



中山大學
SUN YAT-SEN UNIVERSITY

操作系统实验 2

汇编语言编程实验

实验课程: 操作系统原理实验

实验名称: 汇编语言编程实验

专业名称:

学生姓名: 计算机科学与技术

学生学号: 钟旺烜

实验地点: 23336342

实验成绩: 实验楼 B203

报告时间: 2025 年 3 月 23 日

Section 1 实验概述

●实验任务 1: MBR。

1.1、复现实验 2 参考资料中的 example 1 部分。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(16,10)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

●实验任务 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，可以参考该资料。关于键盘扫描码，可以参考该资料。说说你是怎么做的，并将结果截图

●实验任务 3: 汇编。分别将 assignment3 中给出的分支逻辑、循环逻辑和函数的实现的伪代码转化成汇编代码

●实验任务 4: 汇编小程序。字符回旋程序。请你用汇编代码实现一个字符回旋程序，使其能够在 qemu 显示屏上面，以不同的颜色、字符内容进行顺时针绕圈（见下图）。请注意，对于这个绕圈的字符，你还需要考虑他的运动速度，不要让其运动过快（可自行查阅其他资料，采用合适的中断来实现延时效果）

Section 2 实验步骤与实验结果

----- 实验任务 1 -----

●任务要求：

✎Assignment 1 MBR

注意，assignment 1的寄存器请使用16位的寄存器。

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(16,10)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。

●思路分析：利用课程提供的 lab2 参考资料，example1 中通过将程序放入存储设备首扇区，使计算机启动后可以通过加载 MBR 来运行我们事先存储在存储设备首扇区程序。现在我们要实现在 qemu 显示屏上显示相应的字符，如何显示字符的相关原理已经在 lab2 参考资料里有详细的解释，这里不再赘述。有关 MBR 的编写，实验 1.1 也给出了相关的原代码，按照指导步骤一一实现即可。有关通过 qemu 启动操作系统的流程代码也在资料中展现。对于实验 1.2，在实验 1.1 的代码的基础上，修改存储字符颜色地址处的数值，以及相应存储字符位置的地址和字符内容的地址的数值即可，实现过程基本相同

●实验步骤：

实验 1.1

根据实验资料步骤依次执行：首先在创建的 lab2 文件夹中使用 vscode 编写要运行的程序文件，代码内容和 lab2 实验资料中的完全相同，故不做展示，代码名称命名为 code1.1。

随后，在 terminal 中输入指令：

```
qemu-img create hd.img 10m
```

创建一个 10m 的磁盘 hd.img。再输入指令

```
nasm -f bin code1.1 -o code1.1.bin
```

使用 nasm 汇编器来将代码编译成二进制文件。输入指令

```
dd if=code1.1.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

然后将 MBR 写入 hd.img 的首扇区，写入的命令使用的是 linux 下的 dd 命令。

最后，我们启动 qemu 来模拟计算机启动了，命令如下。

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

启动后可以在 qemu 显示屏上看到第一行已经输出“Hello World”。

实验 1.2

实验 1.2 的操作过程与实验 1.1 相似。我们使用 vscode 在文件 code1.2 中编写程序代码。code1.1 的代码在 code1.2 的基础上在存储字符颜色地址的数据、显示字符位置的地址以及存储显示字符内容地址的数据进行了更改，具体改动的关键代码内容由下端代码所示

```
mov ah, 0x74
mov al, '2'
mov [gs:2 * 1290], ax

mov al, '3'
mov [gs:2 * 1291], ax

mov al, '3'
```

```

mov [gs:2 * 1292], ax

mov al, '3'
mov [gs:2 * 1293], ax

mov al, '6'
mov [gs:2 * 1294], ax

mov al, '3'
mov [gs:2 * 1295], ax

mov al, '4'
mov [gs:2 * 1296], ax

mov al, '2'
mov [gs:2 * 1297], ax

```

如上述代码所示，将存储在显示字符地址的高 8 位 ah 寄存器的值设为 0x74,既白底红字；因为实验要求在（16，10）开始显示本人学号 23336342，故从地址 $2 * (80 * 16 + 10) = 2 * 1290 = 2580$ 开始，每隔 2 位地址存入我们需要显示的字符数据。

代码修改完后，使用 nasm 汇编器来将代码编译成二进制文件，输入指令

然后将 MBR 写入 hd.img 的首扇区，写入的命令使用的是 linux 下的 dd 命令。

