

操作系统原理实验报告

**实验名称:** 实验二实模式和保护模式下的OS启动

**授课教师：** 张青

**学生姓名:** 曾慧蕾

**学生学号:** 21307358

1. **实验要求**

**Assignment 1 MBR：**

注意，assignment 1的寄存器请使用16位的寄存器。

内容

1.1 复现Example 1。说说你是怎么做的，并将结果截图。

1.2 修改Example 1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。

**Assignment 2 实模式中断**

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。

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。

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，可以参考

https://blog.csdn.net/deniece1/article/details/103447413。关于键盘扫描码，可以参考

http://blog.sina.com.cn/s/blog\_1511e79950102x2b0.html。说说你是怎么做的，并将结果截图。

**Assignment 3 汇编**

**assignment 3的寄存器请使用32位的寄存器。**

首先执行命令sudo apt install gcc-multilib g++-multilib安装相应环境。

你需要实现的代码文件在assignment/student.asm中。

编写好代码之后，在目录assignment下使用命令make run即可测试，不需要放到mbr中使用qemu启动。

a1、if\_flag、my\_random等都是预先定义好的变量和函数，直接使用即可。

你可以修改test.cpp中的student\_setting中的语句来得到你想要的a1,a2。

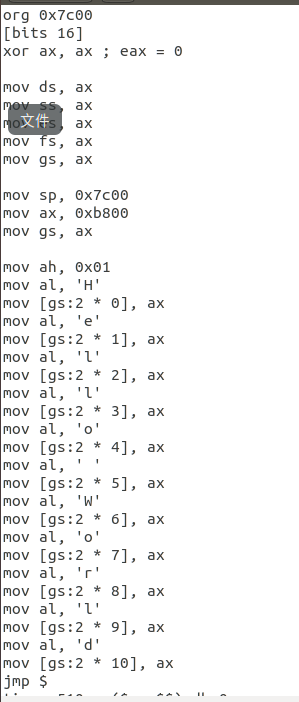
最后附上make run的截图，并说说你是怎么做的

**Assignment 4 汇编小程序**

字符弹射程序。请编写一个字符弹射程序，其从点(2,0)处开始向右下角45度开始射出，遇到边界反弹，反弹后按45

度角射出，方向视反弹位置而定。同时，你可以加入一些其他效果，如变色，双向射出等。注意，你的程序应该不超过510字节，否则无法放入MBR中被加载执行

1. **实验过程**

1.1：根据实验指示流程，向al中依次放入需要输出的字符，并输出即可，该实验要注意的就是输出地址需要实验者自行自增，当然也可直接定义一个常量后循环输出，不过因为做这个实验的时候还不会后面的循环因此还是选择了第一种方法

第1、2行的org 0x7c00和[bits 16]是汇编伪指令，不是实际的指令。org 0x7c00是告诉编译器代码中的代码标号和数据标号从0x7c00开始。也就是说，这些

标号的地址会在编译时加上0x7c00。如果没有这一句，标号的值就默认是标号从代码开始处的偏移地址。此时，如果我们引用标号就会出错。[bits 16]告

诉编译器按16位代码格式编译代码。

对于ax的赋值，这里仅赋为0xb800是因为根据物理地址计算方式，段地址左移四位之后再加偏移地址，因此不再需要最后的一个0.

在mbr文件编写完毕后，向终端输入如下指令：

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

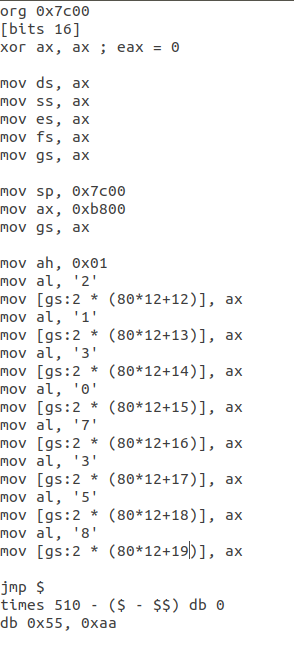
qemu-img create hd.img 10m

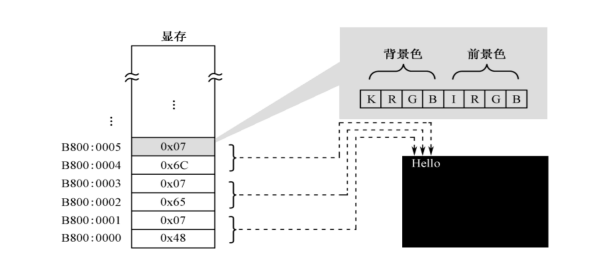
dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc

