

一、实验要求：

- DDL: 2021.03.23 23:59
- 提交的内容：将4个assignment的代码和实验报告放到压缩包中，命名为“lab2-姓名-学号”，并交到课程网站上[\[http://course.dds-sysu.tech/course/3/homework\]](http://course.dds-sysu.tech/course/3/homework)。

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。

Assignment 1 MBR

1.1

复现example 1。说说你是怎么做的，并将结果截图。

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。

Assignment 2 实模式中断

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。

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。

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**，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。

3.1

分支逻辑的实现

请将下列伪代码转换成汇编代码，并放置在标号 `your_if` 之后。

```
if a1 < 12 then
    if_flag = a1 / 2 + 1
else if a1 < 24 then
    if_flag = (24 - a1) * a1
else
    if_flag = a1 << 4
end
```

3.2

循环逻辑的实现

请将下列伪代码转换成汇编代码，并放置在标号 `your_while` 之后。

```
while a2 >= 12 then
    call my_random          // my_random将产生一个随机数放到eax中返回
    while_flag[a2 - 12] = eax
    --a2
end
```

3.3

函数的实现

请编写函数 `your_function` 并调用之，函数的内容是遍历字符数组 `string`。

```
your_function:
    for i = 0; string[i] != '\0'; ++i then
        pushad
        push string[i] to stack
        call print_a_char
        pop stack
        popad
    end
    return
end
```

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)(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。

二、实验过程：

1.1复现example 1:

输入example1的代码，保存为helloworld.asm文件，然后按照步骤进行，步骤图如下：

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: bash + [ ] [ ] ^ x
azhi@azhi-VirtualBox: ~$ nasm -f bin helloworld.asm -o helloworld.bin
nasm: fatal: unable to open input file `helloworld.asm'
azhi@azhi-VirtualBox: ~$ cd lab2
azhi@azhi-VirtualBox: ~/lab2$ nasm -f bin helloworld.asm -o helloworld.bin
azhi@azhi-VirtualBox: ~/lab2$ qemu-img create hd.img 10m
Formatting 'hd.img', fmt=raw size=10485760
azhi@azhi-VirtualBox: ~/lab2$ dd if=helloworld.bin of=hd.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, 0.00819915 s, 62.4 kB/s
azhi@azhi-VirtualBox: ~/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 operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.
azhi@azhi-VirtualBox: ~/lab2$ [ ]
```

刚开始，出现unable to open input file，是因为没有在helloworld.asm文件的目录下打开这个文件。所以后来进入当前的文件夹就可以打开了。最终显示结果如下：

```
QEMU
Hello Worldrsion 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD0+07ECDDDD0 C980

Booting from Hard Disk...
_
```

1.2在(12,12)处开始输出你的学号:

在example1的基础上，修改代码，

```
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 ax, 0xb800
mov gs, ax

mov ah, 0x073
mov al, '1'
mov [gs:2 *(80*12+12)], ax

mov al, '9'
mov [gs:2 *(80*12+13)], ax

mov al, '3'
mov [gs:2 *(80*12+14)], ax

mov al, '3'
mov [gs:2 *(80*12+15)], ax

mov al, '5'
mov [gs:2 *(80*12+16)], ax

mov al, '0'
mov [gs:2 *(80*12+17)], ax

mov al, '3'
mov [gs:2 *(80*12+18)], ax

mov al, '0'
mov [gs:2 *(80*12+19)], ax

jmp $ ; 死循环

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

```

其中 `mov ah, 0x073` 把输出背景设置为白色，将前景设置为青色。

因为要求从第12行第12列开始输出学号，因此输出的位置这样表示 `mov [gs:2 *(80*12+12)], ax`，同样的道理，在第12行第13列输出时，表示为 `mov [gs:2 *(80*12+13)], ax`。

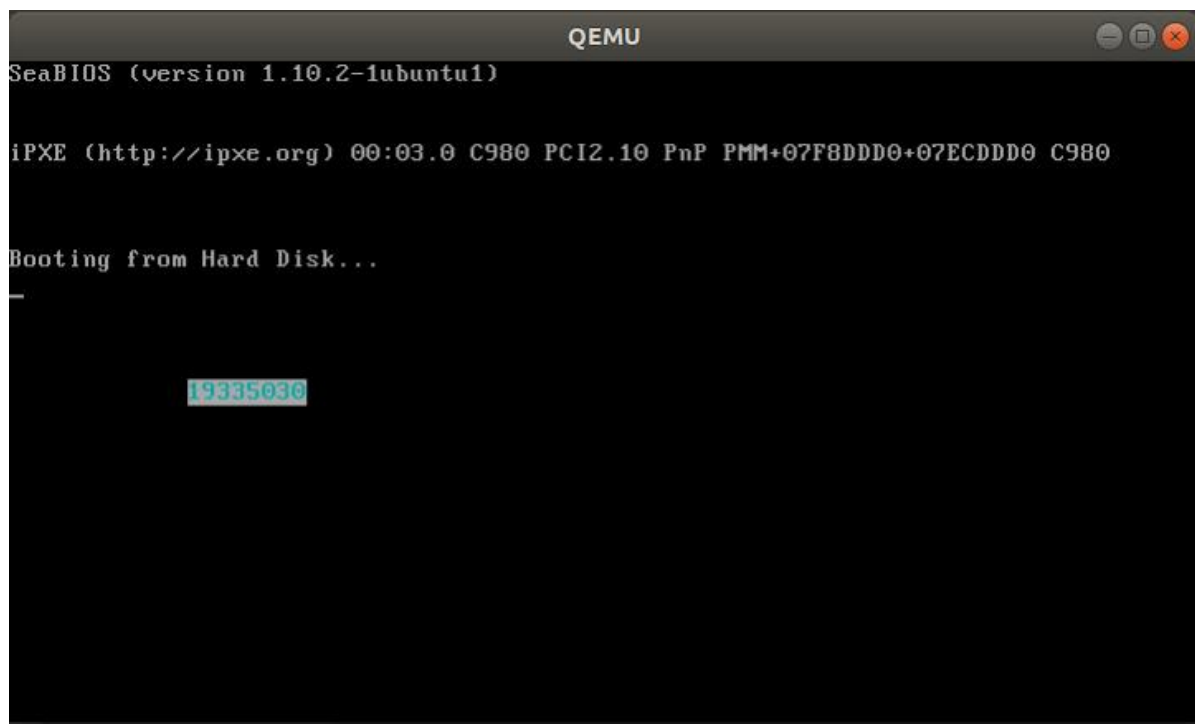
然后编译该文件，在QEMU上显示，具体执行步骤如下：