```
dd if=code1.2.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

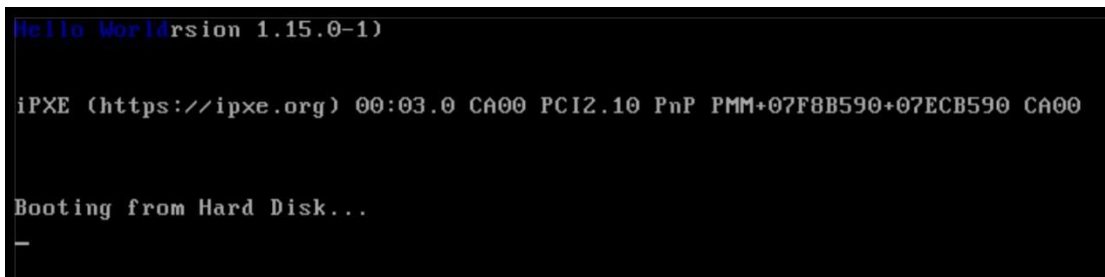
最后，我们启动 qemu 来模拟计算机启动了，命令如下。

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

启动后可以在 qemu 显示屏的 16 行 10 列显示白底红字的学号 23336342

●实验结果展示：通过执行前述代码，可得下图结果。

实验 1.1



```

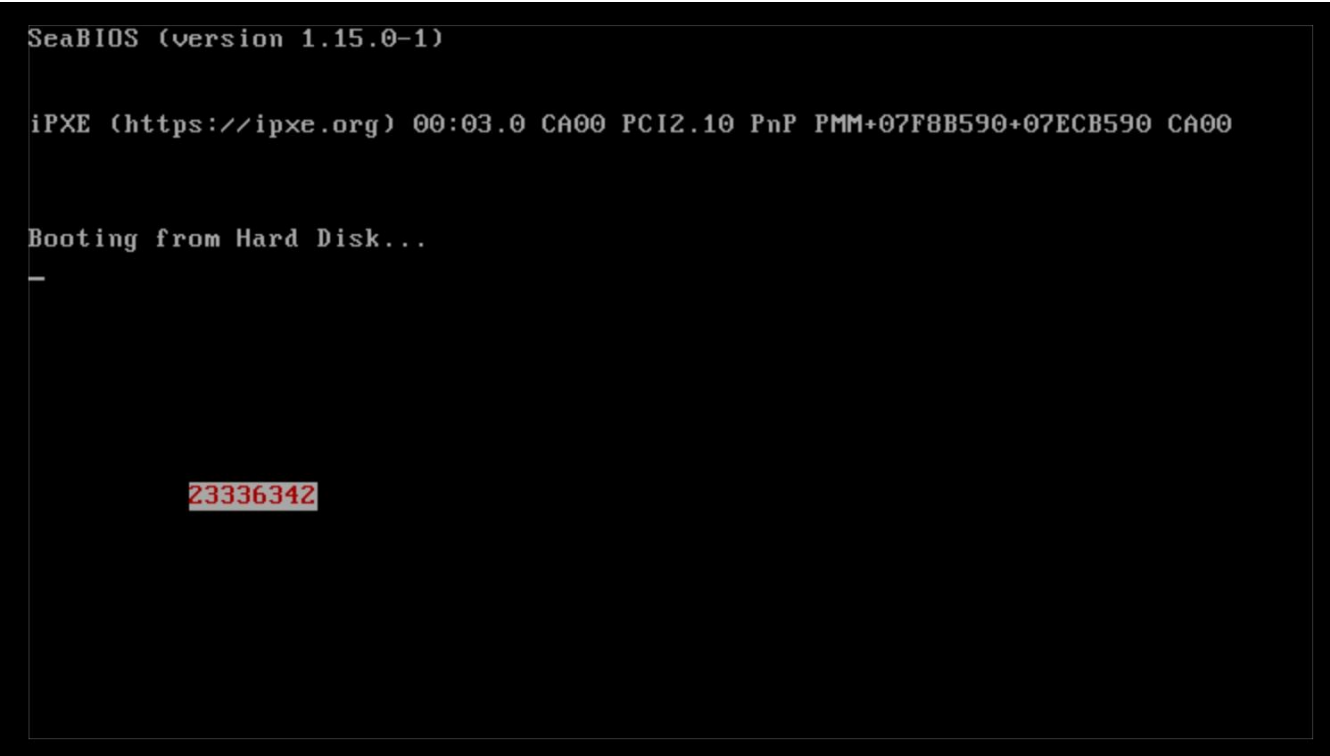
Hello Worldrsion 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B590+07ECB590 CA00

Booting from Hard Disk...
-

```

实验 1.2



----- 实验任务 2 -----

●任务要求：

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[该资料](#)。关于键盘扫描码，可以参考[该资料](#)。说说你是怎么做的，并将结果截图。

●思路分析：

实验 2 主要围绕实模式中断的相关操作，实验 2.1 和 2.2 使用的都是实模式中断 int 10h，由于功能号不同，执行的结果也就不同。下图是本次实验 2.1 和 2.2 会使用到的功能号：

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

根据功能号及其各参数所代表的各项功能，加上如下段文字所示的中断的调用方式
将参数和功能号写入寄存器

int 中断号

从寄存器中取出返回值

我们可以通过编写对应的代码来实现实模式的中断进而实现实验的要求

实验 2.1

实验 2.1 要求利用中断实现光标的位置获取和光标的移动，在实模式中断 int 10h 下，利用功能号 AH=02H 和 AH=03H 便可分别实现光标的移动和获得光标的位置。获取光标的位置只需要在切换实模式中断 int 10h 前先将 AH=03H 设置好，随后读出 DH 和 DL 寄存器的返回值即可得到光标在屏幕的行数和列数；而实现光标的移动需要在切换实模式中断 int 10h 前分别设置好寄存器 BH, DH 和 DL 的值。根据上表的功能图，可知 BH 设置的是页码数，通常设置位零，而 DH 和 DL 分别设置的是光标位置在屏幕的行数和列数,设置好后便可在屏幕上的相应位置显示光标。

实验 2.2

为了使用实模式下的中断输出学号，我们要使用到实模式中断 int 10h 下的 AH=02H 和 AH=09H 的功能号。前者能设置光标位置，而后者能在当前光标位置写字符和属性。模仿 code1.2，我们采取每移动一次光标写一个字符的方法，在两个功能号之间不断切换来实现学号的输出

实验 2.3

实模式的键盘中断可以通过 int 16 来实现。其中 AH=00H 功能号可以从键盘读取字符。执行时，等待键盘输入，一旦输入，字符的 ASCII 码便放入 AL 寄存器中。随后我们利用实模式中断 int 10h 中的 AH=09H 功能号来将储存在 AL 寄存器的字符打印在初始光标处，再利用 AH=02 功能号将光标向左移动，并在移动到屏幕最右侧时进行换行操作，来实现实现键盘输入并回显

●实验步骤：

实验 2.1

我们依然是通过编写将放入存储设备首扇区的程序代码，通过加载 MBR 来利用中断实现光标的位置获取和光标的移动。代码编写在 code2.1 中，编写的代码如下：

```
org 0x7c00
[bits 16]

;get cursor position
mov ah, 0x03
mov bh, 0

int 10h
;set cursor position
mov ah, 0x02
mov bh, 0
```

```

add dh, 3
add dl, 3

int 10h

jmp $

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

```

随后我们先将代码转化为二进制文件：

```
nasm -f bin code2.1 -o code2.1.bin
```

再将二进制文件放入 `hd.img` 中，使用 `qemu` 运行系统

```
dd if=code2.1.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

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

得到光标坐标后，将光标位置的行列分别加三，通过 `qemu` 屏幕显示可以得到光标设置变化后的位置

实验 2.2

仿造实验 1.2 的代码，我们通过实模式中断 `int 10h` 下的 `AH = 02H` 和 `AH=09H` 功能号的不断切换来实现学号的输出，依然通过加载 `MBR` 来实现，代码编写在 `code2.2` 中，关键代码部分如下：

