本科生实验报告

实验课程：操作系统原理实验

实验名称：实模式下 OS 的启动

专业名称：计算机科学与技术（超算）

学生姓名：黄玟瑜

学生学号：19335074

实验地点：中山大学广州校区东校园

实验成绩：

报告时间：2021 年 3 月 21 日

1. 实验要求

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。

2. 实验内容

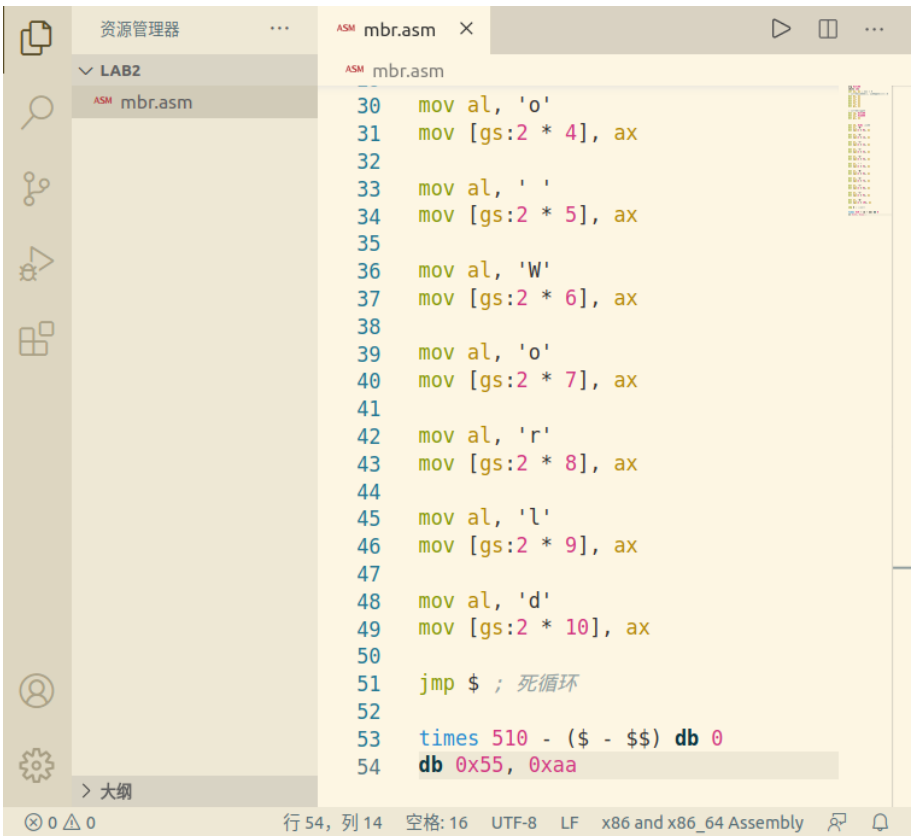
了解操作系统启动的原理， 利用汇编语言实模式即（20 位地址空间） 的启动和保护模式即（32 位地址空间） 下的启动方法，并能够在此基础上利用汇编或者 C 程序实现简单的应用；

1. 回顾、学习 32 位汇编语言的基本语法；
2. 编写简单的汇编程序，进行中断、输入输出测试；
3. 实现实模式下 OS 启动；
4. 在实模式下利用汇编/C/Rust 等实现简单的应用

3. 实验过程

Assignment 1·1

1、 新建文件夹，将 Example1 的代码粘贴进去保存



2、 使用 NASM 汇编器将其编译成二进制文件，命令如下：

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

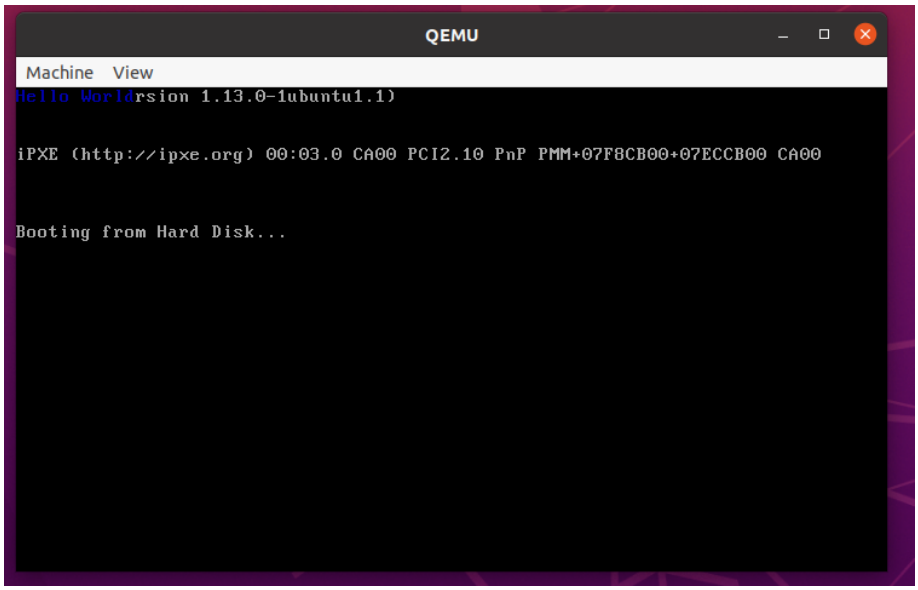
3、 使用 xxd 查看生成文件的内容，可以看到一长串 bit 流

```
wenny@owo:~/lab2$ nasm -f bin /home/wenny/lab2/mbr.asm -o mbr.bin
wenny@owo:~/lab2$ ls
mbr.asm  mbr.bin
wenny@owo:~/lab2$ xxd mbr.bin
00000000: 31c0 8ed8 8ed0 8ec0 8ee0 8ee8 bc00 7cb8  1.............|.
00000010: 00b8 8ee8 b401 b048 65a3 0000 b065 65a3  .......He....ee.
00000020: 0200 b06c 65a3 0400 b06c 65a3 0600 b06f  ...le....le....o
00000030: 65a3 0800 b020 65a3 0a00 b057 65a3 0c00  e.... e....We...
00000040: b06f 65a3 0e00 b072 65a3 1000 b06c 65a3  .oe....re....le.
00000050: 1200 b064 65a3 1400 ebfe 0000 0000 0000  ...de...........
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000190: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001f0: 0000 0000 0000 0000 0000 0000 0000 55aa  ..............U.
wenny@owo:~/lab2$
```