```

azhi@azhi-VirtualBox: ~/lab2$ nasm -f bin 19335030.asm -o 19335030.bin
azhi@azhi-VirtualBox: ~/lab2$ dd if=19335030.bin of=hd.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, 0.00632323 s, 81.0 kB/s
azhi@azhi-VirtualBox: ~/lab2$ qemu-i386 -hda hd.img -serial null -parallel stdio
qemu: unknown option 'hda'
azhi@azhi-VirtualBox: ~/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 operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

```

输出结果如下：



2.1利用中断实现光标的位置获取和光标的移动

实模式中断利用编码 `int 10h` 实现，具体功能实现如下表示：

功能	功能号	参数	返回值
设置光标位置	AH=02H	BH=页码，DH=行，DL=列	无
获取光标位置和形状	AH=03H	BX=页码	AX=0，CH=行扫描开始，CL=行扫描结束，DH=行，DL=列
在当前光标位置写字符和属性	AH=09H	AL=字符，BH=页码，BL=颜色，CX=输出字符的个数	无

所谓属性是指字符的颜色、背景颜色、是否闪烁、有没有底线等性质。在彩色显示卡 (CGA/EGA/VGA 等) 的文字模式中，颜色是用 4 个位表示，故可以表现出 16 种颜色，如下表：

二进制数	颜色	二进制数	颜色
0000	黑色	1000	灰色
0001	蓝色	1001	淡蓝色
0010	绿色	1010	淡绿色
0011	青色	1000	淡青色
0100	红色	1100	淡红色
0101	紫红色	1101	淡紫红色
0110	棕色	1110	黄色
0111	银色	1111	白色

取当前光标位置:

```
;Move the cursor to the next line
org 0x7c00
    FindPosition:
        mov ah,3
        mov bh,0
        int 10h

    jmp $ ; 死循环

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

ah=3功能为获取光标位置和形状, bh=0, 把页码放在第0页。

①先把.asm文件编译成.bin文件。

```
azhi@azhi-VirtualBox: ~/lab2$ nasm -f bin cursor.asm -o cursor.bin
azhi@azhi-VirtualBox: ~/lab2$ qemu-system-i386 cursor.bin
WARNING: Image format was not specified for 'cursor.bin' 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.
azhi@azhi-VirtualBox: ~/lab2$
```

②设置断点

```
azhi@azhi-VirtualBox: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
azhi@azhi-VirtualBox:~$ gdb
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote:1234
Remote debugging using :1234
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x0000ffff in ?? ()
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
```

③在gdb下利用命令 `x/30i $pc` 查看指令地址, 30为指明查看的指令地址数为30。

```
azhi@azhi-VirtualBox: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
gs          0x0      0
(gdb) x/30i $pc
=> 0x7c00:  mov     $0x3,%ah
    0x7c02:  mov     $0x0,%bh
    0x7c04:  int     $0x10
    0x7c06:  jmp     0x7c06
    0x7c08:  add     %al,(%eax)
    0x7c0a:  add     %al,(%eax)
    0x7c0c:  add     %al,(%eax)
    0x7c0e:  add     %al,(%eax)
    0x7c10:  add     %al,(%eax)
    0x7c12:  add     %al,(%eax)
    0x7c14:  add     %al,(%eax)
    0x7c16:  add     %al,(%eax)
    0x7c18:  add     %al,(%eax)
    0x7c1a:  add     %al,(%eax)
    0x7c1c:  add     %al,(%eax)
    0x7c1e:  add     %al,(%eax)
    0x7c20:  add     %al,(%eax)
    0x7c22:  add     %al,(%eax)
    0x7c24:  add     %al,(%eax)
    0x7c26:  add     %al,(%eax)
    0x7c28:  add     %al,(%eax)
    0x7c2a:  add     %al,(%eax)
```

④再设断点，这次把断点设在0x7c06,因为这里程序已经运行完了，后面的是在死循环。

```
0x7c2a:  add     %al,(%eax)
(gdb) b *0x7c06
Breakpoint 2 at 0x7c06
(gdb) c
Continuing.
```

设置完断点后按 c 继续执行程序。

⑤输入 info registers 显示寄存器信息，查看光标位置。由上表知道光标位置被储存在dx寄存器中，其中dh为行，dl为列。

```
azhi@azhi-VirtualBox: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
(gdb) b *0x7c06
Breakpoint 2 at 0x7c06
(gdb) c
Continuing.

Breakpoint 2, 0x00007c06 in ?? ()
(gdb) info registers
eax          0x355      853
ecx          0x607      1543
edx          0x800      2048
ebx          0x0        0
esp          0x6f04     0x6f04
ebp          0x0        0x0
esi          0x0        0
edi          0x0        0
eip          0x7c06     0x7c06
eflags      0x202      [ IF ]
cs          0x0        0
ss          0x0        0
ds          0x0        0
es          0x0        0
fs          0x0        0
gs          0x0        0
(gdb)
```