```

org 0x7c00
[bits 16]

mov bx, 0
mov dh, 16
mov dl, 10
mov ah, 0x02
int 10h

mov bl, 0x74

print:
    mov cx, 1

    mov ah, 0x02
    int 10h
    mov al, '2'
    mov ah, 0x09
    int 10h
    inc dl
    mov ah, 0x02
    int 10h
    mov al, '3'
    mov ah, 0x09
    int 10h

```

代码首先利用 AH = 02H 功能号将光标设置在（16，10）位置，随后利用 AH = 03H 功能号打印白底红字字符‘2’，再调回 AH = 02H 功能号将光标向右移动一列，再利用 AH = 03H 功能号打印白底红字字符‘3’；依次循环进行，直到能够打印完整的学号‘23336342’。

随后我们先将代码转化为二进制文件：

```
nasm -f bin code2.2 -o code2.2.bin
```

再将二进制文件放入 hd.img 中，使用 qemu 运行系统

```
dd if=code2.2.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

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

随后在 qemu 显示屏中我们可以看到在（16，10）开始显示的白底红字的学号‘23336342’

实验 2.3

利用键盘中断实现键盘输入并回显，我们要利用实模式中断 int 16h 读取键盘输入，并使用实模式中断 int 10h 实现键盘输入的回显。我们还要注意光标的换行处理。

将编写的代码放入 code2.3 中，代码如下：

```
org 0x7c00
[bits 16]

mov dh, 16
mov dl, 10

Input:
mov ah, 0x00
int 0x16
or al, 0x00
jnz print_key
jmp Input

print_key:
call cursor_inc

mov ah, 0x09
mov bl, 0x74
mov cx, 0x01
int 0x10
jmp Input

cursor_inc:
inc dl
cmp dl, 80
jne add_cursor
mov dl, 0
inc dh
add_cursor:
```



```
    mov ah, 2; set cursor +1
    int 10h
    ret

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

行在代码中，我将光标初始放在了（16，10）处，原则上来说初始光标可以放在屏幕的任意处。光标换行的逻辑主要在与在每次调用 `cursor_inc` 部分进行 `dl` 值自增 1 实现光标右移时，检查 `dl` 值是否达到了 80（列的上限值），如果达到，就会将 `dl` 值归为零，`dh` 值自增 1，来实现光标的换行

随后我们先将代码转化为二进制文件：

```
nasm -f bin code2.3 -o code2.3.bin
```

再将二进制文件放入 `hd.img` 中，使用 `qemu` 运行系统

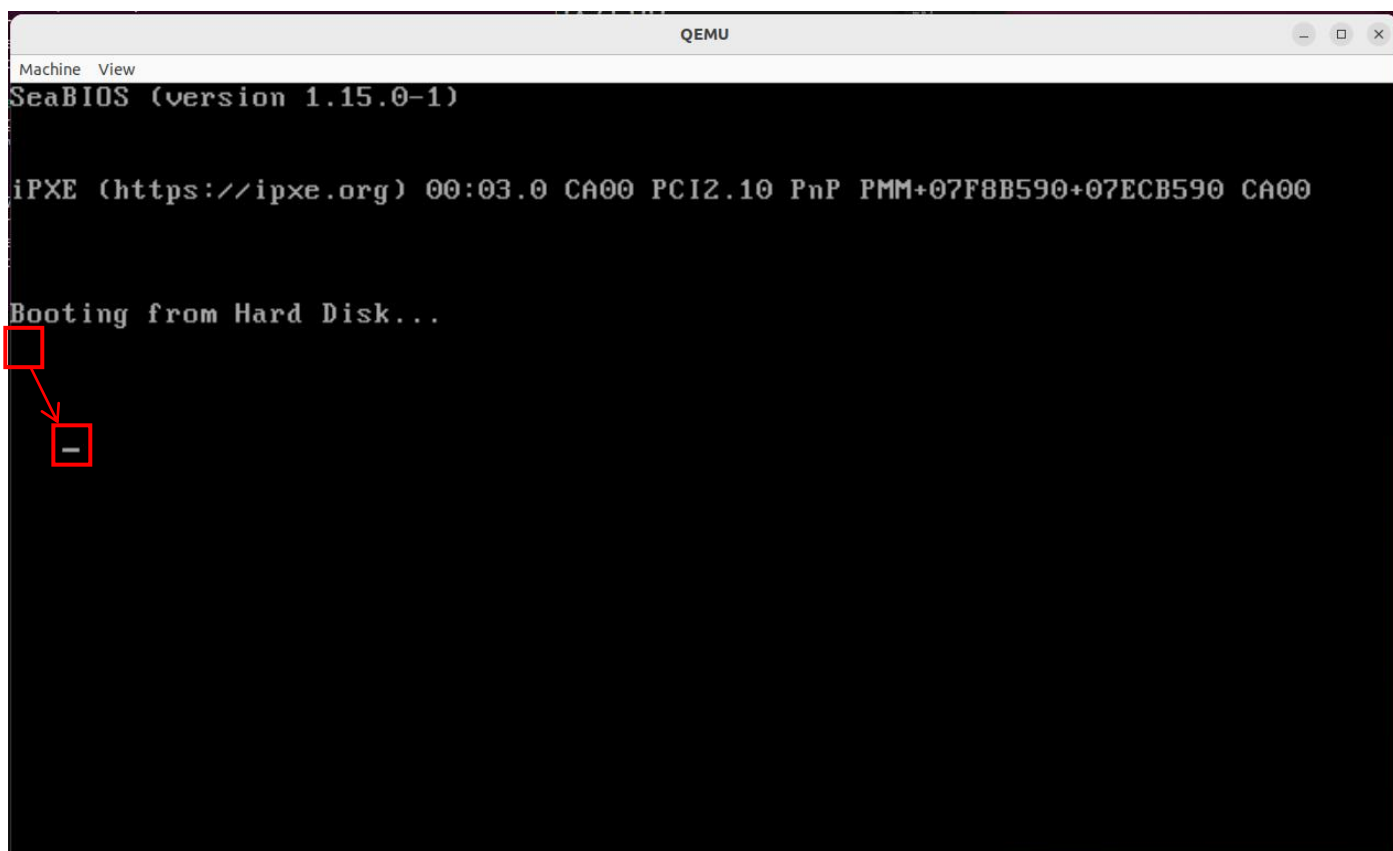
```
dd if=code2.3.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