4、 使用以下指令创建虚拟硬盘，将生成的二进制文件 mbr.bin 写入该虚拟硬盘中

```
wenny@owo:~/lab2$ qemu-img create hd.img 10m
Formatting 'hd.img', fmt=raw size=10485760
wenny@owo:~/lab2$ ls
hd.img  mbr.asm  mbr.bin
wenny@owo:~/lab2$ dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=no
trunc
记录了1+0 的读入
记录了1+0 的写出
512字节已复制，0.00461121 s，111 kB/s
wenny@owo:~/lab2$ qemu-system-i386 -hda hd.img -serial null -parallel s
tdio
WARNING: Image format was not specified for 'hd.img' and probing guesse
d raw.
         Automatically detecting the format is dangerous for raw images
, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions
.
```

5、 使用 qemu 启动，将 hd.img 指定为第 0 号磁盘映像，启动后效果如下：



Assignment 1·2

1、 参考实验指导，根据以下公式计算出（12，12）的显存起始位置应为 0xB8000+2×972

$$\text{显存起始位置} = \text{0xB8000} + 2 \cdot (80 \cdot x + y)$$

2、 设置显示为白底蓝字，则放在高 8 位的编码应为 01110011b，即 0x73

3、 重复上述启动过程，结果如下，可以看到，在（12，12）的位置清楚的显示出了我的学号

Assignment 2·1

1、 使用实模式下的中断来输出你的学号，查阅 Bios 中断向量表，定位到显示服务（INT 10H），找到所需要的中断类型为 13 号中断：在 Teletype 模式下显示字符串

mbr1.asm

```
org 0x7c00
[bits 16]

mov ax, cs                      ; BIOS将代码加载到内存0x7c00处，因此段地址为0x7c
mov ds, ax                      ; 初始化数据段
mov es, ax                      ; 用于指向字符串

mov ax, Message                 ; es:bp指向要显示的字符串
mov bp, ax
mov ah, 0x13                    ; ah为0x13,调用13号中断
mov al, 0x01                    ; al为0,不移动光标，字符串中没有属性内容

mov bh, 0                       ; 第0页显示
mov bl, 0x57                    ; color

mov cx, MessageLen              ; 字符串长度
mov dx, 0x0c0c                  ; dh=12, dl=12, 即第12行,第12列
int 10h                         ; 调用10H中断

jmp $                           ; 无限循环,防止代码进入数据区

Message db "19335074"
MessageLen equ $ - Message


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

ES:BP = 串地址

CX = 串长度

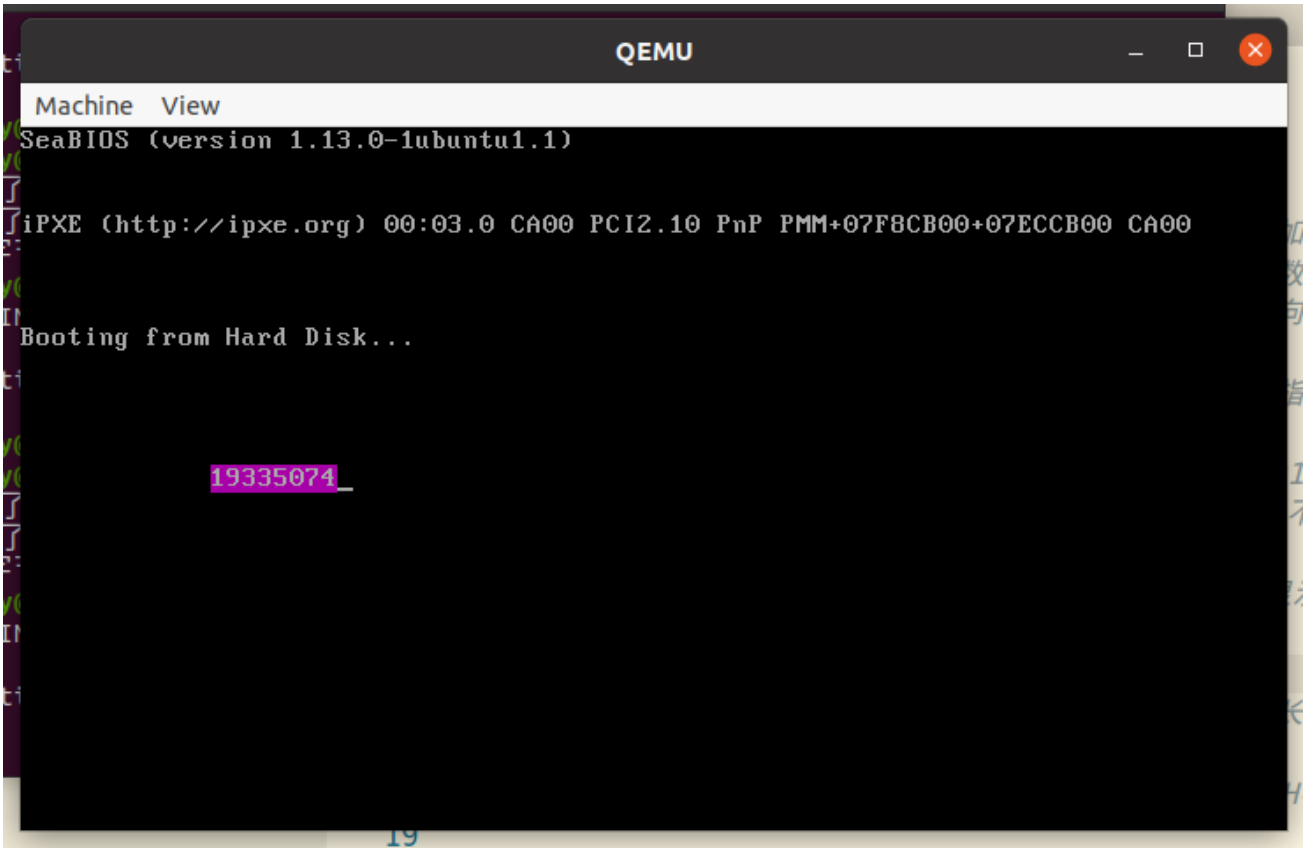
DH， DL = 起始行列

BH = 页号

2、 重复上述启动过程