⑥因此我们知道了光标在第8行，第0列。

实现光标的移动：

修改代码为：

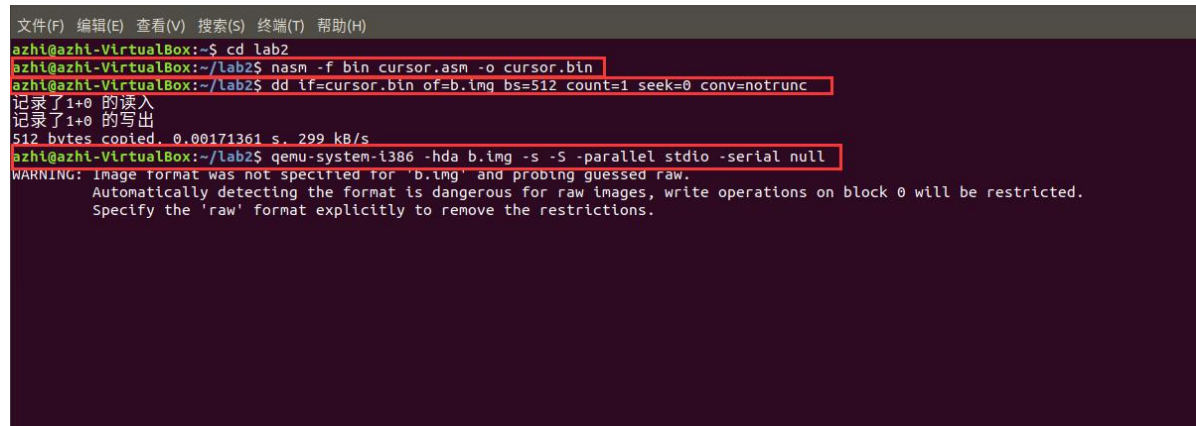
```
org 0x7c00
    Nextline:
    mov ah,3
    mov bh,0
    int 10h
;Move the cursor to the next line
    mov ah,2
    inc dl;
    mov dh,0
    int 10h

    jmp $ ; 死循环

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

这里有多次中断，第一次中断获取光标的位置，第二次中断实现光标的移动。`mov ah,2`的功能是设置光标位置。`inc dl`实现当前列号加1，`mov dh,0`将行号致0。

①将修改后的.asm文件编译成.bin文件，然后将生成的二进制文件写入b.img的首扇区，最后启动qemu。



```
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
azhi@azhi-VirtualBox:~$ cd lab2
azhi@azhi-VirtualBox:~/lab2$ nasm -f bin cursor.asm -o cursor.bin
azhi@azhi-VirtualBox:~/lab2$ dd if=cursor.bin of=b.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, 0.00171361 s, 299 kB/s
azhi@azhi-VirtualBox:~/lab2$ qemu-system-i386 -hda b.img -s -S -parallel stdio -serial null
WARNING: image format was not specified for 'b.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.
```

②在gdb下，用命令remote:1234,让gdb连上qemu，使用gdb进行调试。


```

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
azhi@azhi-VirtualBox:~$ cd lab2
azhi@azhi-VirtualBox:~/lab2$ gdb
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote:1234
Remote debugging using :1234
warning: No executable has been specified and target does not support
determining executable automatically. Try using the "file" command.
0x0000ffff in ?? ()
(gdb) b *0x7c00
breakpoint 1 at 0x7c00
(gdb) c
continuing.

Breakpoint 1, 0x00007c00 in ?? ()
(gdb) x/30i $pc
=> 0x7c00:      mov     $0x3,%ah
0x7c02:      mov     $0x0,%bh
0x7c04:      int     $0x10
0x7c06:      mov     $0x2,%ah
0x7c08:      inc     %dl
0x7c0a:      mov     $0x0,%dh

```

③还是上图，设置第一个断点0x7c00，然后运行，运行后查看指令的地址，进一步设置断点位置。

④用命令 `x/30i $pc` 查看指令地址，显示如下。这里选取了关键的指令 `mov ah,3` 和 `jmp $` 作为断点，可以查看到两次中断前后的对比。

```
Breakpoint 1, 0x00007c00 in ?? ()
(gdb) x/30i $pc
=> 0x7c00:    mov     $0x3,%ah
0x7c02:    mov     $0x0,%bh
0x7c04:    int     $0x10
0x7c06:    mov     $0x2,%ah
0x7c08:    inc     %dl
0x7c0a:    mov     $0x0,%dh
0x7c0c:    int     $0x10
0x7c0e:    jmp     0x7c0e
0x7c10:    add     %al,(%eax)
0x7c12:    add     %al,(%eax)
0x7c14:    add     %al,(%eax)
0x7c16:    add     %al,(%eax)
0x7c18:    add     %al,(%eax)
0x7c1a:    add     %al,(%eax)
0x7c1c:    add     %al,(%eax)
0x7c1e:    add     %al,(%eax)
0x7c20:    add     %al,(%eax)
0x7c22:    add     %al,(%eax)
0x7c24:    add     %al,(%eax)
0x7c26:    add     %al,(%eax)
0x7c28:    add     %al,(%eax)
0x7c2a:    add     %al,(%eax)
0x7c2c:    add     %al,(%eax)
---Type <return> to continue, or q <return> to quit---
0x7c2e:    add     %al,(%eax)
0x7c30:    add     %al,(%eax)
0x7c32:    add     %al,(%eax)
0x7c34:    add     %al,(%eax)
0x7c36:    add     %al,(%eax)
0x7c38:    add     %al,(%eax)
0x7c3a:    add     %al,(%eax)
(gdb) info registers
```

⑤可以看到断点在0x7c06时，`mov ah, 3`还没执行，这时的ah值为3，dh值为0，dl值为0x80。