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

系统启动后，便可以在 `qemu` 显示屏上实现键盘的输入并回显

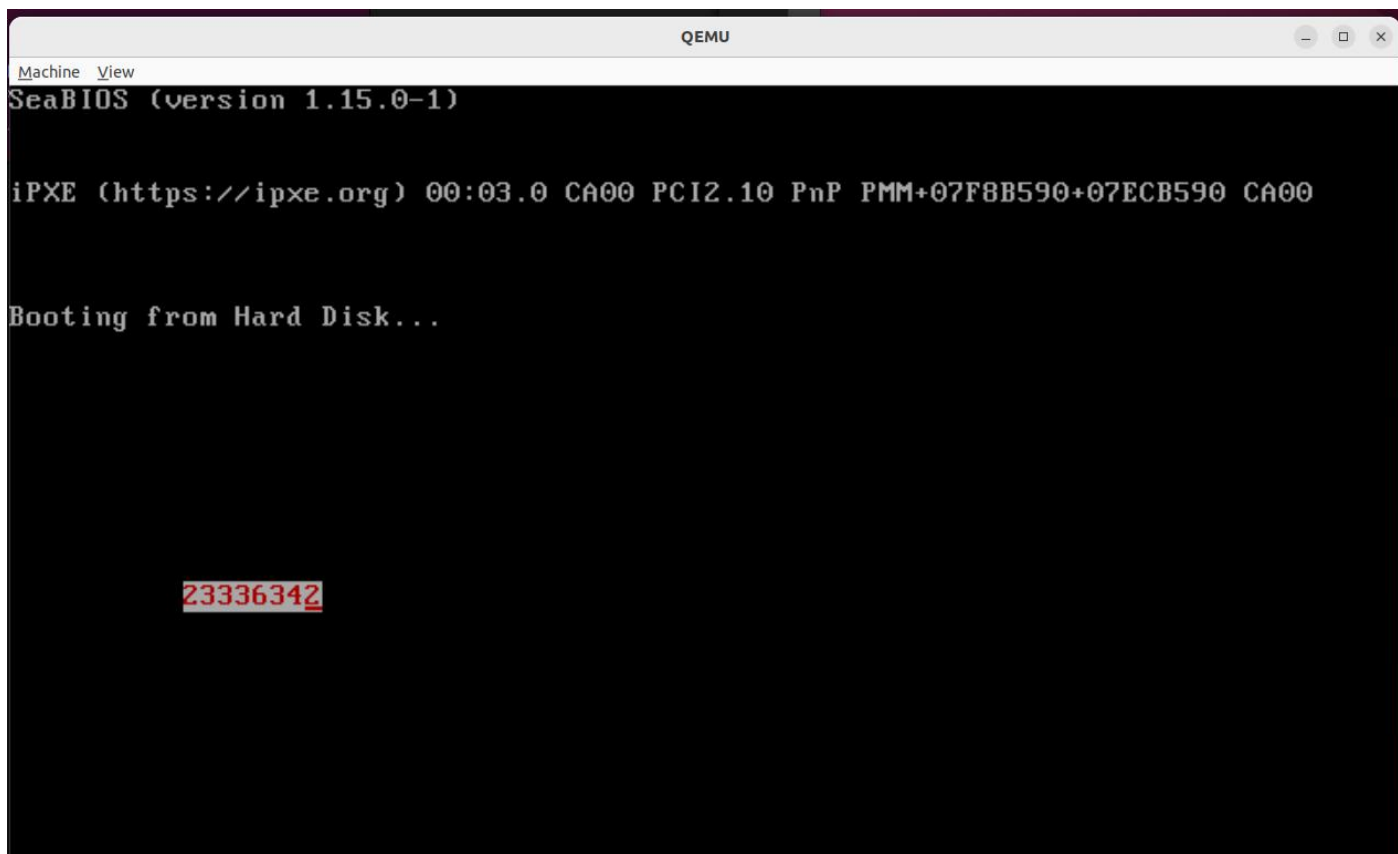
●实验结果展示：通过执行前述代码，可得下图结果。

实验 2.1



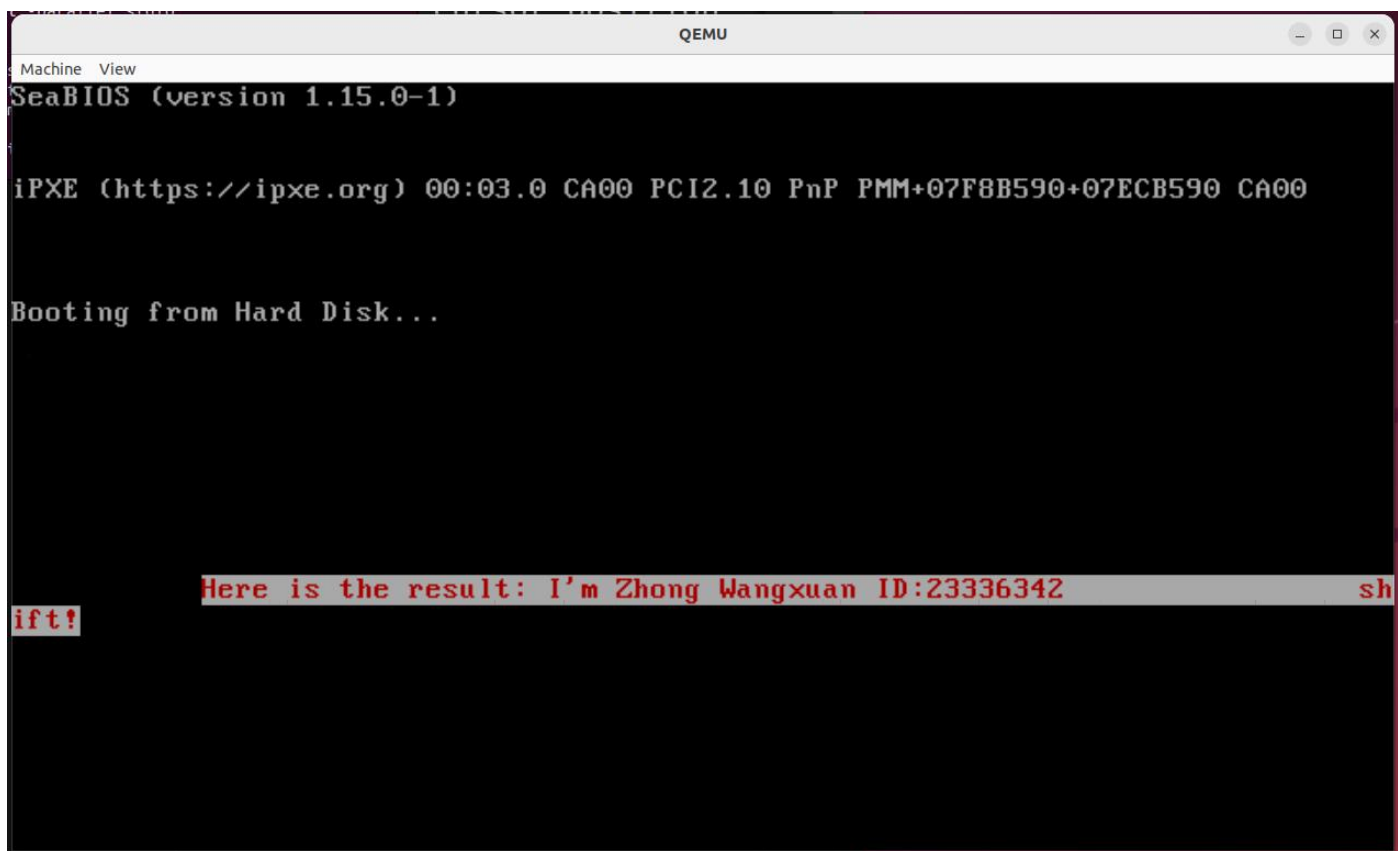
可以看到光标在行列数均加三后的位置

实验 2.2



可以看到从（16，10）开始输出白底红字的学号“23336342”

实验 2.3



我通过键盘输入了如图所示的句子，并且检验了换行功能的有效实现。

实验任务 3

●任务要求:

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`。（请尽可能多测试几组样例，以有充分的理由说明你的代码是对的）
- 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 >= 40 then
    if_flag = (a1 + 3) / 5
else if a1 >= 18 then
    if_flag = 80 - (a1 * 2)
else
    if_flag = a1 << 5
end
```

特别提醒，在使用除法的时候，你可能会遇到浮点数异常报错，请自行查询资料，了解 `idiv` 指令的用法。

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 < 25 then
    call my_random      // my_random将产生一个随机数放到eax中返回
    while_flag[a2 * 2] = 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 `your_string`。

```
your_function:
    for i = 0; your_string[i] != '\0'; ++i then
        pushad
        push (your_string[i] + 9) to stack
        call print_a_char
        pop stack
        popad
    end
    return
end
```

●思路分析:

实验 3 的任务要求将三种不同类型的伪代码通过汇编语言的形式来表示。在提供的 `assignment` 文件中的 `student.asm` 中将编写的汇编代码写入其中。（注意汇编使用 32 位寄存器）