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

即可在qemu中观察到运行结果

1.2 实验流程与1.1基本一致，主要是学号从（12，12）开始需要计算

 想要更改字体的前景色和背景色，参考：

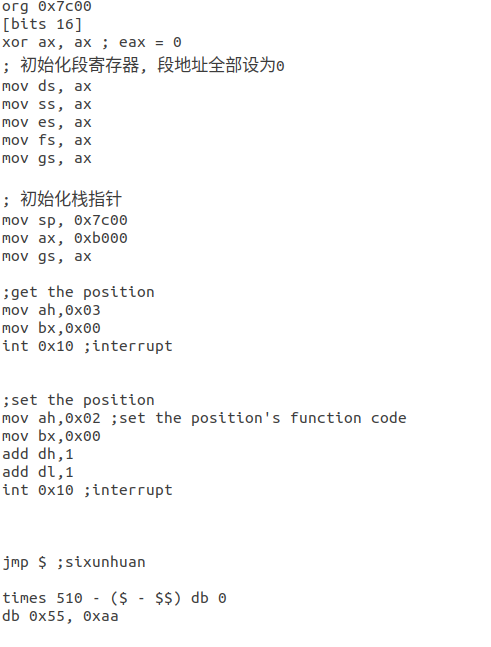


如图所示，而我在这里将0x01改为了0x32

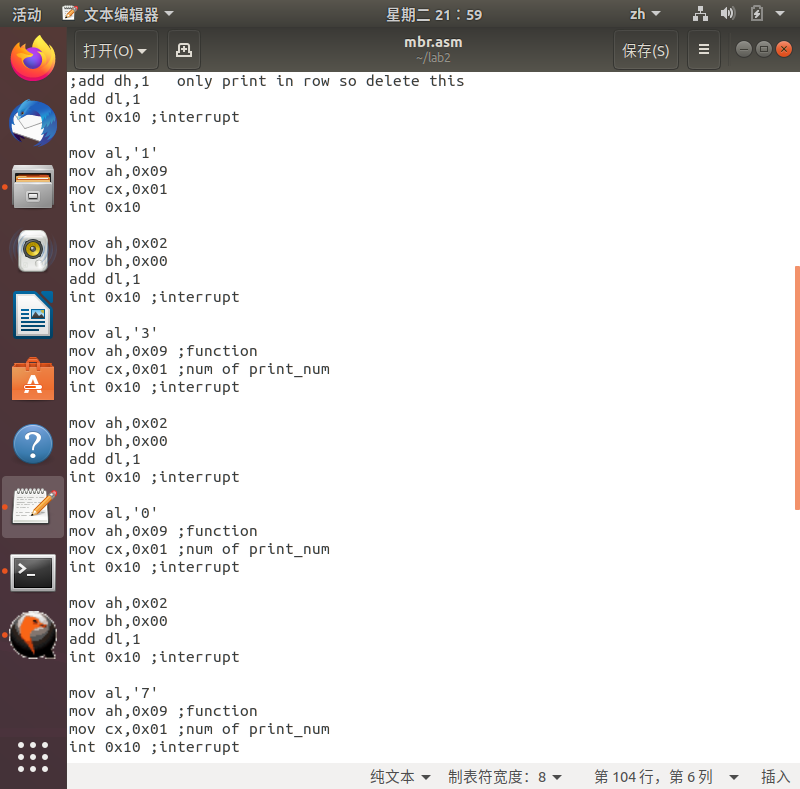
* 1. 在网上查询之后，发现本任务需要的中断号为INT 10

在初始化段寄存器和栈指针后，补全功能号及其对应的参数，写入寄存器中，再中断即可

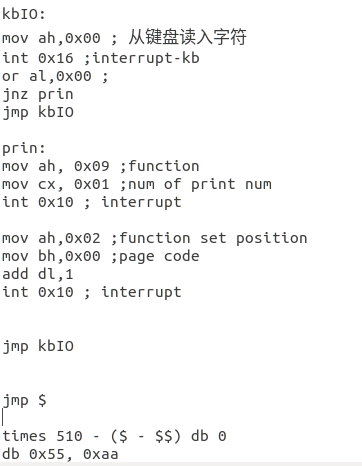
在这里我原先是先补全02H再补全03H，但是发现这样两次qemu显示的光标位置不同，最后采取了先获取光标位置再设置光标位置的顺序