```
wenny@owo:~/lab2$ nasm -f bin mbr1.asm -o mbr1.bin
wenny@owo:~/lab2$ dd if=mbr1.bin of=hd.img bs=512 seek=0 conv=notrunc count=1
记录了1+0 的读入
记录了1+0 的写出
512字节已复制，0.000549441 s，932 kB/s
wenny@owo:~/lab2$ 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.
```

3、 运行结果如下



可以看到，在（12，12）的位置清楚的显示出了我的学号。

Assignment 2·2

1、 利用中断实现光标的位置获取，查阅 Bios 中断向量表，定位到显示服务（INT 10H），找到所需要的中断类型为 3 号中断：读光标位置

其中 `BH = 页号、CH = 光标开始行、CL = 光标结束行、DH = 行、DL = 列`

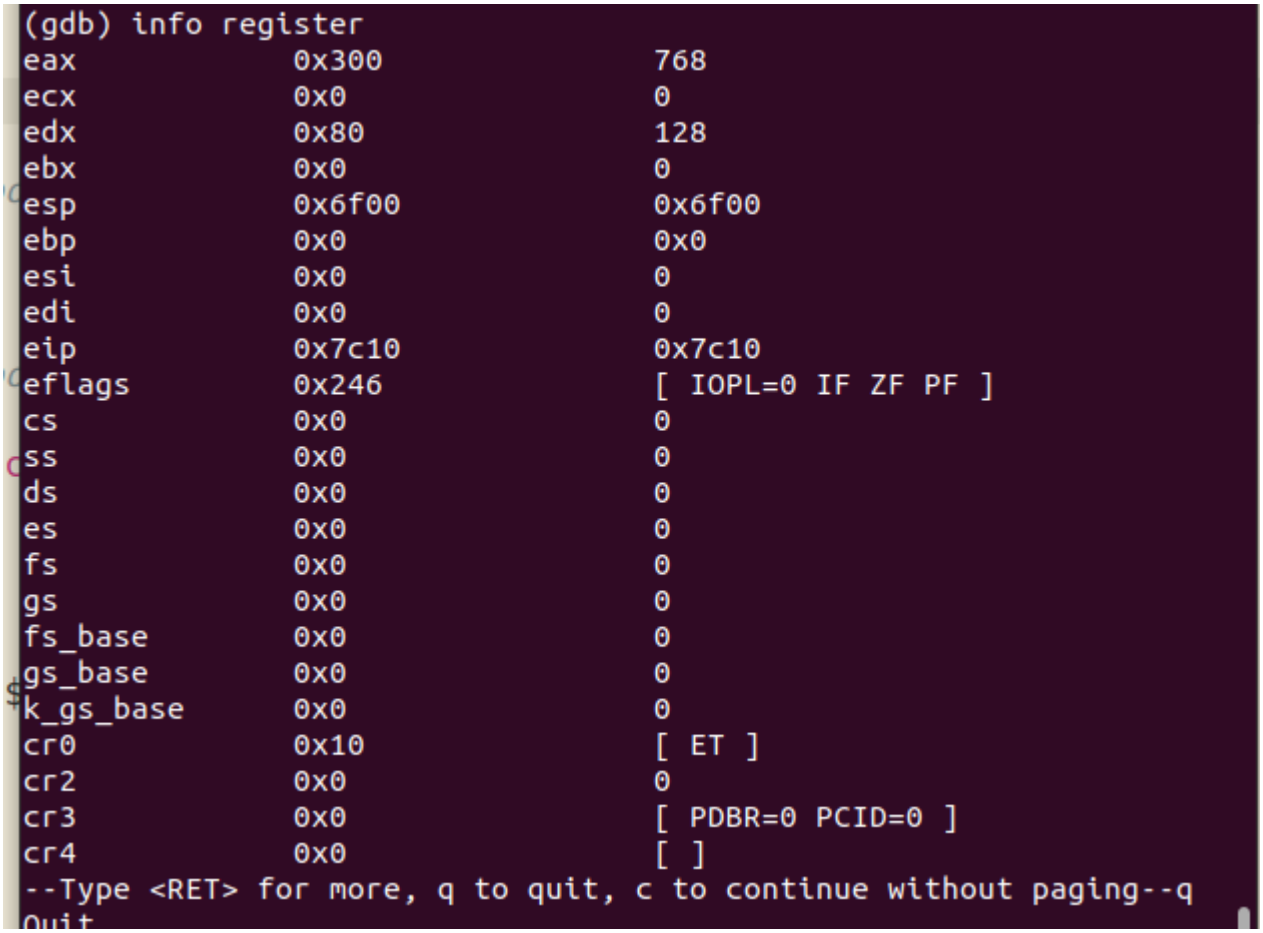
初始化前，各个寄存器的值如下：