实验 3.1

实验 3.1 要求实现的是分支逻辑。对于使用汇编语言实现分支逻辑，关键在于控制转移指令和代码标号的结合使用。使用 `cmp` 指令和 `jcondition`（有条件跳转指令的统称），便可以实现分支逻辑。对于各个分支逻辑中的数值运算，熟悉汇编算术和逻辑指令的用法即可一一实现

实验 3.2

实验 3.2 要求实现的是循环逻辑。循环逻辑的实现关键在于 `loop` 指令与控制转移指令，其中 `loop` 指令可以实现代码块循环进行，而控制转移指令主要用于判断何时结束循环，即提供书写结束循环条件的方法。

实验 3.3

实验 3.3 要求实现函数。函数的汇编实现关键在于栈的使用，因为在函数内的操作一般不应该影响到函数外寄存器的数值，但考虑到计算机的处理器中可用的寄存器是有限的，所以要通过栈指令在函数开始与结束处分别实现将寄存器的值进行入栈和出栈操作，以便函数执行后不会影响函数外的值。同时，此伪代码中同样出现了将数据入栈和出栈的操作，故熟悉栈指令很有必要。伪代码中还涉及到 `for` 循环的实现，其实现过程与实验 3.2 中 `while` 循环的实现基本一致，同样为 `loop` 指令与控制转移指令的结合使用

●实验步骤：

实验 3.1

在 `student.asm` 相应代码标号下编写实现分支逻辑的汇编代码，编写代码如下：

```
your_if:
    mov eax, [a1]
    cmp eax, 40
    jge ge40 (大于 40 跳转)
    cmp eax, 18
    jge ge18 (大于 18 跳转)
    shl eax, 5 (实现 a1 << 5)

    mov [if_flag], eax

    jmp your_while (分支逻辑结束，跳到循环逻辑)

ge40:
    add eax, 3
    mov edx, 0
    mov ecx, 5
    idiv ecx (实现(a1 + 3) / 5)
    mov [if_flag], eax

    jmp your_while (分支逻辑结束，跳到循环逻辑)

ge18:
    mov ecx, eax
    shl ecx, 1
    sub ecx, 80
    neg ecx (实现 80 - (a1 * 2))
    mov [if_flag], ecx

    jmp your_while (分支逻辑结束，跳到循环逻辑)
```

实验 3.2

在 `student.asm` 相应代码标号下编写实现循环逻辑的汇编代码，编写代码如下

