.
Breakpoint 3, protect_mode_begin () at bootloader.asm:66
(gdb) x/5xg 0x8800
0x8800: 0x0000000000000000 0x00cf92000000ffff
0x8810: 0x0040960000000000 0x0040920b80007fff
0x8820: 0x00cf98000000ffff
(gdb)
```

## 打开第 21 根地址线

查看 eax 的值，发现第四位确实变成了 0010



```
bootloader.asm
35 mov dword [GDT_START_ADDRESS+0x10],0x00007fff ; 基地址为0x00000000,界限0x07fff
36 mov dword [GDT_START_ADDRESS+0x1c],0x0040920b ; 精度为字节
37
38 ;创建保护模式下平铺模式代码段描述符
39 mov dword [GDT_START_ADDRESS+0x20],0x000000ffff ; 基地址为0,段界限为0xffff
40 mov dword [GDT_START_ADDRESS+0x24],0x00cf9800 ; 精度为4kb,代码段描述符
41
42 ;只执行的单一代码段,不可能访问,存储系统程序的
43
44 ;初始化描述符表寄存器GDT
45 mov word [pgdt],39 ;描述符表的界限
46 lgdt [pgdt] ;设置GDT寄存器,让cpu能找到GDT的位置
47
48 in al,0x92 ;南桥芯片内的端口
49 or al,0x08000000
50 out 0x92,al ;打开A20
51
52 ;打开第21根地址线
53
54 cll ;保护模式下中断机制尚未建立,应禁止中断
55 mov eax,crl
56 or eax,1
57 mov cr0,edx ;设置PE位
58 ;开启cr0保护模式标志位

remote Thread 1 In: bootloader_tag_end L50 PC: 0x7e8c
focus set to cmd window.
(gdb) info registers
eax 0x302 770
ecx 0x0 0
edx 0x0 0
ebx 0x1c 28
esp 0x7c00 0x7c00
ebp 0x0 0
esi 0x7eeb 32491
edi 0x0 0
eip 0x7e8c 0x7e8c <bootloader_tag_end+10>
eflags 0x202 [ IF ]
cs 0x0 0
ss 0x0 0
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
(gdb)
```

## 开启 cr0 保护模式标志位

让程序单步执行到该部分代码，查看 eax 的值，观察到 eax 最低位为 1，保护模式开关打开



不断进行下去，颜色和字符内容每次加 1

```
loop:
mov  eax,dword[x]
mov  ebx,80
mul  ebx
add  eax,dword[y]
mov  ebx,2
mul  ebx
mov  ebx,eax
mov  ah,byte[color]
mov  al,byte[char]
mov  word[gs:ebx],ax
inc  byte[color]
inc  byte[char]
cmp  byte[char],'9'
jg  mod
cotiue:
mov  eax,dword[x]
add  eax,dword[xx]
mov  dword[x],eax
mov  eax,dword[y]
add  eax,dword[yy]
mov  dword[y],eax
```

四个模块分别处理触碰到左上右下四壁时的情况，使字符触碰边界反弹，处理方法是对步长取反，如 1 变为-1

```
bottom:
add  dword[x],-2
mov  dword[xx],-1
cmp  dword[y],80
je  right
cmp  dword[y],-1
je  left
jmp  delay
```

```
top:
add  dword[x],2
mov  dword[xx],1
cmp  dword[y],80
je  right
cmp  dword[y],-1
je  left
jmp  delay
```



```

right:
add dword[y],-2
mov dword[yy],-1
cmp dword[x],25
je bottom
cmp dword[x],-1
je top
jmp delay

```

