



本科生实验报告

实验课程：_____操作系统_____

实验名称：_____Lab3_____

专业名称：_____信息与计算机科学_____

学生姓名：_____张文沁_____

学生学号：_____20337268_____

实验地点：_____无固定地点_____

实验成绩：_____

报告时间：_____2022. 3. 12_____

1. 实验要求

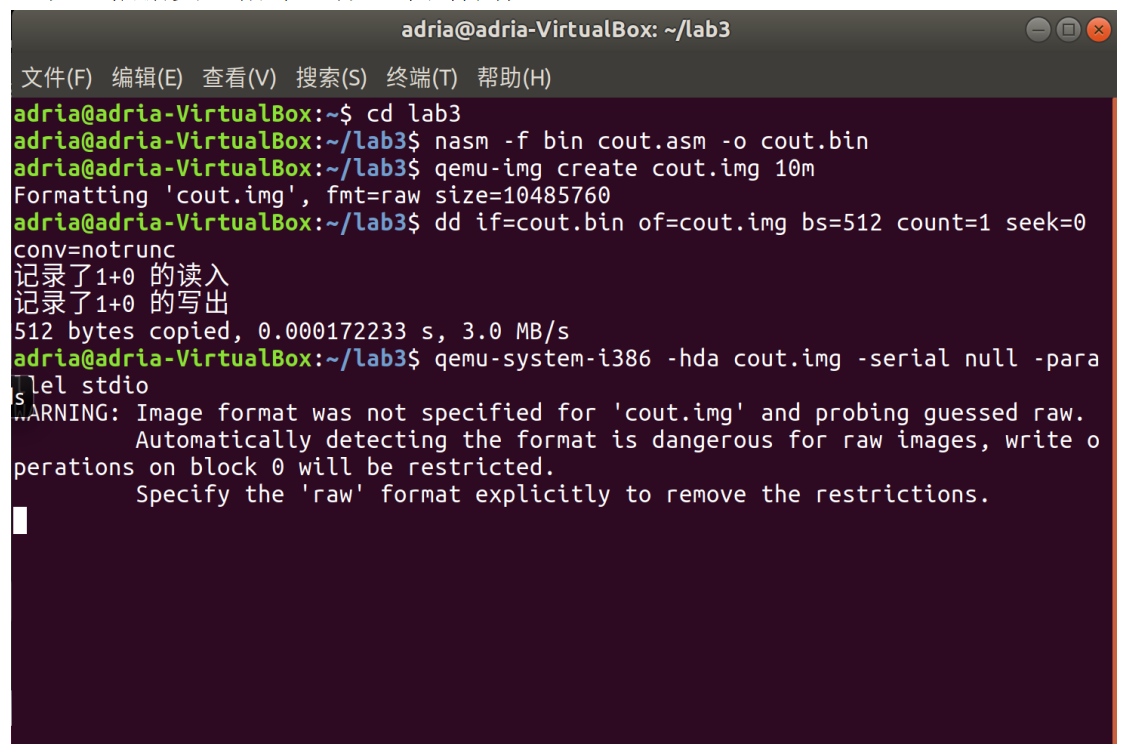
- a) 实验不限语言， C/C++/Rust 都可以。
- b) 实验不限平台， Windows、Linux 和 MacOS 等都可以。
- c) 实验不限 CPU， ARM/Intel/Risc-V 都可以。

2. 实验过程

a) Assignment 1 MBR

- i. 1.1: 复现 example 1。说说你是怎么做的，并将结果截图。

过程：根据实验指导进行一系列操作



```
adria@adria-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
adria@adria-VirtualBox:~$ cd lab3
adria@adria-VirtualBox:~/lab3$ nasm -f bin cout.asm -o cout.bin
adria@adria-VirtualBox:~/lab3$ qemu-img create cout.img 10m
Formatting 'cout.img', fmt=raw size=10485760
adria@adria-VirtualBox:~/lab3$ dd if=cout.bin of=cout.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, 0.000172233 s, 3.0 MB/s
adria@adria-VirtualBox:~/lab3$ qemu-system-i386 -hda cout.img -serial null -parallel stdio
WARNING: Image format was not specified for 'cout.img' and probing guessed 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.
```