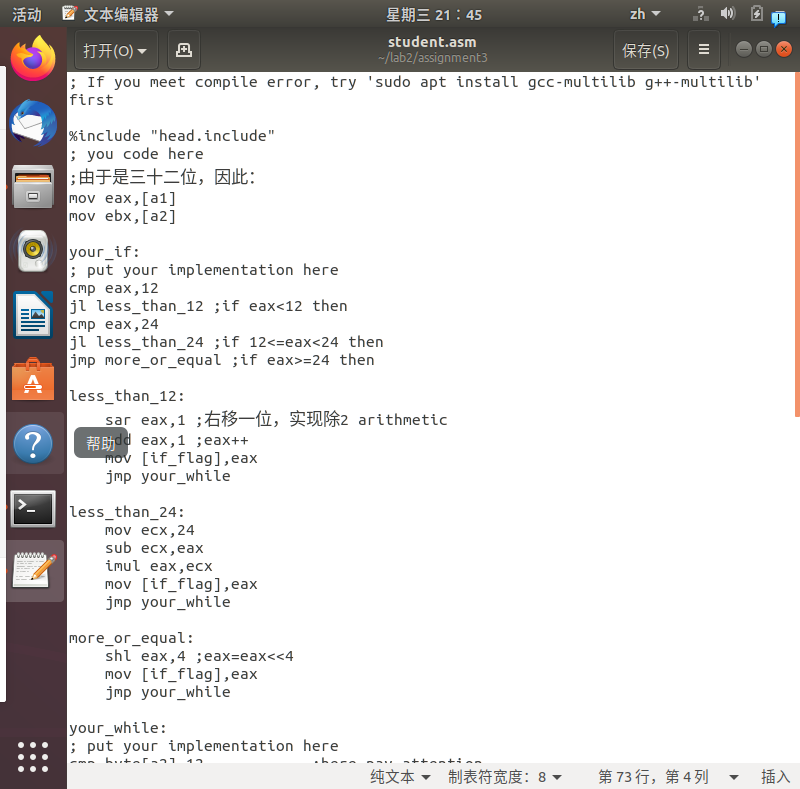
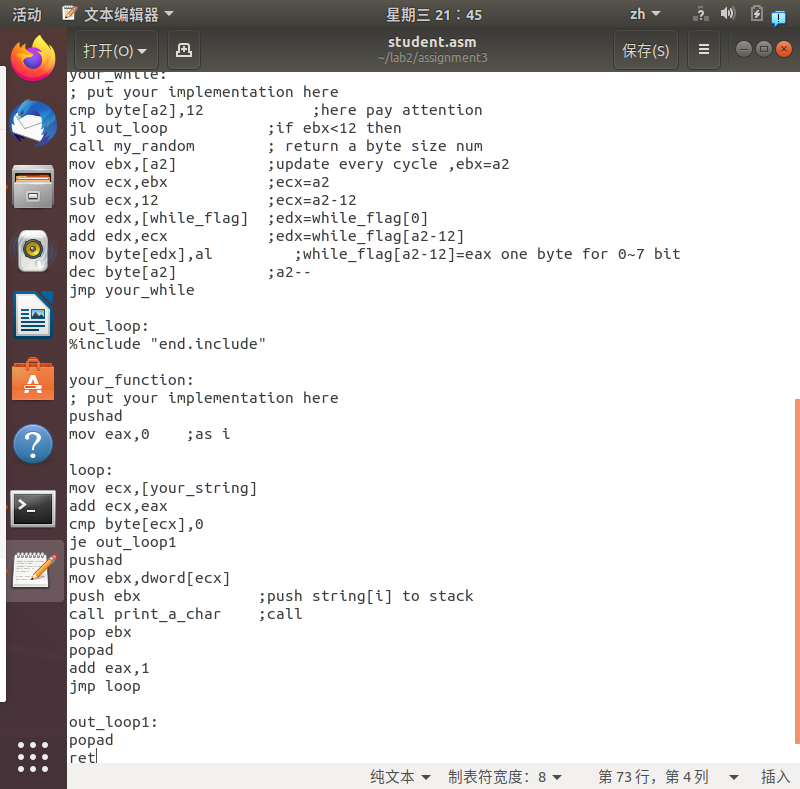
* 1. 思路同1.2，每输出一个字符便将光标位置设置往后一位，挨个输出，直到输完为止，在调用03H后以09H（写字符）-02H（设置位置）的顺序循环进行