```
(gdb) b 0x7c06
Breakpoint 3 at 0x7c06
(gdb) info registers
eax          0x355      853
ecx          0x0        0
edx          0x80       128
ebx          0x0        0
esp          0x6f04     0x6f04
ebp          0x0        0x0
esi          0x0        0
edi          0x0        0
eip          0x7c04     0x7c04
eflags      0x202      [ IF ]
cs           0x0        0
ss           0x0        0
ds           0x0        0
es           0x0        0
fs           0x0        0
gs           0x0        0
```

⑥断点在0x7c0e时，获取光标位置和修改光标位置的指令都已经执行完，可以看到此时ah变为了2，dh变为了0，dl变为了1，这和代码对光标的修改一致。因此光标移动成功。

```
Breakpoint 5, 0x00007c0e in ?? ()
(gdb) info registers
eax            0x255      597
ecx            0x607      1543
edx            0x1        1
ebx            0x0        0
esp            0x6f04     0x6f04
ebp            0x0        0x0
esi            0x0        0
edi            0x0        0
eip            0x7c0e     0x7c0e
eflags         0x202     [ IF ]
cs             0x0        0
ss             0x0        0
ds             0x0        0
es             0x0        0
fs             0x0        0
gs             0x0        0
(gdb)
```

2.2使用实模式下的终端输出学号

首先令 ah=2 设置光标的位置为 (0,0)，即第0行第0列。然后令 ah=9，设置要写的字符，注意这里要用ASCII码，并让 bh=9 设置字符的颜色为蓝色，cx 设置要输出的字符个数为1，每次输出一个字符。

每写好一个字符，都要更新光标的位置。ah=9;int 10h 的功能是在当前光标位置写字符，因此每写完一次字符都要更新光标的位置。否则输出会不符合预期。下面是具体实现：

```
org 0x7c00
write:
; set the position of cursor
; remember to set bh=0
mov bh,0
mov ah,2
mov dh,0
mov dl,0
int 10h

; write my first number
mov ah,9
mov al,49
mov bl,9
mov cx,1
int 10h

; remember to change the position of cursor
mov ah,2
mov bh,0
inc dl
int 10h
```

```
; write the rest numbers
mov ah,9
mov al,57
mov bl,9
mov cx,1
int 10h

; remember to change the position of cursor
mov ah,2
mov bh,0
inc dl
int 10h

mov ah,9
mov al,51
mov bl,9
mov cx,1
int 10h

; remember to change the position of cursor
mov ah,2
mov bh,0
inc dl
int 10h

mov ah,9
mov al,51
mov bl,9
mov cx,1
int 10h

; remember to change the position of cursor
mov ah,2
mov bh,0
inc dl
int 10h

mov ah,9
mov al,53
mov cx,1
mov bl,9
int 10h

; remember to change the position of cursor
mov ah,2
mov bh,0
inc dl
int 10h

mov ah,9
mov al,48
mov cx,1
mov bl,9
int 10h

; remember to change the position of cursor
mov ah,2
mov bh,0
```

```

inc dl
int 10h

mov ah,9
mov al,51
mov bl,9
mov cx,1
int 10h

; remember to change the position of cursor
mov ah,2
mov bh,0
inc dl
int 10h

mov ah,9
mov al,48
mov cx,1
mov bl,9
int 10h

jmp $ ; 死循环

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

```

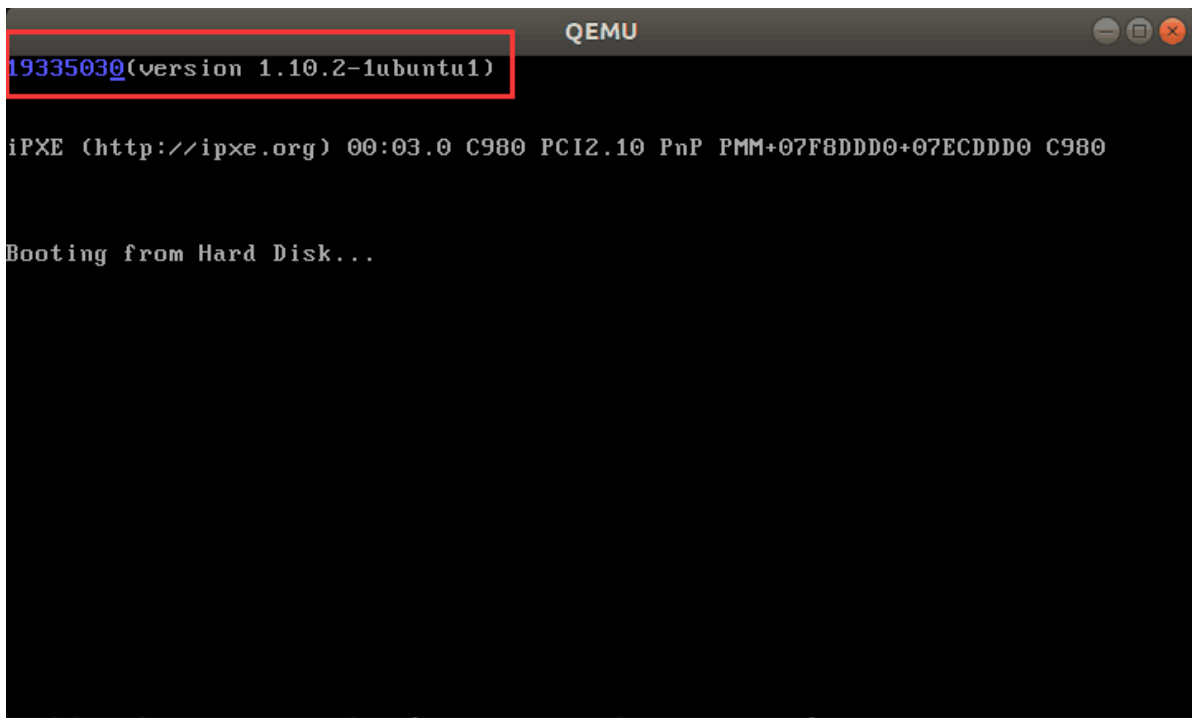
①按照以下步骤编译运行，并在qemu端显示。