- ii. 1.2: 请修改 example 1 的代码，使得 MBR 被加载到 0x7C00

后在(12,12)(12,12)处开始输出你的学号。注意，你的学号显示的前景色和背景色必须和教程中不同。

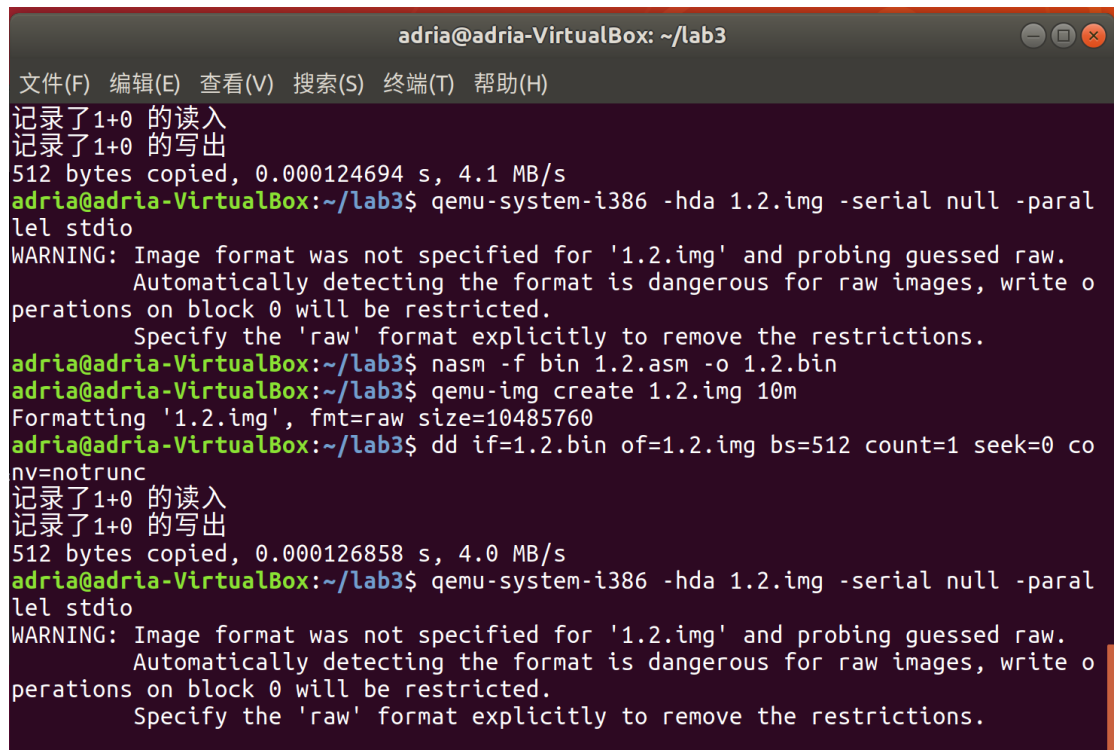
过程：根据实验指导进行一系列操作并进行要求下的修改。根据

显示位置公式计算初始位置。并改变显示颜色 (0x07 白色)。由 qemu

显示屏是按 25x80 个字符来排列的矩阵可知 (12,12) 显存起始位置

$$=0xB8000+2*(80*12+12)=0xB8000+2 \cdot 972$$

用到命令如下:



```
adria@adria-VirtualBox: ~/lab3
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
记录了1+0 的读入
记录了1+0 的写出
512 bytes copied, 0.000124694 s, 4.1 MB/s
adria@adria-VirtualBox:~/lab3$ qemu-system-i386 -hda 1.2.img -serial null -parallel stdio
WARNING: Image format was not specified for '1.2.img' and probing guessed 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.
adria@adria-VirtualBox:~/lab3$ nasm -f bin 1.2.asm -o 1.2.bin
adria@adria-VirtualBox:~/lab3$ qemu-img create 1.2.img 10m
Formatting '1.2.img', fmt=raw size=10485760
adria@adria-VirtualBox:~/lab3$ dd if=1.2.bin of=1.2.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, 0.000126858 s, 4.0 MB/s
adria@adria-VirtualBox:~/lab3$ qemu-system-i386 -hda 1.2.img -serial null -parallel stdio
WARNING: Image format was not specified for '1.2.img' and probing guessed 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.
```