* 1. 思路同2.2 弄个循环输出就行，这里的主要难点是如何读入键盘的输入 上网查资料后发现是INT 0x16可调用键盘I/0中断 思路如下：循环判断键盘是否有输入，若是，则跳到回显函数中，回显函数请参考2.2



3感觉这个任务的难点主要是考察汇编基础，我在各种寄存器的选取与使用、偏移地址内存中的变量和变量地址的选择都卡了很久

4.。。。。这个任务真的是，其实各种碰撞的条件判断都还好说，初始化几个变量之后每次移动都向方向寄存器里面存入状态，发生碰撞以后就改变状态再塞进寄存器里然后挨个输出就好了，字符内容的变化也好说，设置好循环的判断条件就行。就是各种效果我实在是做的没有办法了，变色啊延时啊什么的。。真的没有办法了最后上网找了位大佬的代码借鉴了

原本还想弄双射效果的，结果用我的思路的话内存超了，没有办法遂放弃

1. **关键代码**

**这里仅放出第四个任务的代码，别的都已在前面放过**

**org 7c00h**

**[bits 16]**

**;一共四种移动方式:左上\左下\右上\右下**

**;赋值**

**\_DR equ 1 ;右下**

**\_UR equ 2 ;右上**

**\_UL equ 3 ;左下**

**\_DL equ 4 ;左上**

**delay equ 300**

**ddelay equ 200**

**START:**

**mov ax, cs**

**mov ds, ax**

**mov ax, 0b800h ;mbr从这里开始**

**mov es, ax**

**mov si, 0**

**mov di, 0**

**PRINT:**

**mov bx, outname**

**mov al, [bx+si]**

**cmp al, 0**

**jz LOOP1 ;到了结尾则跳出**

**mov byte[es:60+di], al ;print char**

**mov ah, 0x32**

**mov byte[es:60+di+1], ah ;set color**

**add si, 1**

**add di, 2 ;由于地址是2\*i所以这里直接加2**

**jmp PRINT**

**LOOP1:**

**dec word[count]**

**jnz LOOP1**

**mov word[count], delay**

**dec word[dcount]**

**jnz LOOP1**

**mov word[count], delay**

**mov word[dcount], ddelay**

**mov al,1**

**cmp al, byte[rdul]**

**je DnRt**

**mov al, 2**

**cmp al, byte[rdul]**

**je UpRt**

**mov al, 3**

**cmp al, byte[rdul]**

**je UpLt**

**mov al, 4**

**cmp al, byte[rdul]**

**je DnLt**

**jmp $**

**;down-right**

**DnRt:**

**inc