```
your_while:
    mov ebx, dword[a2]
    cmp ebx, 25
    jge end_while    (跳出循环条件比较)

    call my_random (调用 my_random)

    mov esi, [while_flag]
    mov [esi + 2 * ebx], eax
    inc ebx
    mov dword[a2], ebx (实现 while_flag[a2 * 2] = eax; ++a2)
    jmp your_while

end_while: (结束循环)
```

实验 3.3

在 `student.asm` 相应代码标号下编写实现函数的汇编代码，编写代码如下

```
your_function:
    mov esi, [your_string]
    mov eax, 0 (eax 储存的为 i 的值，用于 for 循环)
loop:
    movzx ecx, byte[esi + eax]
    cmp ecx, 0
    je loopend (跳出循环条件)
    pushad
    add ecx, 9
    push ecx
    call print_a_char
    pop ecx
    popad
    inc eax (实现函数主要内容)
    jmp loop;

loopend:
    ret
```

汇编代码完成后，直接在 `terminal` 的 `assignment` 文件目录下运行指令 `make run`，得到相应的输出

●实验结果展示：通过执行前述代码，可得下图结果。

在 `terminal` 的 `assignment` 文件目录下运行指令 `make run`，得到相应的输出如下：

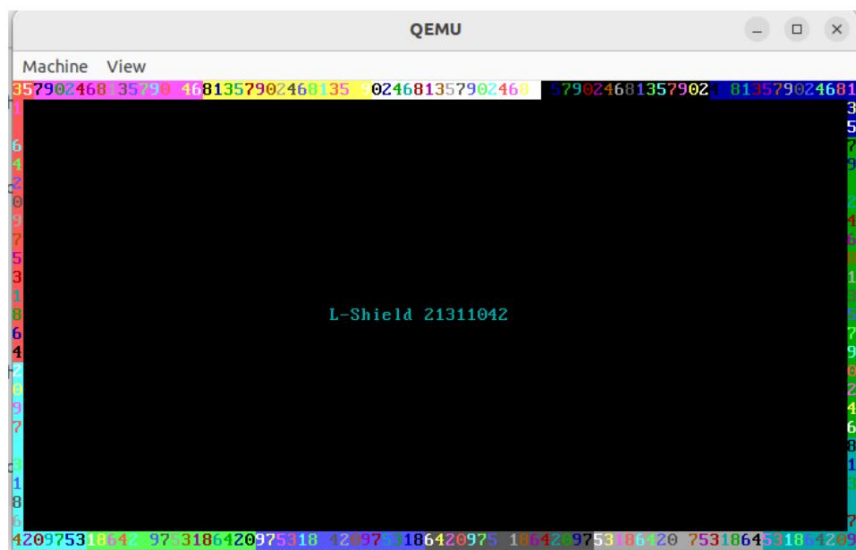
```
zhong@zhong-virtual-machine: ~/lab2/assignment
zhong@zhong-virtual-machine:~/lab2/assignment$ make run
/usr/bin/ld: student.o: warning: relocation against `while_flag' in read-only section `.text'
/usr/bin/ld: warning: creating DT_TEXTREL in a PIE
>>> begin test
>>> if test pass!
>>> while test pass!
Mr.Chen, students and TAs are the best!
```

----- 实验任务 4 -----

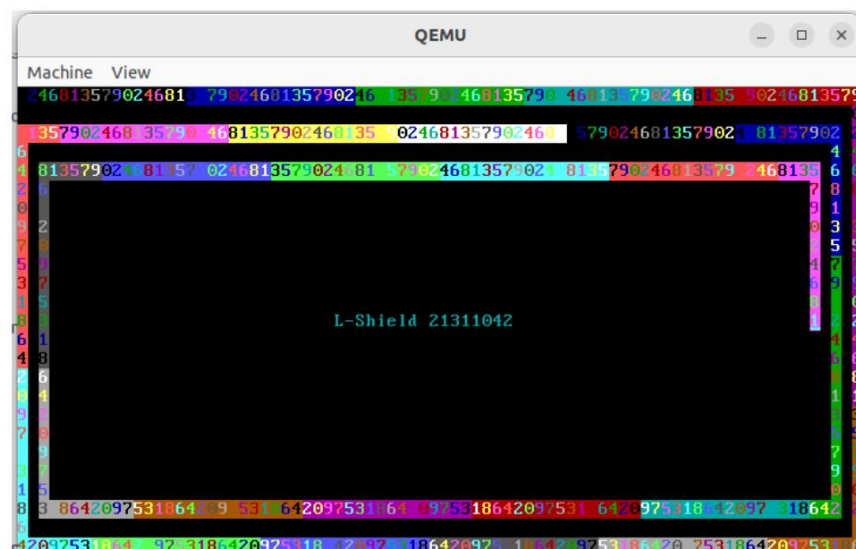
●任务要求:

Assignment 4 汇编小程序

必做: 字符回旋程序。请你用汇编代码实现一个字符回旋程序, 使其能够在qemu显示屏上面, 以不同的颜色、字符内容进行顺时针绕圈(见下图)。请注意, 对于这个绕圈的字符, 你还需要考虑他的运动速度, 不要让其运动过快(可自行查阅其他资料, 采用合适的中断来实现延时效果)



选做1: 如果你认为该任务太简单, 你还可以实现一个不断向内绕圈的字符(见下图)。请注意, 你需要设置一个期限, 不能让他无休止地向内绕圈, 导致访问非法内存。



●思路分析:

实验 4 需要实现一个字符回旋小程序, 我们主要采用实验一中在屏幕上打印字符的思想, 从 (0, 0) 出发, 每打印一个字符便按顺时针方向再打印下一个字符。在顺时针打印字符的过程中, 关键在于实现字符串打印方向的转换, 即打印字符到 (0, 79) 后, 转变从左到右的打印方向, 在 (1, 79)

开始向下打印字符。同理，我们还要在（24，79），（24，0）和（0，0）位置实现打印方向的转变。我们可以写四个判断函数，来判断打印字符是否到达了边界处，进而进行相对应的标记，来实现打印方向的转变。

对于打印字符的内容和颜色变化，不同字符内容的打印可以先定义一条字符串，通过自增一的数据取字符串长度的模运算来实现打印字符串字符的循环；对于字符不同颜色的变化，通过将一个 8 位寄存器不断自增 1 来实现，因为在显示一个字符的两个字节地址处，正好高八位用来设置字符的颜色属性。

打印字符我们通过依次使用 0xB8000~0xBFFFF 对应地址输入值来实现，利用公式：

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

来确定字符打印的位置。

选做一：

选做一要求实现一个不断向内绕圈的字符。基本思路和实验 4 基本一致，只需要在每次转换打印方向时将已经打印的一边的边界往屏幕内收缩即可。

为了防止向内圈打印字符时访问非法内存，我们在进入 main_loop 前添加一条 cmp 指令，当上边界的为 11 时，跳出程序。

●实验步骤：

在我们编写字符回旋程序前，我们还得写一部分代码，将 qemu 显示屏上原本会显示的字符给消除，并在屏幕中央处显示自己的名称首字母简写加上学号。这部分代码很好实现，只需要一个循环逻辑将 0xB8000~0xBFFFF 对应地址的值全部清零，再在屏幕中央对应地址处赋值显示名称首字母简写加上学号的值即可，此段代码一同和字符回旋程序代码编写在文件 code4 中，代码如下：