b) Assignment 2 实模式中断

- i. 2.1: 请探索实模式下的光标中断, 利用中断实现光标的位置获取和光标的移动。

通过输出字符串的方式证明进行了光标的获取和移动, 获取体现在可以输出到指定位置, 移动体现在输出之后光标在字符串最后面表示光标随着字符串的输出进行了移位。

- ii. 2.2: 请修改 1.2 的代码, 使用实模式下的中断来输出你的学号。

实现方式类似实验 2.1, 因为都是中断进行的输出, 所以在 2.1 的基

础上进行了修改，使之输出学号，使用了 int 10h 中断的 13H 功能

- iii. 2.3: 在 2.1 和 2.2 的知识的基础上，探索实模式的键盘中断，利用键盘中断实现键盘输入并回显。

使用了 int 16h 中断的 0 号功能。它会将键盘输入的字符的 ASCII 码放在寄存器 al 中。

c) Assignment 3 函数的实现

- i. 3.1: 分支逻辑的实现
- ii. 3.2: 循环逻辑的实现
- iii. 3.3: 请编写函数 `your_function` 并调用之，函数的内容是遍历字符数组 `string`

学习教程中所给关于汇编语言的内容完成了实验。debug 过程遇到困难在最后阐述。

d) Assignment 4 汇编小程序

先要写一个函数来延时，控制画框显示速度和抖动速度；接下来写函数判断字符该往哪个方向走并打印字符。具体实现见代码区。

4. 关键代码

a) Assignment 1 MBR

- i. 1.1: 复现 example 1。说说你是怎么做的，并将结果截图。

```
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.   ; 初始化栈指针
11.   mov sp, 0x7c00
12.   mov ax, 0xb800
13.   mov gs, ax
14.   ;输出
15.   mov ah, 0x01 ;蓝色
16.   mov al, 'H'
17.   mov [gs:2 * 0], ax
18.   ...
19.   jmp $ ; 死循环
20.   times 510 - ($ - $$) db 0
21.   db 0x55, 0xaa

```

- ii. 1.2: 请修改 example 1 的代码, 使得 MBR 被加载到 0x7C00 后在(12,12)处开始输出你的学号。注意, 你的学号显示的前景色和背景色必须和教程中不同。说说你是怎么做的, 并将结果截图。

答: 和上文实验一致, 只是修改了

`mov ax,0xb8ac` (起始位置, 但是似乎并不精确, 第二次直接在代码中加入了 `mov[gs:2*972],ax` 的字样)

`mov al,0x07` (显示颜色)

b) Assignment 2 实模式中断

- i. 2.1: 请探索实模式下的光标中断, 利用中断实现光标的位置获取和光标的移动。

通过输出字符串的方式证明进行了光标的获取和移动, 获取体现在可以输出, 移动体现在输出之后光标在字符串最后面表示光标随着字符串的输出进行了移位。

```

1.    SECTION MBR vstart=0x7c00
2.    ;将起始地址设置为 0x7c00—因为 BIOS 会将 MBR 程序加载到 0x7c00 处
3.
4.        mov     sp, 0x7c00
5.    ;清空屏幕，使用 BIOS 提供的中断
6.        mov     ax, 0x600
7.        mov     bx, 0x700
8.        mov     cx, 0x0
9.        mov     dx, 0x184f
10.       int     0x10
11.    ;下面获取当前的光标位置
12.    ;    INT 0x10;    功能号:0x03    功能描述:获取当前光标位置
13.    ;    输入:
14.    ;        AH--功能号:    0x03
15.    ;        BH--带获取光标的页码号
16.    ;    输出:
17.    ;        CH--光标开始行
18.    ;        CL--光标结束行
19.    ;        DH--光标所在行号
20.    ;        DL--光标所在列号
21.
22.    ;调用功能号为 0x3 的 BIOS 中断，获取当前光标位置的相关信息，并将相关信息保
    存在对应的寄存器中
23.
24.        mov     ah, 0x03
25.        mov     bx, 0
26.        int     0x10
27.
28.    ;    下面进行打印字符串
29.    ;    INT 0x10;    功能号:0x13    功能描述:打印出字符串
30.    ;    输入:
31.    ;        ES:BP--字符串地址
32.    ;        AH--功能号    0x13
33.    ;        AL--设置写字符串方式 1 表示光标跟随移动
34.    ;        CX--字符串长度(不包括最后的 0)
35.    ;        BH--设置要显示的页号
36.    ;        BL--设置字符属性 0x2 表示黑底绿字
37.
38.        mov     ax, cs
39.        mov     es, ax
40.        mov     ax, String
41.        mov     bp, ax
42.        mov     ax, 0x1301
43.        mov     bx, 0x2