```
Breakpoint 1, 0x00007c00 in ?? ()
(gdb) info register
eax            0xaa55             43605
ecx            0x0                0
edx            0x80               128
ebx            0x0                0
esp            0x6f00             0x6f00
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x7c00             0x7c00
eflags         0x202              [ IOPL=0 IF ]
cs             0x0                0
ss             0x0                0
ds             0x0                0
es             0x0                0
fs             0x0                0
gs             0x0                0
fs_base        0x0                0
gs_base        0x0                0
k_gs_base      0x0                0
cr0            0x10               [ ET ]
cr2            0x0                0
cr3            0x0                [ PDBR=0 PCID=0 ]
cr4            0x0                [ ]
```

初始化后，各个寄存器的值如下，可以看到段寄存器和 AX 都被置为 0

```
(gdb) info register
eax            0x0                0
ecx            0x0                0
edx            0x80               128
ebx            0x0                0
esp            0x6f00             0x6f00
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x7c0a             0x7c0a
eflags         0x246              [ IOPL=0 IF ZF PF ]
cs             0x0                0
ss             0x0                0
ds             0x0                0
es             0x0                0
fs             0x0                0
gs             0x0                0
fs_base        0x0                0
gs_base        0x0                0
k_gs_base      0x0                0
cr0            0x10               [ ET ]
cr2            0x0                0
cr3            0x0                [ PDBR=0 PCID=0 ]
cr4            0x0                [ ]
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
```

调用中断，获得的光标位置如下，其中页号为 0、光标开始行为 0、光标结束行为 0、光标在是第 0 行第 128 列



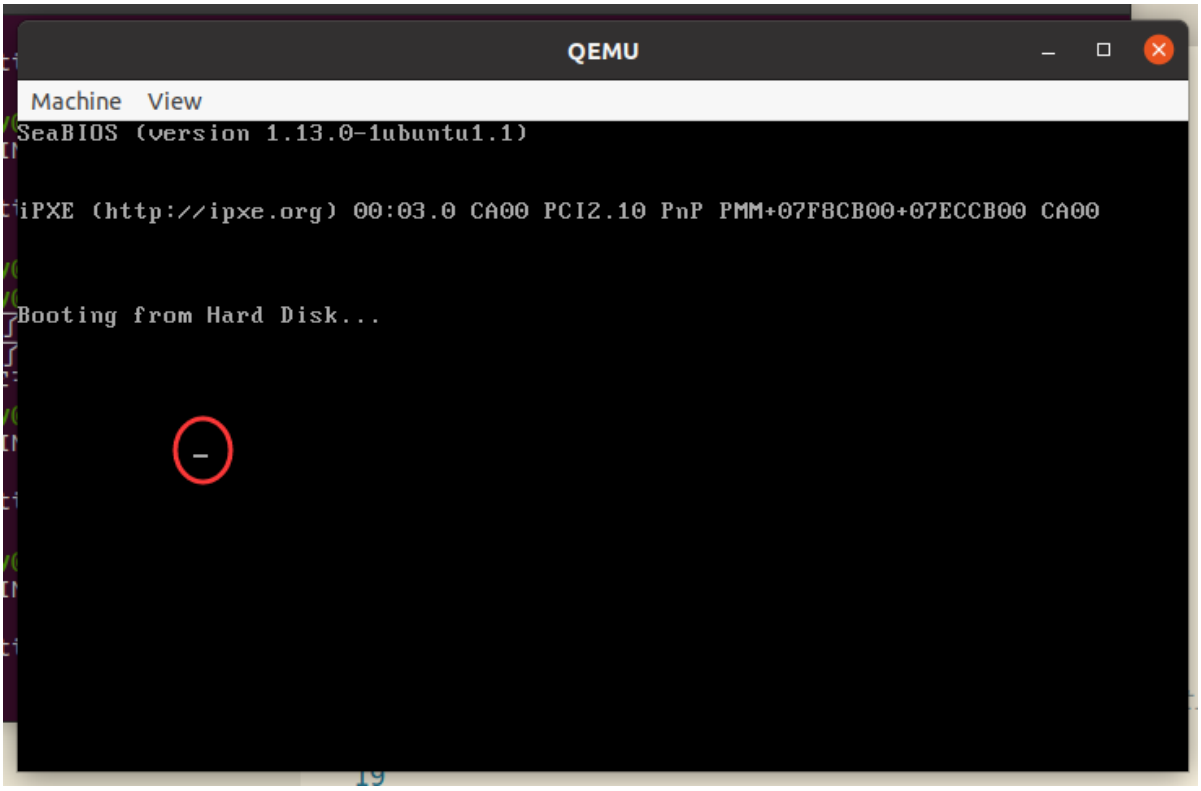
2、 利用中断实现光标的移动，查阅 Bios 中断向量表，定位到显示服务（INT 10H）找到所需要的中断类型为 2 号中断：置光标位置

BH = 页号、DH = 行、DL = 列

将 DX 置为 0X0C0C， 即（12，12）的位置，