word[x] ;x++**

**inc word[y] ;y++**

**mov bx, word[x]**

**mov ax, 25**

**cmp ax, bx**

**je dr\_to\_ur ;到达底部**

**mov bx, word[y]**

**mov ax, 80**

**cmp ax, bx**

**je dr\_to\_dl ;到达右侧**

**jmp show**

**dr\_to\_ur:**

**mov word[x], 23 ;反弹后 行数减一**

**mov byte[rdul], \_UR**

**jmp show**

**dr\_to\_dl:**

**mov word[y], 78 ;反弹后 列数减一**

**mov byte[rdul], \_DL**

**jmp show**

**;up-right**

**UpRt:**

**dec word[x] ;x--**

**inc word[y] ;y++**

**mov bx, word[y]**

**mov ax, 80**

**cmp ax, bx**

**je ur\_to\_ul**

**mov bx, word[x]**

**mov ax, 0**

**cmp ax, bx**

**je ur\_to\_dr**

**jmp show**

**ur\_to\_ul:**

**mov word[y], 78**

**mov byte[rdul], \_UL**

**jmp show**

**ur\_to\_dr:**

**mov word[x], 2**

**mov byte[rdul], \_DR**

**jmp show**

**;up-left**

**UpLt:**

**dec word[x] ;x--**

**dec word[y] ;y--**

**mov bx, word[x]**

**mov ax, 0**

**cmp ax, bx**

**je ul\_to\_dl**

**mov bx,word[y]**

**mov ax, -1**

**cmp ax, bx**

**je ul\_to\_ur**

**jmp show**

**ul\_to\_dl:**

**mov word[x], 2**

**mov byte[rdul], \_DL**

**jmp show**

**ul\_to\_ur:**

**mov word[y], 1**

**mov byte[rdul], \_UR**

**jmp show**

**;down-left**

**DnLt:**

**inc word[x] ;x++**

**dec word[y] ;y--**

**mov bx, word[y]**

**mov ax, -1**

**cmp ax, bx**

**je dl\_to\_dr**

**mov bx, word[x]**

**mov ax, 25**

**cmp ax, bx**

**je dl\_to\_ul**

**jmp show**

**dl\_to\_dr:**

**mov