```

azhi@azhi-VirtualBox: ~/lab2$ nasm -f bin write.asm -o write.bin
azhi@azhi-VirtualBox: ~/lab2$ dd if=write.bin of=b.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, 0.00371337 s, 138 kB/s
azhi@azhi-VirtualBox: ~/lab2$ qemu-system-i386 -hda b.img -serial null -parallel stdio
WARNING: Image format was not specified for 'b.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.
azhi@azhi-VirtualBox: ~/lab2$ █

```

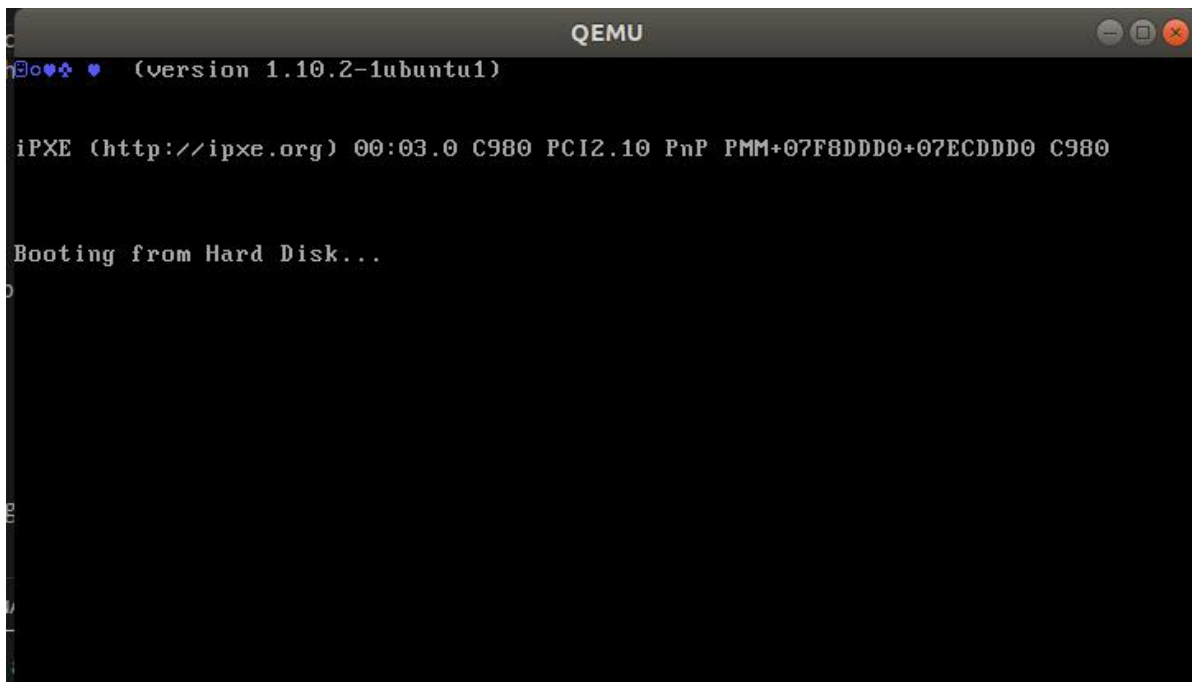
②显示结果



```
QEMU
19335030(version 1.10.2-1ubuntu1)
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD0+07ECDDDD0 C980
Booting from Hard Disk...
```

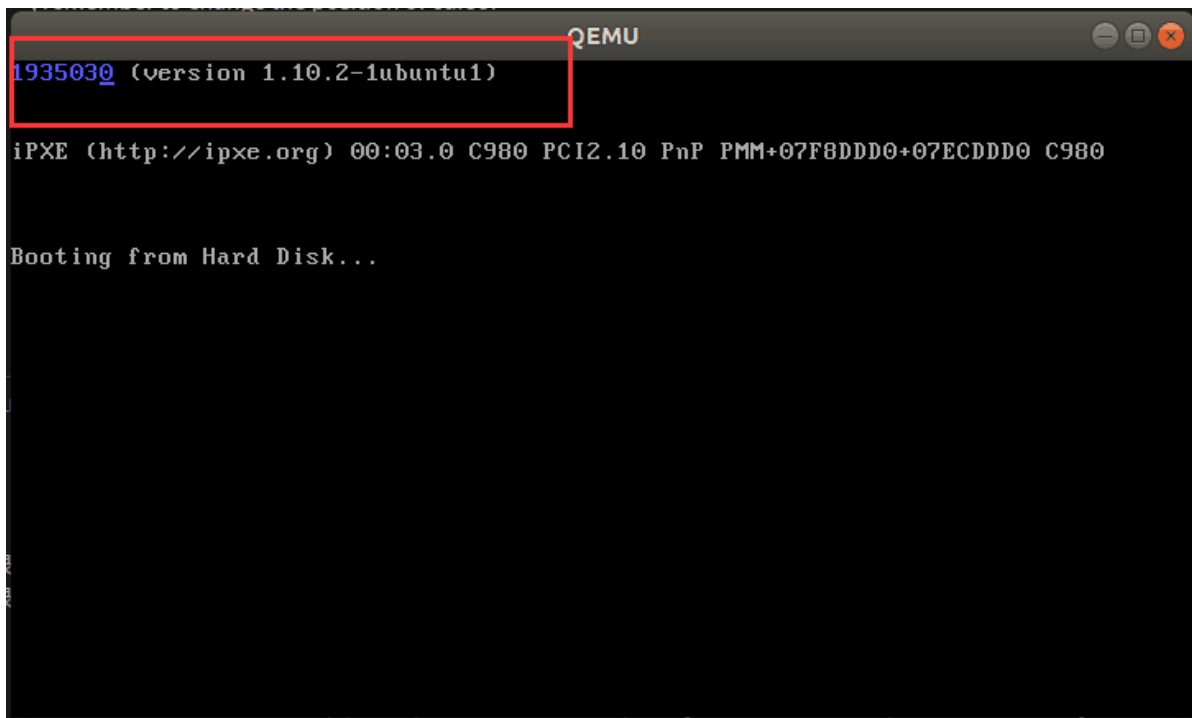
试错:

a、刚开始的时候,我把要输出的数字值直接赋给了al,结果显示的是乱码,才发现要赋相应字符的ASCII码。