```
(gdb) info register
eax            0x200              512
ecx            0x607              1543
edx            0xc0c              3084
ebx            0x0                0
esp            0x6f00             0x6f00
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x7c19             0x7c19
eflags         0x246              [ IOPL=0 IF ZF PF ]
cs             0x0                0
ss             0x0                0
ds             0x0                0
es             0x0                0
fs             0x0                0
gs             0x0                0
fs_base        0x0                0
gs_base        0x0                0
k_gs_base      0x0                0
cr0            0x10               [ ET ]
cr2            0x0                0
cr3            0x0                [ PDBR=0 PCID=0 ]
cr4            0x0                [ ]
cr8            0x0                0
efer           0x0                [ ]
xmm0           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0,
```

标位置调用中断后，可以看到光标的的位置被置到（12，12）的位置上



Assignment 2·3

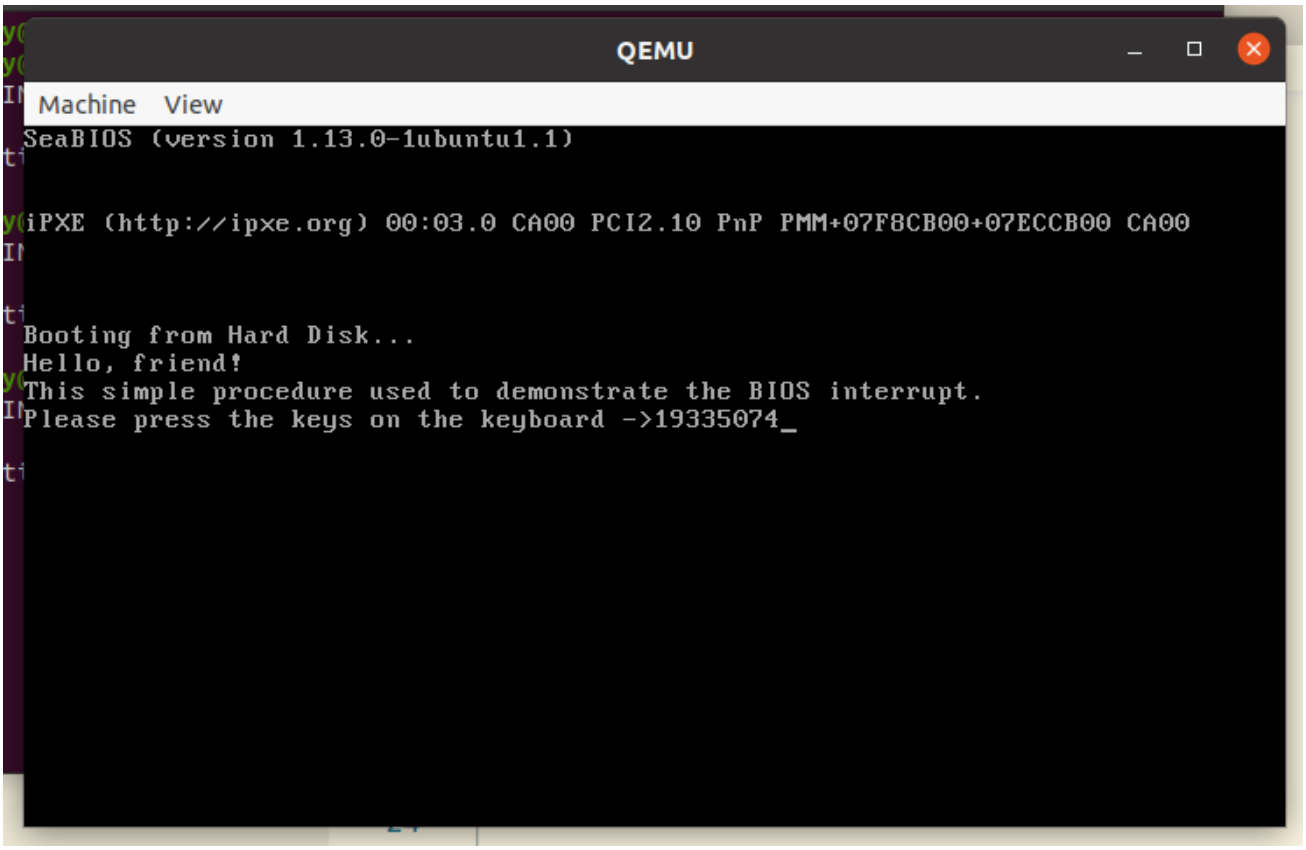
1、 利用中断实现键盘输入，查阅 Bios 中断向量表，定位到 INT 16H，找到所需要的中断类型为 0 号中断：键盘读取字符

```
;input
    mov ah,0x00
    int 0x16
```

2、 利用中断实现键盘回显，查阅 Bios 中断向量表，定位到 INT 10H，找到所需要的中断类型为 14 号中断：显示字符

```
;diaplay
    mov ah,0x0e
    mov bl,0x07
    int 0x10
```

结果如下所示



Assignment 3·1