```

; 清屏模块
clear_screen:
    pusha                (保存所有通用寄存器)
    mov ax, 0xB800       (显存段地址)
    mov es, ax           (设置 ES 段寄存器)
    xor di, di           (DI = 0, 显存偏移从 0 开始)
    mov ax, 0x0720       (AH=0x07 (属性: 黑底白字), AL=0x20 (空格字符))
    mov cx, 2000         (80x25=2000 个字符)
    cld                  (清除方向标志 (递增))
    rep stosw            (重复存储 AX 到 ES:DI, 每次+2)
    popa                 (恢复所有通用寄存器)
    ret                  (返回)
```


;名称首字母简写加上学号显示函数

draw_id_name:

Pusha (保存所有通用寄存器)

xor ax, ax

mov ax, 0xb800 (显存段地址)

mov es, ax (设置 ES 段寄存器)

mov ah, 0x74 (AH=0x74 (属性: 白底红字))

mov al, '2'

mov [es:2 * 995], ax (从(12, 35)开始打印学号和姓名首字母简写)

mov al, '3'

mov [es:2 * 996], ax

mov al, '3'

mov [es:2 * 997], ax

mov al, '3'

mov [es:2 * 998], ax

mov al, '6'

mov [es:2 * 999], ax

mov al, '3'

mov [es:2 * 1000], ax

mov al, '4'

mov [es:2 * 1001], ax

mov al, '2'

mov [es:2 * 1002], ax

mov al, '-'

mov [es:2 * 1003], ax

mov al, 'Z'

mov [es:2 * 1004], ax

mov al, 'W'

mov [es:2 * 1005], ax

mov al, 'X'

mov [es:2 * 1006], ax

popa (恢复所有通用寄存器)

ret (返回)

字符回旋程序的关键代码如下所示:

start:

call clear_screen

call draw_id_name(调用清屏函数和打印学号名字函数)

xor ax, ax

mov ds, ax

mov ax, 0xb800

mov es, ax

mov byte [direction], 0 (标记打印方向)

mov byte [color], 0x1F (字符颜色属性)

mov byte [charIndex], 0 (打印字符索引)

mov byte [row], 0 (标记打印字符的行索引)

mov byte [col], 0 (标记打印字符的列索引)

mov byte [left], 0

mov byte [right], 79

mov byte [top], 0

mov byte [bottom], 24 (数据标号循环初始赋值)

main_loop:

cmp byte [direction], 0 (判断方向标号, 方向标号为 0, 向右打印)

je move_right

cmp byte [direction], 1 (判断方向标号, 方向标号为 1, 向下打印)

je move_down

cmp byte [direction], 2 (判断方向标号, 方向标号为 2, 向左打印)

je move_left

jmp move_up (否则, 向上打印)

move_right:

inc byte [col] (打印的列增加 1)

mov bl, byte[right]

cmp byte [col], bl (比较列是否到达右边界)

jb draw

mov byte [direction], 1 (到达右边界时, 切换方向标号)

jmp draw

move_down:

inc byte [row] (打印的行增加 1)

mov bl, byte[bottom]

cmp byte [row], bl (比较行是否到达下边界)

jb draw

mov byte [direction], 2 (到达下边界时, 切换方向标号)

jmp draw

move_left:

dec byte [col] (打印的列减少 1)

```

mov bl, byte[left]
cmp byte[col],bl      (比较列是否到达左边界)
ja draw
mov byte [direction], 3  (到达左边界时, 切换方向标号)
jmp draw

```

move_up:

```

dec byte [row]      (打印的行减少 1)
mov bl, byte[top]
cmp byte[row],bl    (比较列是否到达上边界)
ja draw
mov byte [direction], 0  (到达左边界时, 切换方向标号)

```

draw:

```

mov ax, [charIndex]
mov bl, 8
div bl
mov [charIndex],ah  (用模运算实现字符串的循环打印)

call draw_char
inc byte[color]      (字符颜色属性变换)
inc byte[charIndex]  (字符索引自加 1, 打印下一字符)
mov cx, 0x0000
mov dx, 0x5fff
mov ah, 0x86
int 0x15             (运用 int 15h 实模式中断的 AH = 86h 功能号, 使用延迟功能, 可以
                     控制字符的打印速度)
jmp main_loop        (跳回主循环)

```

draw_char: (打印字符函数)

```

pusha
xor ax, ax          (ax = 0)
mov al, [row]
mov bx, 80
mul bx
add al, [col]
adc ah, 0
shl ax, 1
mov di, ax          (通过行列计算储存字符的地址)

```

```

mov bx, [charIndex]
mov al, [chars + bx]  (储存打印字符)

```

```

mov ah, [color]      (设置字符颜色属性)
mov [es:di], ax
popa
ret

```

随后我们先将代码转化为二进制文件：

```
nasm -f bin code4 -o code4.bin
```

再将二进制文件放入 `hd.img` 中，使用 `qemu` 运行系统

```
dd if=code4.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

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

系统在 `qemu` 上运行后，可以看到字符回旋程序正常运行

选做 1：

选做 1 的代码编写在 `code4.1` 中，其中更新的代码段为

```
move_right:
    inc byte [col]
    mov dl, byte[right]
    cmp byte [col],dl
    jb draw
    mov byte [direction], 1
    inc byte [top]      (上边界加一)
    jmp draw

move_down:
    inc byte [row]
    mov dl, byte[bottom]
    cmp byte [row],dl
    jb draw
    mov byte [direction], 2
    dec byte [right]    (右边界减一)
    jmp draw

move_left:
    dec byte [col]
    mov dl, byte[left]
    cmp byte [col],dl
    ja draw
    mov byte [direction], 3
    dec byte [bottom]   (下边界减一)
    jmp draw

move_up:
    dec byte [row]
    mov dl, byte[top]
    cmp byte [row],dl
    ja draw
    mov byte [direction], 0
    inc byte [left]     (左边界加一)

draw:
    mov ax, [charIndex]
```

```

mov bl, 8
    div bl
    mov [charIndex],ah
    call draw_char
    inc byte[color]
    inc byte[charIndex]

    mov cx, 0x0000
    mov dx, 0x0fff
    mov ah, 0x86
    int 0x15

    mov al, [top]
    cmp al, 11
    je end_loop (跳出循环判断)