```
QEMU
19335030(version 1.10.2-1ubuntu1)
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDDD0+07ECDDDD0 C980
Booting from Hard Disk...
```

b、修改了ASCII码之后,输出发现是想要的字符了,但是在两个连续的3那里却少了一个,为什么会这样呢?我是在int 10h, ah=9的模式下,让cx=2,即输出的字符个数是2,结果却没有显示成功。原因是后面光标只移了一位,写5时把原来的3覆盖住了。于是我老实实在地一个个输出,终于得到正确的输出。



2.3利用键盘终端实现键盘输入并回显

代码如下：

```
org 0x7c00
rw_keyboard:

mov ah,0x00
int 0x16

mov ah,0x0e
mov bh,0x07
int 0x10

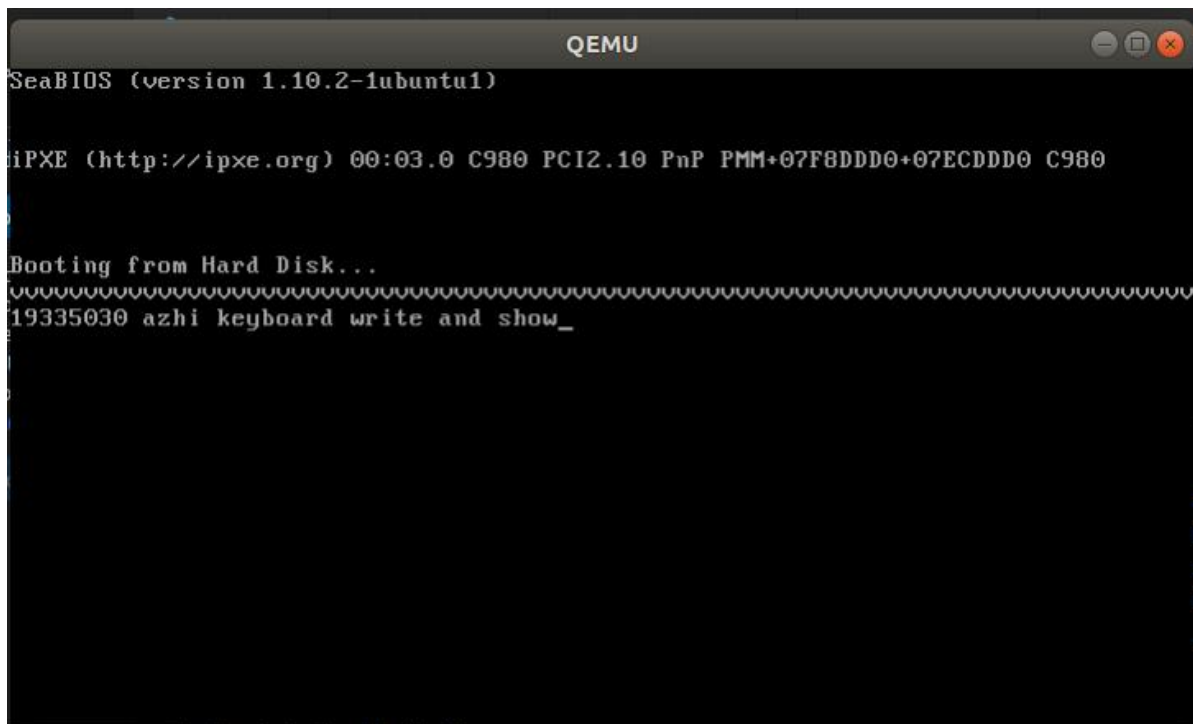
jmp rw_keyboard

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

先在 `int 16h`，`ah=0` 模式下从键盘读入字符，输入的字符转化成ASCII码储存在`al`寄存器中。读入一个字符后，切换到 `int 10h`，`ah=0x0e` 模式，显示字符，这里每显示一个字符，光标就会自动往右移一格。`bh`默认为0。`bh`设置字符前景颜色，`bh=0x07`，前景颜色为银色。

运行结果如下：

在弹出的qemu界面中输入字符，可以看到相应的输出。实现成功。



3.1分支逻辑的实现:

```
; If you meet compile error, try 'sudo apt install gcc-multilib g++-multilib'
first
[BITS 32]
#include "head.include"
; you code here

your_if:
mov edx,[a1]
cmp edx,24
jge larger_than_24
cmp edx,12
jge smaller_than_24
jge samller_than_12

samller_than_12:
mov eax,[a1]
mov ebx,2
idiv ebx
inc eax
mov [if_flag],eax
jmp end

smaller_than_24:
mov eax,24
sub eax,[a1]
imul eax,edx
mov [if_flag],eax
jmp end

larger_than_24:
mov eax,[a1]
shl eax,4
mov [if_flag],eax
```



```
jmp end
```

```
end:
```

if判断语句通过cmp和jmp指令配合实现。三种if条件，分别为三种情况：smaller_than_12, smaller_than_24, larger_than_24. 每种情况运行完后一定要记得jmp end。不然程序执行了一种情况后，可能会执行另一种情况。