```
 8  your_if:
 9      mov eax, [a1]
10      cmp eax,12
11      jl c1
12      mov eax, [a1]
13      cmp eax,24
14      jl c2
15      jmp c3
16  c1:
17      mov eax, [a1]
18      xor edx, edx
19      mov ebx, 2
20      idiv ebx
21      add eax, 1
22      mov [if_flag], eax
23      jmp end_if
24  c2:
25      mov eax, [a1]
26      sub eax, 24
27      neg eax
28      mov ebx, [a1]
29      imul eax, ebx
30      mov [if_flag], eax
31      jmp end_if
32  c3:
33      mov eax, [a1]
34      shl eax, 4
35      mov [if_flag], eax
36      jmp end_if
37  end_if:
```

根据 a1 的值分 3 钟情况，c1、c2、c3，在每次判断后跳转到相应的情况并执行代码，执行完毕后跳转到末尾。

Assignment 3·2

```
your_while:
    mov edx, [a2]
    cmp edx, 12
    jl end_while
    call my_random
    mov ebx, [while_flag]
    mov ecx, [a2]
    mov byte[ebx+(ecx-12)*1], al
    mov edx, [a2]
    dec edx
    mov [a2], edx
    jmp your_while
end_while:
    ; ret
```

循环前将 a2 和 12 相比较，若不满足 a2>=12 则跳转到程序尾部，退出循环。将 eax 的低 8 位以字节的长度写入对应内存中，防止其他字符的位置被写入错误信息。

Assignment 3·3

```
56  your_function:
57      mov eax,0   ;i
58  my_loop:
59      mov ebx, [your_string]
60      mov byte ecx, [eax*1+ebx]
61      cmp ecx, 0
62      je end_my_function
63
64      pushad
65      mov ebx, [your_string]
66      mov byte ecx, [eax*1+ebx]
67      push ecx
68      call print_a_char
69      pop ecx
70      popad
71
72      inc eax
73      jmp my_loop
74  end_my_function:
75      ; jmp $
76      ret
77
```

将 eax 视作 i，循环前从内存读取 string[i]，若 string[i]='\0'则退出循环。

否则跳转到 for 循环的头。

实验结果如下所示：

```
[2]+  已停止               gdb
wenny@owo:~/lab2/assignment$ make run
student.asm:60: warning: register size specification ignored [-w+other]
student.asm:66: warning: register size specification ignored [-w+other]
>>> begin test
>>> if test pass!
>>> while test pass!
Mr.Chen, students and TAs are the best!
```

Assignment 4

```
16    mov dx, 0x0200
17    mov bx, 0x0101
18    ;dh 行 dl 列
19    ;bh 行方向 bl 列方向 1=increase 0=decrease
20    bounce:
21        mov ah, 0
22    ;cursor 1:position
23        mov al, dh
24        imul cx, ax, 80
25        mov al, dl
26        add cx, ax
27        shl cx, 1
28        mov bp, cx
29    ;print a char
30        mov ch, [count]
31        call inc_count  ;increase to change the color
32        mov cl, 'w'
33        mov word [ gs : bp ], cx
```

使用 dx 来保存字符位置的行和列，其中 dh 保存行数，dl 保存列数，bx 保存方向；count 用来作为颜色改变的计数器。

执行过程中会不断重复 bounce 过程，它会先根据字符的位置还有颜色参数等显示字符，再根据字符的方向更新字符的位置，即下一个字符出现的地方，同时更新颜色。

对于颜色的更新定义一个 1 字节的变量 count，每次调用后 count 的值增加 1，将 count 的值作为显存前景色和背景色的设置。

```
34  ;cursor 2:position which is symatrical to the other
35      mov al, dh
36      sub al, 24
37      neg al
38      imul cx, ax, 80
39      mov al, dl
40      sub al, 79
41      neg al
42      add cx, ax
43      shl cx, 1
44      mov bp, cx
45  ;print a char
46      mov ch, [count]
47      call inc_count
48      mov cl, 'w'
49      mov word [ gs : bp ], cx
```

另一个字符的位置和原字符中心对称，行数为（24-dh），列数为（79-dl）。

```
50  ;always display
51      mov ah, 0x05
52      mov al, 'w'
53      mov [gs:2 * 33], ax
54      mov al, 'e'
55      mov [gs:2 * 34], ax
56      mov al, 'n'
57      mov [gs:2 * 35], ax
58      mov al, 'n'
59      mov [gs:2 * 36], ax
60      mov al, 'y'
61      mov [gs:2 * 37], ax
62
63      mov al, 0
64      mov [gs:2 * 38], ax
65      mov al, '1'
66      mov [gs:2 * 39], ax
67      mov al, '9'
68      mov [gs:2 * 40], ax
69      mov al, '3'
70      mov [gs:2 * 41], ax
71      mov al, '3'
72      mov [gs:2 * 42], ax
73      mov al, '5'
74      mov [gs:2 * 43], ax
75      mov al, '0'
76      mov [gs:2 * 44], ax
77      mov al, '7'
78      mov [gs:2 * 45], ax
79      mov al, '4'
80      mov [gs:2 * 46], ax
```

名字和学号是一直会显示的。

```
81   ;change position depending on the direction stored in bx
82   ;if inc
83       add dh, bh
84       add dl, bl
85   ;if dec
86       mov ch, bh
87       cmp ch, 0
88       jne L1          ;if equal, decrease dh by 1, or else just skip it
89       call dec_dh
90   L1:
91       mov ch, bl
92       cmp ch, 0
93       jne L2
94       call dec_dl     ;if equal, decrease dl by 1
95   L2:
96
97       call getDir     ;update the direction
98       call delay
99       jmp bounce      ;loop
100  dec_dh:
101      dec dh
102      ret
103  dec_dl:
104      dec dl
105      ret
```

根据方向更新字符的位置。