```

```

44.      mov     cx, 0x12
45.      int     0x10
46.      ;调用 0x13 的 BIOS 中断，将 0:String 地址处，长度为 0x12（后期随长度进行修改）的字符串进行了输出，并且光标跟随移动
47.      jmp     $
48.      String db "Cursor behind me."
49.      times 510 - ($ - $$) db 0
50.      db      0x55, 0xaa

```

- ii. 2.2: 请修改 1.2 的代码，使用实模式下的中断来输出你的学号。

实现方式类似实验 2.1，因为都是中断进行的输出，所以在 2.1 的基础上进行了修改，使之输出学号

- iii. 2.3: 在 2.1 和 2.2 的知识的基础上，探索实模式的键盘中断，利用键盘中断实现键盘输入并回显

使用了 int 16h 中断的 0 号功能。它会将键盘输入的字符的 ASCII 码放在寄存器 al 中，设置寄存器 ah 的颜色，将 ax 复制给 0xB8000，就会在 qemu 屏幕的第一个位置显示输入的字符。结果如下：输入字符 a，qemu 屏幕上显示 a

```

1.      mov     gs,ax
2.      mov     sp,0x7c00
3.      mov     ax,0xb800
4.      mov     gs,ax
5.
6.      start:
7.      ;清屏
8.      mov     ah,6
9.      mov     al,0
10.     mov     ch,0
11.     mov     cl,0
12.     mov     dh,25
13.     mov     dl,80
14.     int     10h
15.     mov     ah,0;调用 16 号中断的 0 号功能
16.     int     16h
17.     mov     ah,0xcf
18.     mov     [gs:2*0],ax
19.     ;死循环

```

```
20.    jmp $
21.    times 510 - ($ - $$) db 0
22.    db 0x55,0xaa
```

c) Assignment 3 函数的实现

i. 3.1: 分支逻辑的实现

```
1.    %include "head.include"
2.
1.    your_if:
2.    mov edx,[a1]
3.    cmp edx,12
4.    jl case1
5.    cmp edx,24
6.    jl case2
7.    jl case3
8.
9.    case1:
10.   mov eax,[a1]
11.   mov ebx,2
12.   idiv ebx
13.   inc eax
14.   mov [if_flag],eax
15.   jmp end
16.
17.   case2:
18.   mov eax,24
19.   sub eax,[a1]
20.   imul eax,edx
21.   mov [if_flag],eax
22.   jmp end
23.
24.   case3:
25.   mov eax,[a1]
26.   shl eax,4
27.   mov [if_flag],eax
28.   jmp end
29.
30.   end:
```

ii. 3.2: 循环逻辑的实现

```
1.    your_while:
2.    loop1:
```



```

3.    mov edx,[a2]
4.    cmp edx,12
5.    jl end_loop
6.    call my_random
7.    mov ebx,[while_flag]
8.    mov edx,[a2]
9.    mov byte[ebx+(edx-12)],a1
10.   dec edx
11.   mov [a2],edx
12.   jmp loop1
13.
14.   end_loop:

```

iii. 3.3: 请编写函数 `your_function` 并调用之, 函数的内容是遍

历字符数组 `string`

```

1.    your_function:
2.    xor eax,eax ;清零操作
3.    xor ecx,ecx
4.    mov ebx,[your_string]
5.    loop2:
6.    mov cl,byte[eax+ebx] ;注意是字节寄存器
7.    inc eax
8.    cmp ecx,0
9.    je end_for
10.   pushad
11.   push ecx
12.   call print_a_char
13.   pop ecx
14.   popad
15.   jmp loop2
16.
17.   end_for:
18.   ret

```

d) Assignment 4 汇编小程序

i. 延时程序

```

1.    Dn_Rt equ 1
2.    Up_Rt equ 2
3.    Up_Lt equ 3
4.    Dn_Lt equ 4