```

end_loop 代码编号在代码段的结尾处

```

end_loop:

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

```

随后我们先将代码转化为二进制文件：

```
nasm -f bin code4.1 -o code4.1.bin
```

再将二进制文件放入 hd.img 中，使用 qemu 运行系统

```
dd if=code4.1.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

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

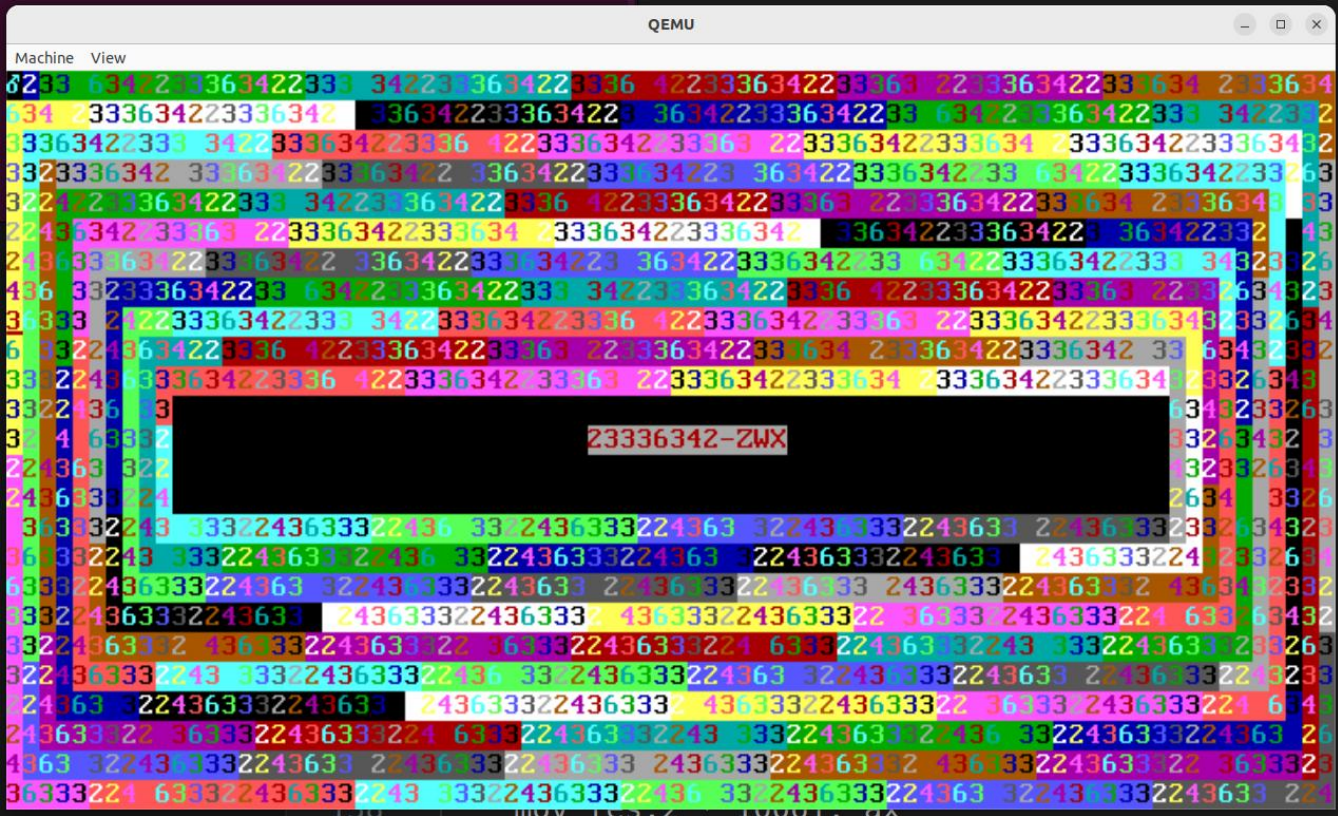
系统在 qemu 上运行后，可以看到不断向内绕圈的字符程序正常运行

●实验结果展示：通过执行前述代码，可得下图结果。

实验 4



选做 1



Section 5 实验总结与心得体会

1、通过本次的实验，我对计算机系统的启动流程，如何利用汇编语言编写程序加载 MBR 实现在 qemu 上打印字符，如何运用实模式中断实现光标位置的获取和移动、打印字符和键盘输入并回显，如何将伪代码用汇编语言编写以及如何编写一个汇编小程序有了更深的理解并掌握了更多有关系统操作的知识。

2、在实模式中断的实验 2.3 中，刚开始对于如何实现键盘中断一直迷惑不解，在 deepseek 的帮助以及 csdn 上相关内容的帮助下，我得知了与键盘中断有关的 int 16h 实模式中断，并顺利实现了代码。

3、实验 3 中在初步测试过程中一直显示编译错误，在使用 gdb 对程序进行多次调试后，终于发现是因为在 your_function 部分我进行了两次 popad 的缘故，导致汇编出错。教训告诉我们要多利用 gdb 对自己的程序进行调试才能更好的发现错误。

4、本次实验让我认识了更多有关计算机系统汇编方面的知识，进一步激发了我探索计算机系统开发的兴趣。

Section 6 对实验的改进建议和意见

实验 4 的汇编小程序希望能够有相应的编写代码思路的指导

Section 7 附录：参考资料清单

[lab2 · apshuang/SYSU-2025-Spring-Operating-System - 码云 - 开源中国](#)