word[y], 1**

**mov byte[rdul], \_DR**

**jmp show**

**dl\_to\_ul:**

**mov word[x], 23**

**mov byte[rdul], \_UL**

**jmp show**

**show:**

**xor ax, ax ;eax=0**

**mov ax, word[x]**

**mov bx, 80**

**mul bx ;80\*x**

**add ax, word[y] ;80\*x+y**

**mov bx, 2**

**mul bx ;2\*(80\*x+y)**

**mov bx, ax**

**mov ah, byte[color]**

**mov al, byte[char]**

**mov [es:bx], ax**

**add byte[char],2**

**cmp byte[char],'9'**

**jg reset ;jump when greater than 9**

**jmp keep**

**reset:**

**mov byte[char],'1'**

**cmp byte[char], 'z'+1**

**jnz keep**

**mov byte[char], '0'**

**keep:**

**inc byte[color]**

**cmp byte[color], 0x10**

**jnz LOOP1**

**mov byte[color], 0x40 ; 循环显示不同样式的字符**

**jmp LOOP1**

**end:**

**jmp $**

**count dw delay**

**dcount dw ddelay**

**rdul db \_DR**

**color db 0x01**

**x dw 2 ;start from(2,0)**

**y dw 0**

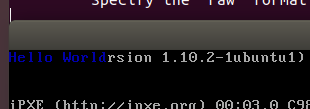
**char db '1' ;初始字符**

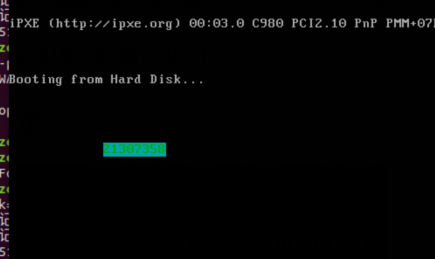
**outname db 'Zenghlei 21307358', 0**

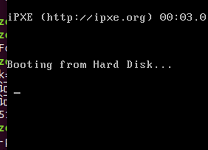
**times 510-($-$$) db 0**

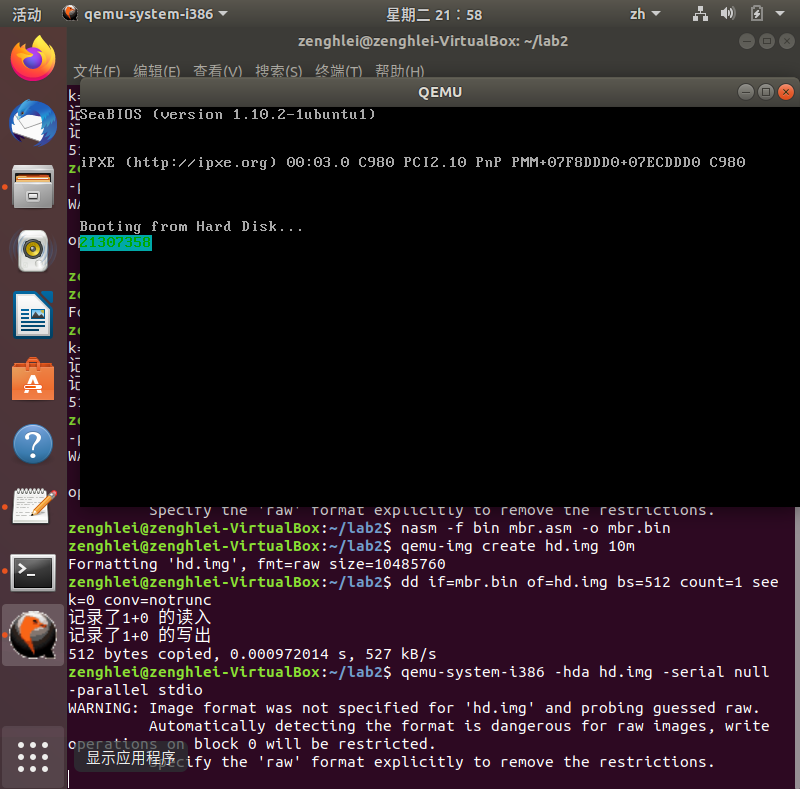
**dw 0aa55h**

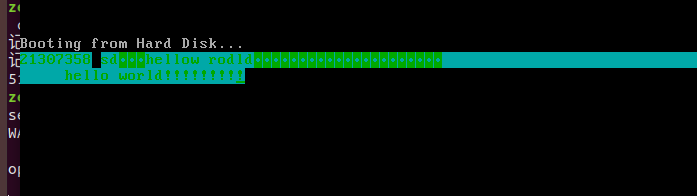
1. **实验结果**

1.1开头输入

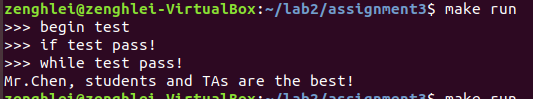
1.2 在（12，12）输出

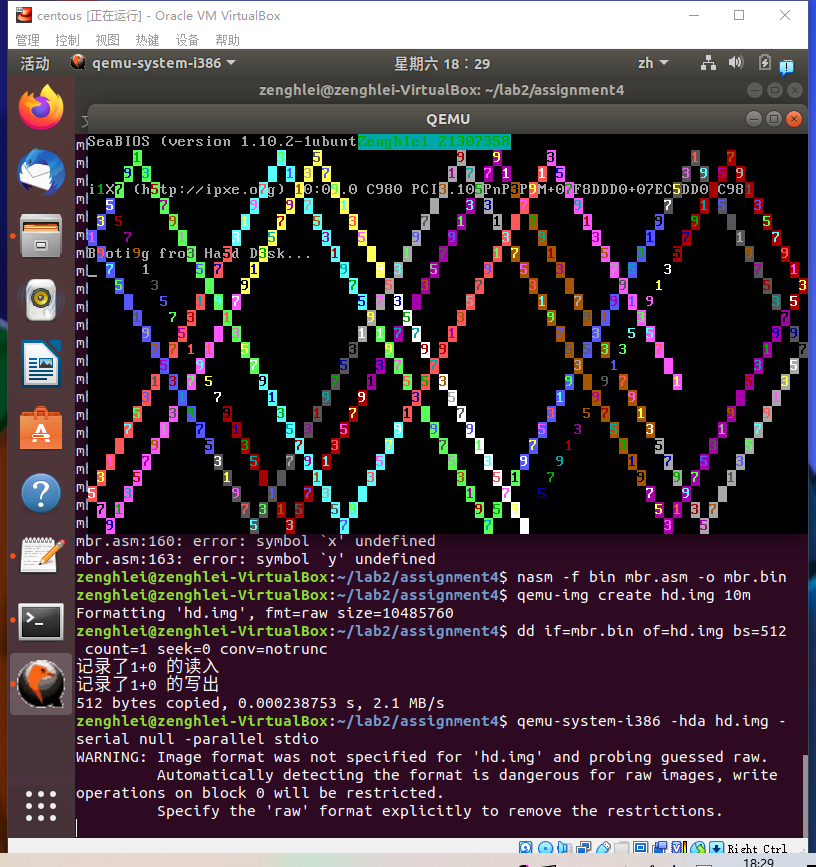
2.1 可见光标确实从（0，0）变为（1，1）

2.2 中断输出学号



2.3键盘回显

3. 

4. 

1. **总结**

进一步了解了汇编知识，熟悉了实模式和保护模式下的OS启动，编写MBR等基本知识。