```

```

5.      delay equ 1000
6.      ddelay equ 100      ; 计时器延迟计数,用于控制画框的速度

```

ii. 判断走向

```

1.      mov al,1
2.      cmp al,byte[rdu1]
3.      jz  DnRt  ;跳转到对应的区块进行处理
4.      mov al,2
5.      cmp al,byte[rdu1]
6.      jz  UpRt
7.      mov al,3
8.      cmp al,byte[rdu1]
9.      jz  UpLt
10.     mov al,4
11.     cmp al,byte[rdu1]
12.     jz  DnLt

```

iii. 打印字符

```

1.      show:
2.      xor ax,ax
3.      ; 计算显存地址
4.      mov ax,word[x]
5.      mov bx,80
6.      mul bx
7.      add ax,word[y]
8.      mov bx,2
9.      mul bx
10.     mov bp,ax
11.     mov ah,[curcolor2]
12.     ; 弹字符的背景色和前景色
13.     inc byte[curcolor2]
14.     cmp byte[curcolor2], 0fh
15.     jnz skip
16.     skip:
17.     mov al,byte[char]      ; AL = 显示字符值(默认值为 20h=空格符)
18.     mov word[gs:bp],ax    ; 显示字符的 ASCII 码值
19.
20.     mov si, myinfo
21.     mov di, 2
22.     mov cx, word[infoLen]
23.     loop2:                ; 显示 myinfo 中的每个字符
24.     mov al, byte[ds:si]
25.     inc si
26.     mov ah, [curcolor]    ; 背景色和前景色

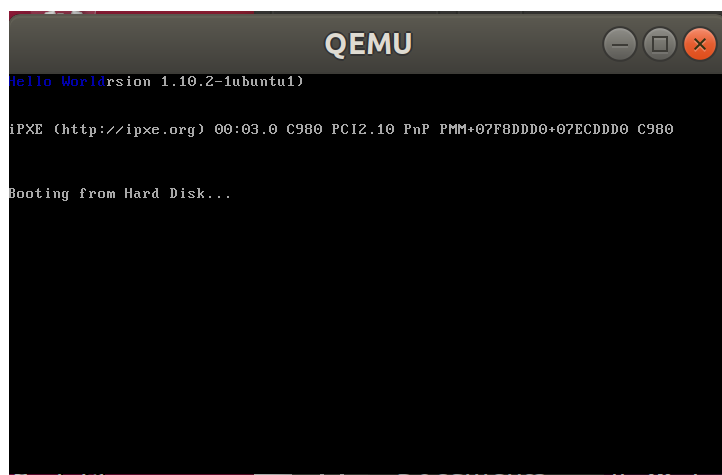
```

```
27.      add byte[curcolor], 12h ; 变色, 数字可随意调节
28.      mov word [gs:di], ax
29.      add di, 2
30.      loop loop2
31.      jmp loop1
```