取变量的值和给变量赋值都要用 '[' 和 ']'。

3.2 循环逻辑的实现：

```
; put your implementation here
your_while:
loop:
mov edx,[a2]
cmp edx,12
jl end_loop
call my_random
mov ebx,[while_flag]
; !!!! because the value of edx had been changed while exe cmp edx,12
mov edx,[a2]
mov byte[ebx+(edx-12)],a1
dec edx
mov [a2],edx
jmp loop
end_loop:
```

while依然是使用cmp和跳转指令实现。先将a2的值保存在寄存器edx上，再将edx的值和12比较，根据比较结构进行跳转。这里要注意cmp指令执行完后，寄存器edx的值是做了减法的，因此在后面利用a2的值进行寻址时，一定要记得再对edx赋值，否则，虽然没有报错，但是运行结果将不符合预期。

3.3 函数的实现：

```
; put your implementation here
#include "end.include"

your_function:
; put your implementation here

xor eax,eax
xor ecx,ecx
mov ebx,[your_string]
for:
mov cl,byte[eax+ebx] ; byte corresponds to cl, not ecx
inc eax
cmp ecx,0
je end_for
pushad
push ecx
call print_a_char
pop ecx
popad
```

```

jmp for

end_for:

ret

```

先初始化eax和ecx为0. 再用寄存器ebx存数组your_string的地址。注意，字符大小是byte。因此在读具体字符到寄存器时要用byte【】来读，寄存器的大小也要和byte对应，因此用来cl，而不是exc，如果大小不对应，编译时会报错。其他按照思路进行即可。

编译运行：

```

azhi@azhi-VirtualBox: ~/lab2/assignment
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
azhi@azhi-VirtualBox:~$ cd lab2
azhi@azhi-VirtualBox:~/lab2$ cd assignment
azhi@azhi-VirtualBox:~/lab2/assignment$ make fun
make: *** 没有规则可制作目标“fun”。 停止。
azhi@azhi-VirtualBox:~/lab2/assignment$ make run
>>> begin test
>>> if test pass!
>>> while test pass!
Mr.Chen, students and TAs are the best!
azhi@azhi-VirtualBox:~/lab2/assignment$

```

进入相应目录，在终端输入命令 `make run`，可以看到输出的信息，没有提示错误。证明成功。

assignment4: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)(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。静态示例效果如下，动态效果见视频 [assignment/assignment-4-example.mp4](#)。

代码：

```

org                07c00h
mov                ax, cs
mov                ds, ax
mov                es, ax
mov                bx, 007ch                ;初始化要打印的页号为0，并设置背景为白色和前景颜色为
浅红色

loop1:
    dec word[count]                ; 递减计数变量

```

```

    jnz loop1                ; >0: 跳转;
    mov word[count],delay
    dec word[dcount]        ; 递减计数变量
    jnz loop1
    mov word[count],delay
    mov word[dcount],ddelay
    ; 用二重循环实现时延50*500个单位时间

    jmp      Entrance      ;进行一个周期的工作
    jmp      $              ;halt

Entrance:
    jmp      BoundaryCheckx

DispStr:
    mov      ax, BootMessage    ;打印字符串
    mov      bp, ax
    mov      cl, byte[Strlen]   ;字符串长
    mov      ch, 0
    mov      ax, 1301h          ;写模式
;    mov      bx, 003fh          ;页号0, 黑底白字
    mov      dh, byte[x]        ;行=x
    mov      dl, byte[y]        ;列=y
    int      10h                ;10h号接口

;更新状态
Updatexy:
    mov      al, byte[x]
    add      al, byte[vx]
    mov      byte[x], al
    mov      al, byte[y]
    add      al, byte[vy]
    mov      byte[y], al

;颜色变化
changeCol:
    mov      dx, bx
    cmp      dx, 001eh
    jz       col1
    mov      dx, bx
    cmp      dx, 005ah
    jz       col2
    mov      dx, bx
    cmp      dx, 006bh
    jz       col3
    mov      dx, bx
    cmp      dx, 0025h
    jz       col4
    mov      dx, bx
    cmp      dx, 0042h
    jz       col5
    mov      dx, bx
    cmp      dx, 003fh
    jz       col6
    mov      dx, bx
    cmp      dx, 007ch
    jz       col7

;jmp      loop1              ;无限循环

```

BoundaryCheckx:

```
mov     al, byte[x]
add     al, byte[vx]    ;预测下一刻的x
cmp     al, byte[upper] ;如果x小于上边界
jl      Changevx       ;更新vx
cmp     al, byte[lower] ;如果x大于下边界
jg      Changevx       ;更新vx
```

BoundaryChecky:

```
mov     al, byte[y]
add     al, byte[vy]
cmp     al, byte[left]  ;如果y小于左边界
jl      Changevy       ;更新vy
add     al, byte[Strlen];预测下一刻的yr=y+字符串长
cmp     al, byte[right] ;如果yr大于下边界
jg      Changevy       ;更新vy
jmp     DispStr         ;如果不需要更新vx vy就继续打印流程
```