```

left:
add dword[y],2
mov dword[yy],1
cmp dword[x],25
je bottom
cmp dword[x],-1
je top
jmp delay

```

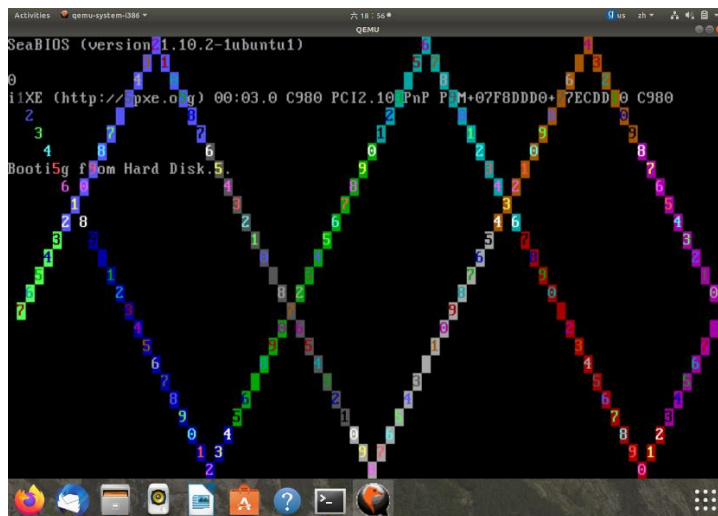
最后设置一个循环延迟，每次打印出一个字符串后执行一个循环延迟，保证弹射效果

```

delay:
mov ecx,100000000
loop2:
cmp ecx,0
je loop
dec ecx
jmp loop2

```

实验结果如下



### 3. 关键代码

## 任务 1.2

```
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 bx, 0x7e00
; 确定读入内容的初始地址

interrupt:
mov ah, 2
mov al, 1
mov ch, 0
mov cl, 2
mov dh, 0
mov dl, 80h
int 13h
; 一次只能读一个扇区, 因此要写个循环

inc cl
add bx, 0x200
; 每读取一个扇区要更新写入地址, 即加 512byte

loop:
cmp cl, 6
jle interrupt

times 510 - ($ - $$) db 0
db 0x55, 0xaa
; 从当前位置到第 510 个字节用 0 填充, 在最后两个字节写入相应
字符, mbr 要占够 512B 的空间

jmp 0x0000:0x7e00
; 跳转到 bootloader 开始运行
```

## 任务 2 bootloader

```
%include "boot.inc"

[bits 16]
mov ax, 0xb800
mov gs, ax
mov ah, 0x03 ;青色
xor ebx, ebx
mov esi, bootloader_tag

loop:
cmp byte[esi], 0
je bootloader_tag_end
mov al, byte[esi]
mov word[gs:bx], ax
inc esi
add bx, 2
jmp loop

bootloader_tag_end:
;空描述符
mov dword [GDT_START_ADDRESS+0x00], 0x00
mov dword [GDT_START_ADDRESS+0x04], 0x00
;因为一个段描述符是有上下各 32 位组成的，所以这里分成两段，每
段分配两个字

;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 0xFFFF，因为高 32 位还有一个 f
mov dword [GDT_START_ADDRESS+0x0c], 0x00cf9200 ; 粒度为
4KB，存储器段描述符
;可读写并向上拓展的，共分配了  $16^5 \times 4KB = 4GB$  的内存

;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

;只执行的非一致代码段, 不可随便访问, 存储系统程序的

;初始化描述符表寄存器 GDTR

mov word [pgdt], 39 ;描述符表的界限

lgdt [pgdt]

;设置 GDTR 寄存器, 让 cpu 能找到 GDT 的位置

in al, 0x92 ; 南桥芯片内的端口

or al, 0000\_0010B

out 0x92, al ; 打开 A20

;打开第 21 根地址线

cli ; 保护模式下中断机制尚未建立, 应禁止中断

mov eax, cr0

or eax, 1

mov cr0, eax ; 设置 PE 位

;开启 cr0 保护模式标志位

;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

```

;下一行开始打印
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 $

pgdt dw 0
    dd GDT_START_ADDRESS
;分配 48 位内存表示 GDTR 内容

bootloader_tag db 'run bootloader',0
;用汇编 db 伪指令定义字符串要手动加 0
;数据段要放在代码段之后
;定义字符串、字符要用 db, 表示给每个字符分配一个 byte

protect_mode_tag db 'enter protect mode'
protect_mode_tag_end:

```

## 任务 2 mbr

```

load_bootloader:
    push ax
    push bx
    call asm_read_hard_disk ; 读取硬盘
    add sp, 4
    ;由于之前把 ax 和 bx 都 push 到了栈中, sp=sp-4, 此处要加回
去, 相当于 pop 掉了 ax 和 bx
    ;相当于在调用函数前给函数传递需要的参数
    inc ax
    add bx, 512
    loop load_bootloader

    jmp 0x0000:0x7e00 ; 跳转到 bootloader

jmp $ ; 死循环

; asm_read_hard_disk(memory, block)

```

; 加载逻辑扇区号为 block 的扇区到内存地址 memory

asm\_read\_hard\_disk:

```
    push bp
    mov bp, sp
```

```
    push ax
    push bx
    push cx
    push dx
```

```
    mov ax, [bp + 2 * 3] ; 逻辑扇区低 16 位
    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
```

```
    xor ax, ax
    inc dx        ; 0x1f5
    out dx, al    ; LBA 地址 23~16 = 0
```

```
    inc dx        ; 0x1f6
    mov al, ah
    and al, 0x0f
    or al, 0xe0    ; LBA 地址 27~24 = 0
    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 bx, [bp + 2 * 2]
        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

        pop dx
        pop cx
        pop bx
        pop ax
        pop bp

        ret

```

### 任务 3 字符弹射

```

; 让字符从 (2, 0) 处开始射出
loop:
    mov eax, dword[x]
    mov ebx, 80
    mul ebx
    add eax, dword[y]
    mov ebx, 2
    mul ebx
    mov ebx, eax
    mov ah, byte[color]
    mov al, byte[char]
    mov word[gs:ebx], ax
    inc byte[color]
    inc byte[char]
    cmp byte[char], '9'
    jg mod
cotiue:
    mov eax, dword[x]
    add eax, dword[xx]
    mov dword[x], eax
    mov eax, dword[y]
    add eax, dword[yy]

```

```
mov dword[y], eax
```

```
;若触碰到底部  
cmp dword[x], 25  
je bottom
```

```
;若触碰到顶部  
cmp dword[x], -1  
je top
```

```
;若触碰到右壁  
cmp dword[y], 80  
je right
```

```
;若触碰到左壁  
cmp dword[y], -1  
je left
```

```
jmp delay
```

```
bottom:  
add dword[x], -2  
mov dword[xx], -1  
cmp dword[y], 80  
je right  
cmp dword[y], -1  
je left  
jmp delay
```

```
top:  
add dword[x], 2  
mov dword[xx], 1  
cmp dword[y], 80  
je right  
cmp dword[y], -1  
je left  
jmp delay
```

```
right:  
add dword[y], -2  
mov dword[yy], -1  
cmp dword[x], 25  
je bottom  
cmp dword[x], -1
```



```

        je top
        jmp delay

left:
add dword[y], 2
mov dword[yy], 1
cmp dword[x], 25
je bottom
cmp dword[x], -1
je top
jmp delay

delay:
mov ecx, 100000000
loop2:
cmp ecx, 0
je loop
dec ecx
jmp loop2

mod:
sub byte[char], 10
jmp cotiue

```

## 4. 实验结果

见实验过程

## 5. 总结

相较于上次的汇编实验，这次用汇编代码写程序时明显要顺手的多，但是在实验过程中也碰到了一些汇编语法看不懂，不知道为什么要这么写的问题。另外，在实验过程中犯了一些比较隐蔽的错误，导致花费了很多时间去 debug，第一个就是在将程序写入虚拟硬盘时没有事先创建一个 10m 硬盘（以为可以使用之前实验的虚拟硬盘，但其实不在一个文件夹），虽然在将 bin 文件写入时不会报错，因为系统会自动给你创建一个虚拟硬盘，但是这个硬盘只有 0.8m 大小，这也导致了我后续字符显示不出来。第二个问题就是在启动 GDB 时不是在 lab3 目录下启动，导致后面 gdb 找不到 symbol 文件。这些错误都是没有一个良好编程习惯导致，后续会加强。再就是感谢这个清明假期让我有时间做这次的选做题，仔细探索后发现这个字符弹射程序并没有想象中困难，做出来之后感觉很有成就感！