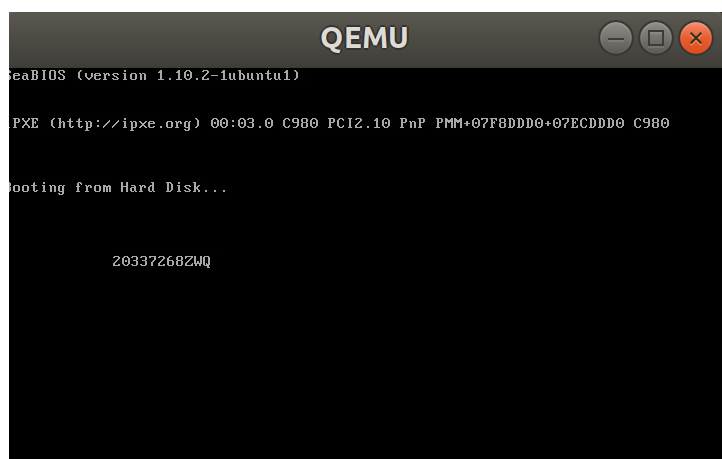
5. 实验结果

a) Assignment 1 MBR

- i. 1.1: 复现 example 1。说说你是怎么做的，并将结果截图。



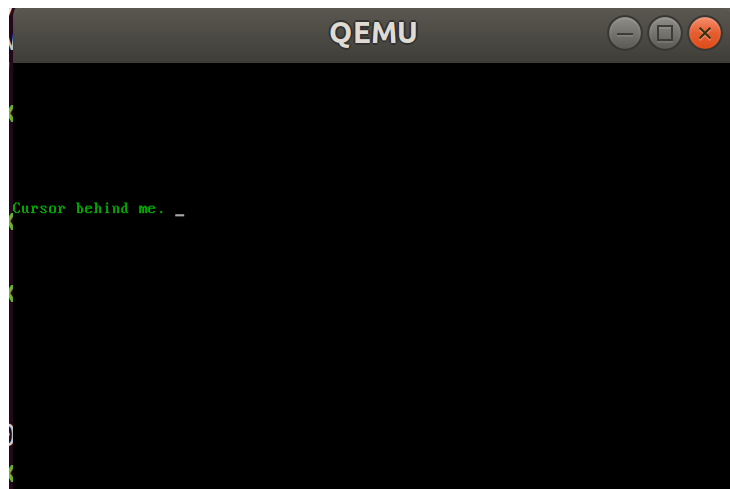
- ii. 1.2: 请修改 example 1 的代码，使得 MBR 被加载到 0x7C00 后在(12,12)处开始输出你的学号。注意，你的学号显示的前景色和背景色必须和教程中不同。说说你是怎么做的，并将结果截图。



b) Assignment 2 实模式中断

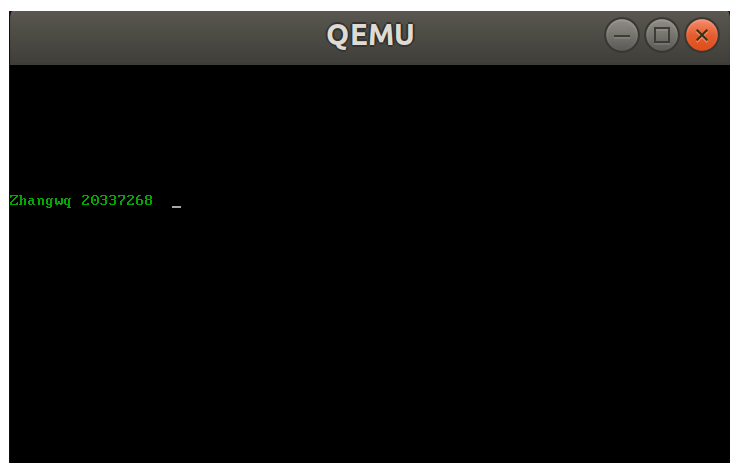
- i. 2.1: 请探索实模式下的光标中断，利用中断实现光标的位置获取和光标的移动。说说你是怎么做的，并将结果截图。