```
106  getDir:
107  ;if the cursor meets the up and down side, flip dl
108  ;if the cursor meets the left and right side, flip dh
109      mov ch, dh
110      cmp ch, 0
111      jne L3
112      call flip_bh
113  L3:
114      mov ch, dh
115      cmp ch, 24
116      jne L4
117      call flip_bh
118  L4:
119      mov ch, dl
120      cmp ch, 0
121      jne L5
122      call flip_bl
123  L5:
124      mov ch, dl
125      cmp ch, 79
126      jne L6
127      call flip_bl
128  L6:
129      ret
```

根据字符的位置和方向更新方向。

```
136   delay:
137       pushad
138       mov cx, 1
139       mov dx, 0
140       mov ah, 0x86
141       int 0x15
142       popad
143       ret
```

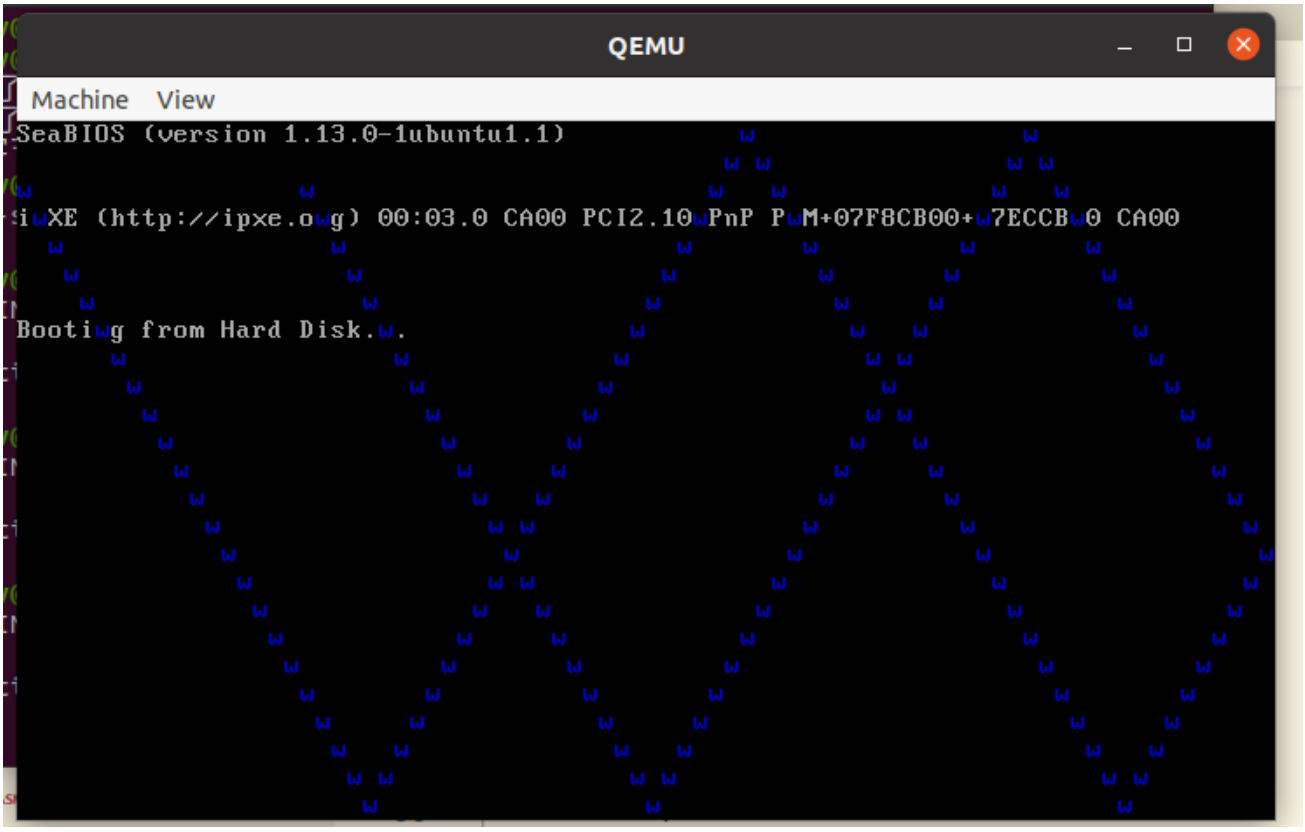
延时的设置。