Changevx:

```
neg     byte[vx]
jmp     BoundaryChecky
```

Changevy:

```
neg     byte[vy]
jmp     DispStr
```

;手动设置7种颜色

```
col1:
    mov bx,003fh
    jmp     loop1
```

```
col2:
    mov bx,007ch
    jmp     loop1
```

```
col3:
    mov bx,001eh
    jmp     loop1
```

```
col4:
    mov bx,005ah
    jmp     loop1
```

```
col5:
    mov bx,006bh
    jmp     loop1
```

```
col6:
    mov bx,0025h
    jmp     loop1
```

```
col7:
    mov bx,042h
    jmp     loop1
```

```

BootMessage:      db      "zhi" ;输出的字符
strlen            db      3      ;输出的字符长度
delay             equ      50
ddelay            equ      500
count             dw      delay
dcount            dw      ddelay

x                 db      0
y                 db      2
vx                db      1
vy                db      1
left              db      0
upper             db      0
right             db      79      ;这个宽度根据屏幕大小来定
lower            db      24

boundary          db      10

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

```

代码解析：

开始的org 0x7c00h让编译器把代码装载到0x7c00处，把它编译成BIN文件时，执行时PC会识别到扇区末的0xaa55代号，认为它是引导扇区，然后会把这个扇区放到0x7c00处，把程序计数器也记为7c00h，从这里开始执行程序。从这里开始我们接管PC。

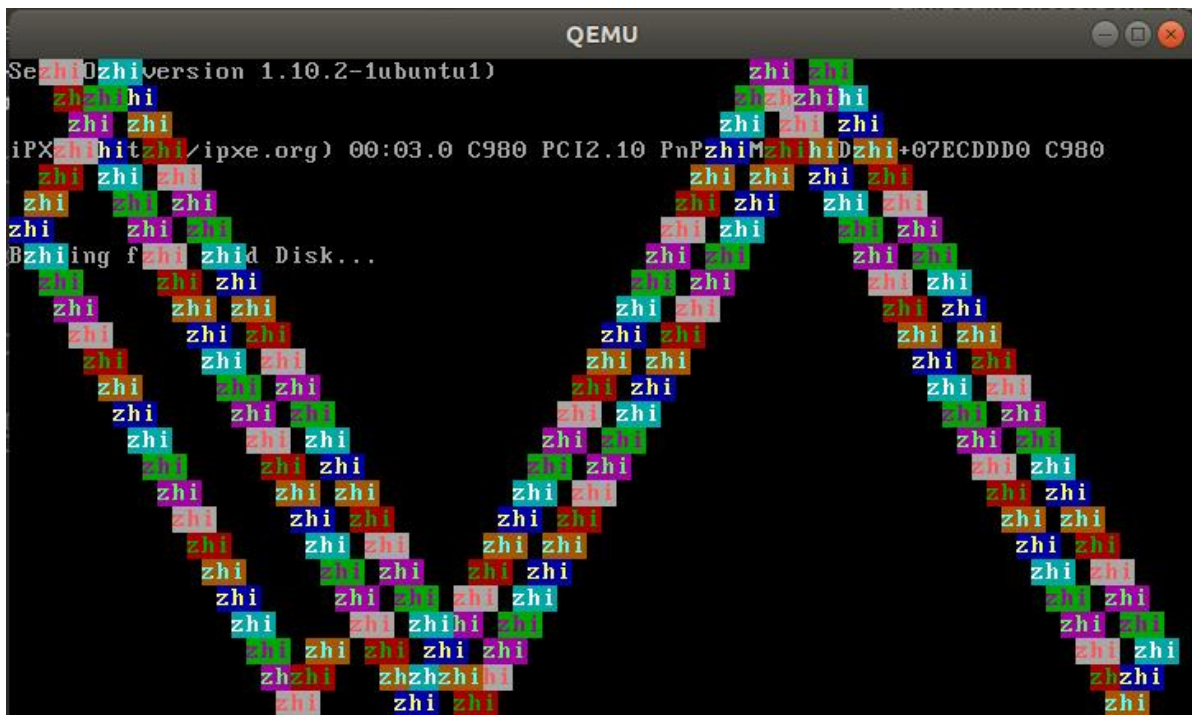
上面的代码可以分为三个部分，第一个部分实现计时，第二个部分是打印字符的具体实现，第三个部分是相关的信息。

首先第一个部分由loop1实现计时。loop1里面有两层嵌套循环，实现时延50*500个单位时间。每过一个时延就执行一次字符串的打印。

第二部分的实现又分为实现字符串的打印，更新最新状态（包括字符串输出的下一个位置、颜色）。更新字符串的位置的时候需要对字符串的位置进行判断，如果到了边界就要更新下一次的移动方向，如果没到边界x和y就简单地执行+1操作。这里手动设置了7种前景和背景不同的颜色，每输出一次就换一种颜色，利用cmp和跳转指令实现。

第三部分交代了要输出的字符串为“zhi”，输出的字符长度为3，还设置了两个延时时常。还设置了字符串输出的上下界和左右界，这个要根据屏幕大小设置。x，y设置为（0,2）表示开始的位置为(2,0)点开始移动。

运行：



再过一会就是满屏幕的“zhi”了，能感觉到它在动了么？

三、实验感想：

这次实验内容比较多，也因为刚开始对x86汇编语言不熟悉，所以做的过程遇到了很多困难。

但是一步步做下来之后收获还是比较大的。主要收获有：

- 1、assignment1 和 assignment2 让我对编译和调试arm文件的过程和qemu的使用更加熟悉，对光标的处理等等也让我对中断指令的应用更熟练。
- 2、开始只对MIPS汇编有很肤浅的了解，经过实验，实现了三段伪代码的汇编转换，也实现了一个字符弹射的小程序，对x86汇编的寄存器和指令的使用有了更深入的了解。比如mov指令的多种格式的含义、cmp指令的结果影响寄存器的值，跳转指令执行后会继续执行下面的代码等等。

3、由于这次实验对我来说难度比较大，因此我也和同学针对问题进行了交流和探讨，这让我懂得还是人多力量大，大家一起讨论的时候，会分享所学，进步就会比较快。