解释：通过输出字符串的方式证明进行了光标的获取和移动，获取体现在可以输出，移动体现在输出之后光标在字符串最后面表示光标随着字符串的输出进行了移位。



- ii. 2.2: 请修改 1.2 的代码，使用实模式下的中断来输出你的学号。说说你是怎么做的，并将结果截图。

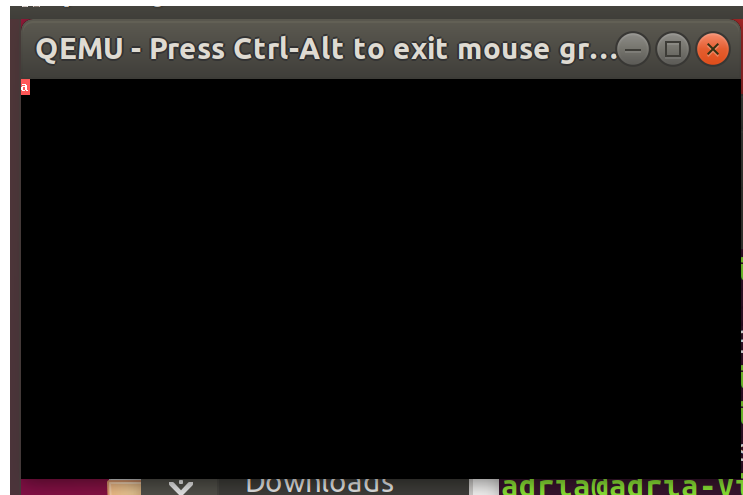
解释：因为原理相同，所以代码同上。



- iii. 2.3: 在 2.1 和 2.2 的知识的基础上，探索实模式的键盘中

断，利用键盘中断实现键盘输入并回显

结果如下：



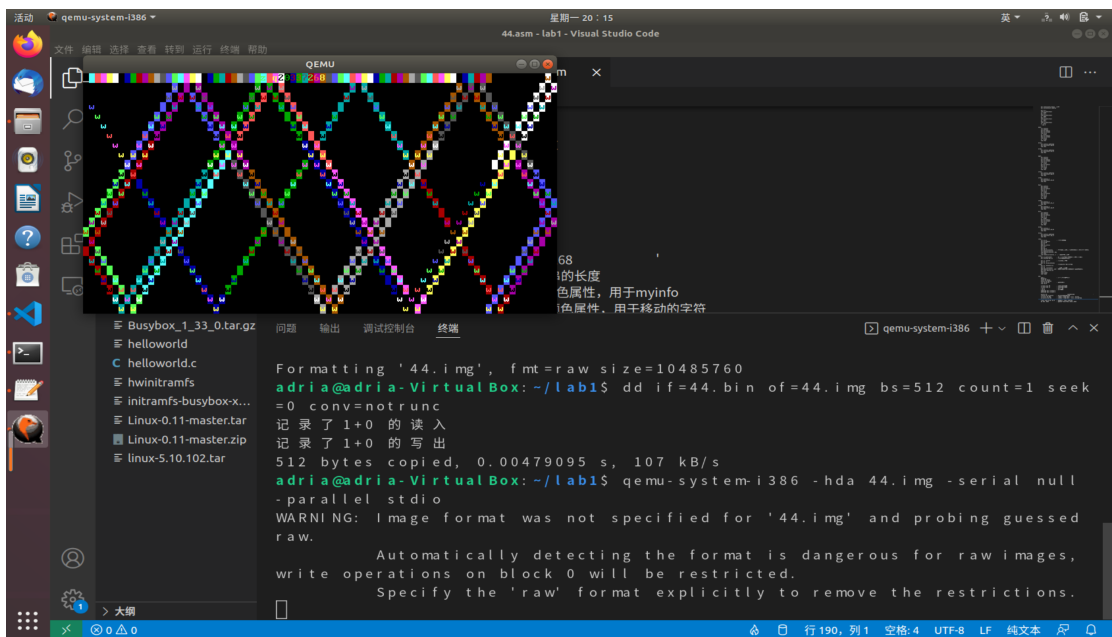
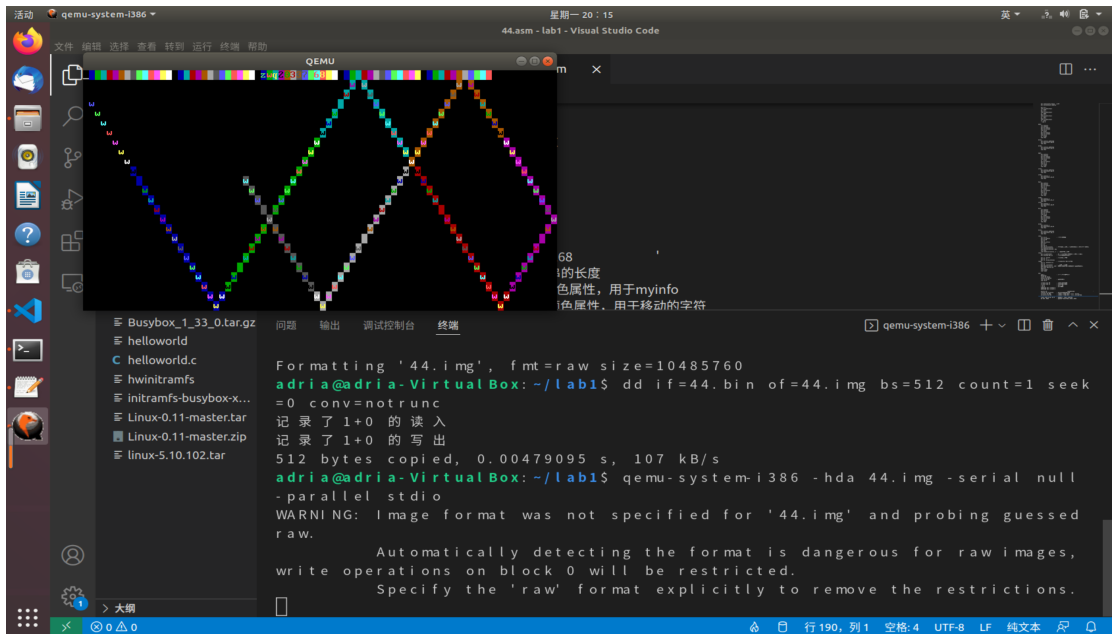
c) Assignment 3 函数的实现