```
144   inc_count:
145       mov al, [count]
146       inc al
147       mov [count], al
148       ret
```

颜色计数值增加 1。

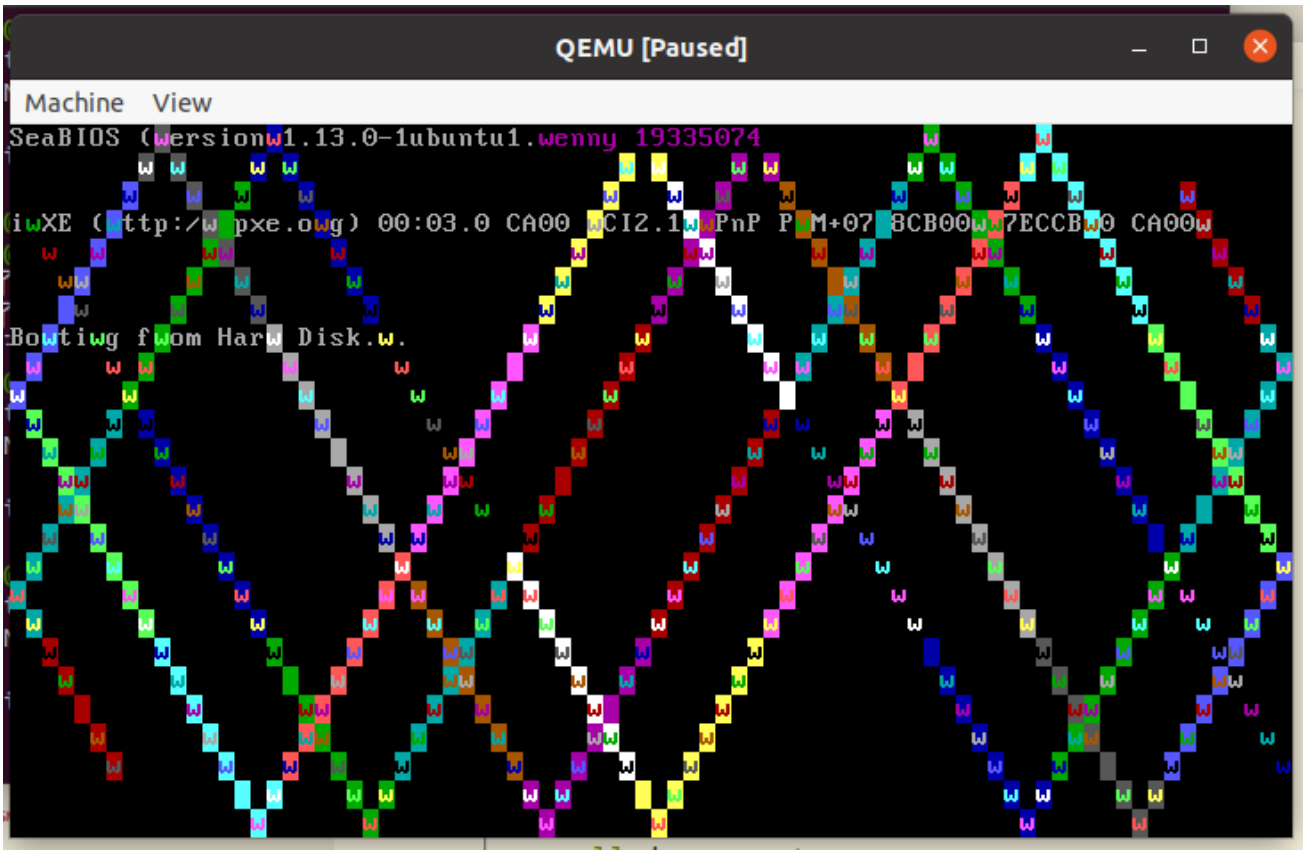
运行结果如下所示：

开始设计程序时本着由简到繁的想法，先实现了单个字符的显示和弹射。



后来想到对称、用 count 改变颜色后。

最后加上名字学号。



4. 总结

通过这次实验，我学习到了 x86 汇编、计算机的启动过程、IA-32 处理器架构和字符显存原理。根据所学的知识，开始能自己编写程序，然后让计算机在启动后加载运行，增进了对计算机启动过程的理解，同时，也学会了使用 gdb 来调试程序的基本方法。

Assignment1：Assignment1 的完成是较为简单的，读懂 TA 写的指导后不需要太多其他的操作，只是重复步骤和计算，但也是在这个过程中明白了计算机启动的大体流程。

Assignment2：一开始对着参考资料一头雾水，不知道究竟该干什么，因为读题不仔细以为要实现的是博客前一部分的内容，这部分花了不少时间，后来才发现要实现的博客最后一段所说的内容，通过网上搜索资料，查阅 BIOS 的中断向量表，以及参考他人调用的实例，最后成功解决了问题。总而言之，感觉像是一个函数调用的过程，实现起来也很方便，但由于分析需求错误花了很多时间，希望下次自己能目标明确一些。

Assignment3：遇到的问题更多是由于对 x86 不熟悉造成的，虽然 MIPS 写了很多，但 x86 的一些规则还是和 MIPS 有所差别。调试过程中出现最多的是段错误，这来自于对 x86 的寻址方式不了解不熟悉，应该在以后的编程中更加注意。其中还要注意的是 cmp 这条指令，它将第一个操作数减去第二个操作数，并根据比较结果设置机器状态寄存器 eflags 中的条件码，因此使用后作为操作数的寄存器的值会有所改变，这是一个值得注意的地方。

Assignment4：一开始看到题目时也是一头雾水，因为对汇编语言还不够掌握，觉得困难的地方是不知道能否仅凭借汇编实现，还是需要别的知识，担心在所储备

的知识不够的情况下就设计算法写代码会浪费很多时间和精力（在其他科很多 DLL 赶着跑的情况下）。尝试去 GitHub 上搜索有关的例子和实现方法但没找到，于是开始自己琢磨，首先想到用一个寄存器来锁存字符位置，一个锁存方向的方法，再想明白方向在什么时候改变，怎么改变方向等问题，再查阅了解了延时中断的使用方法，最后实现和完善程序。

此外还学会了 gdb 的基本使用方法，包括设置断点、显示指令、显示寄存器状态等操作，学会这些方法为这次实验奠定了基础，在以后的实验中也会使用。

总而言之，此次实验使我受益匪浅。