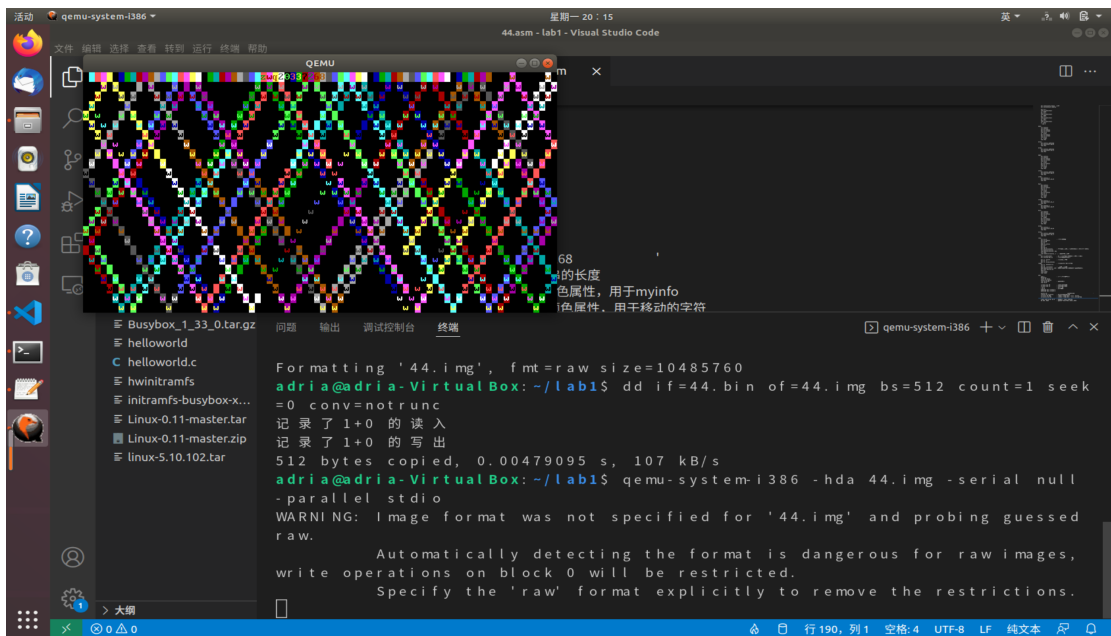
- i. 3.1: 分支逻辑的实现
- ii. 3.2: 循环逻辑的实现
- iii. 3.3: 请编写函数 `your_function` 并调用之，函数的内容是遍历字符串 `string`

在终端进行运行会报错，但是放在 vs code 下面又可以正常运行

```
>>> begin test
>>> if test pass!
>>> while test pass!
Mr.Chen, students and TAs are the best!
```

d) Assignment 4 汇编小程序





6. 总结

- nasm 汇编语言使用技能还需要提高
- 2.3 中只能实现一个字符的输入和输出，无法完成字符串的功能
- `mov eax,[a1]` ;a1 不能直接使用，而是要经过间接寻址才可以访问
- 使用了 `quit` 函数（自定义函数）导致只有一个输出

```

adria@adria-VirtualBox:~$ cd lab3
adria@adria-VirtualBox:~/lab3$ cd assignment
adria@adria-VirtualBox:~/lab3/assignment$ make run
>>> begin test
0
adria@adria-VirtualBox:~/lab3/assignment$

```

- 分辨率怎么调试，在强制关闭虚拟机之后分辨率变差且无法恢复原样，附带使得终端无法执行命令，只得用 vs code 完成任务 4
- 关于任务：1.2 按照计算的值来进行输出会出格子，取近似

mov ax,0xb8ac 或者 0xb8ab 0xb8aa 都相去甚远, 最后查询得知
可以使用 `mov[gs:2*972],ax`