



中山大學
SUN YAT-SEN UNIVERSITY

本科生实验报告

实验课程: 操作系统原理实验

任课教师: 刘宁

实验题目: 实模式和保护模式下的OS启动

专业名称: 信息与计算科学

学生姓名: 罗弘杰

学生学号: 22336173

实验地点: 实验中心D503

实验时间: 2024/3/15

Section 1 实验概述

在第二章中, 同学们将会学习到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, 同学们能自己编写程序, 然后让计算机在启动后加载运行, 以此增进同学们对计算机启动过程的理解, 为后面编写操作系统加载程序奠定基础。同时, 同学们将学习如何使用gdb来调试程序的基本方法。

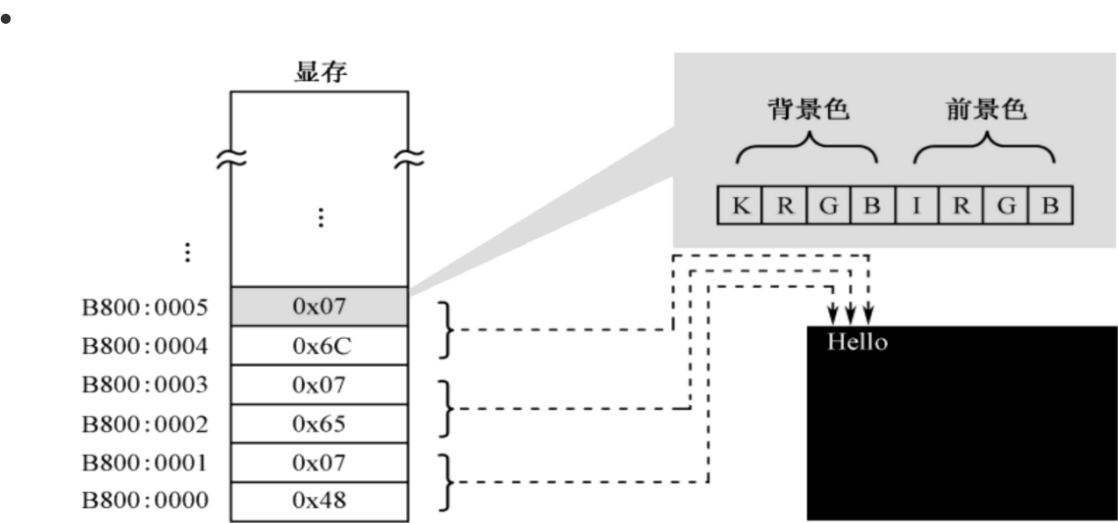
- 独立完成实验5个部份环境配置、编译Linux内核、Qemu启动内核并开启远程调试、制作Initramfs和编译并启动Busybox。
- 编写实验报告、结合实验过程来谈谈你完成实验的思路和结果, 最后需要提供实验的5个部份的程序运行截屏来证明你完成了实验。
- 实验不限语言, C/C++/Rust都可以。
- 实验不限平台, Windows、Linux和MacOS等都可以。
- 实验不限CPU, ARM/Intel/Risc-V都可以。。

Section 2 预备知识与实验环境

该节总结实验需要用到的基本知识, 以及主机型号、代码编译工具、重要三方库的版本号信息等。

- 预备知识: x86汇编语言程序设计、Linux系统命令行工具, qemu虚拟机模拟, gdb调试工具

- IA-32处理器将显示矩阵映射到内存地址0xB8000~0xBFFFF处，这段地址称为显存地址。在文本模式下，控制器的最小可控制单位为字符。每一个显示字符自上而下，从左到右依次使用0xB8000~0xBFFFF中的两个字节表示。在这两个字节中，低字节表示显示的字符，高字节表示字符的颜色属性，如下所示。



字符的颜色属性的字节高4位表示背景色，低4位表示前景色，如下所示。

R	G	B	背景色	前景色	
			K=0 时不闪烁，K=1 时闪烁	I=0	I=1
0	0	0	黑	黑	灰
0	0	1	蓝	蓝	浅蓝
0	1	0	绿	绿	浅绿
0	1	1	青	青	浅青
1	0	0	红	红	浅红
1	0	1	品（洋）红	品（洋）红	浅品（洋）红
1	1	0	棕	棕	黄
1	1	1	白	白	亮白

在上面的对显示矩阵的点的描述中，我们使用的是二维的点，但对应到显存是一维的，因此我们需要进行维度的转换。显示矩阵的点 (x, y) 对应到显存的起始位置如下所示。

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

其中， (x, y) 表示第 x 行第 y 列，公式中乘2的原因是每个显示字符使用两个字节表示。

我们来看个具体例子，在上面输出“Hello”的图中，我们在第0行第1列输出了背景色为黑色，前景色为白色的字符e，那么对应到显示矩阵的点是 $(0, 1)$ ，此时显存的起始位置如下所示。

$$\text{显存起始位置} = 0xB8000 + 2 \cdot (80 \cdot 0 + 1) = 0xB8002$$

- 实验环境：
 - 虚拟机版本/处理器型号：ubuntu-18.0.4，阿里云服务器 通用cpu
 - 代码编辑环境：vim
 - 代码编译工具：gcc, nasm
 - 重要三方库信息：无

Section 3 实验任务

该节描述需要完成的几个实验任务，即重述实验题目的总要求，建议使用项目编号分点阐述。详细要求可在下一节【实验步骤与实验结果】中列出。

实验任务1：

学习x86汇编基础，理解实模式下计算机启动的过程，复现“操作系统的启动Hello World--编写MBR”部分的实验。

实验任务2：

探索实模式中中断，利用中断实现光标移动和在光标处打印字符等等

实验任务3：

汇编代码实现分支逻辑，循环逻辑，以及函数的实现

实验任务4：

实现一个字符弹射程序

Section 4 实验步骤与实验结果

该节描述每个实验任务的具体的完成过程，包括思路分析、代码实现与执行、结果展示三个部分，实验任务之间的划分应当清晰明了，实验思路分析做到有逻辑、有条理。

----- 实验任务1-----

任务要求：

编写汇编代码，编译后加入MBR中，启动qemu,读取MBR,显示“hello world”

思路分析：

参考实验资料给出的代码，将helloworld字符加载到0xB8000~0xBFFFF的显存内，注意每个字符由两个字节表示，低字节表示字符的内容，高字节表示颜色（前四位表示背景色，后四位表示前景色）。结尾加入一段死循环，让字符串恒定显示。注意物理地址的计算公式：物理地址=gs<<4+2*1=0xB800<<4+2*1=0xB8002；在开始的时候要给gs赋值到0x800再左移四位就到达显存位置。

```
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; 初始化显存的段寄存器
mov gs, ax

mov ah, 0x01 ; 蓝色 ; 规定颜色
mov al, 'H'
mov [gs:2 * 0], ax
```

```
mov al, 'e'
mov [gs:2 * 1], ax

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

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

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

mov al, ' '
mov [gs:2 * 5], ax

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

mov al, 'o'
mov [gs:2 * 7], ax

mov al, 'r'
mov [gs:2 * 8], ax

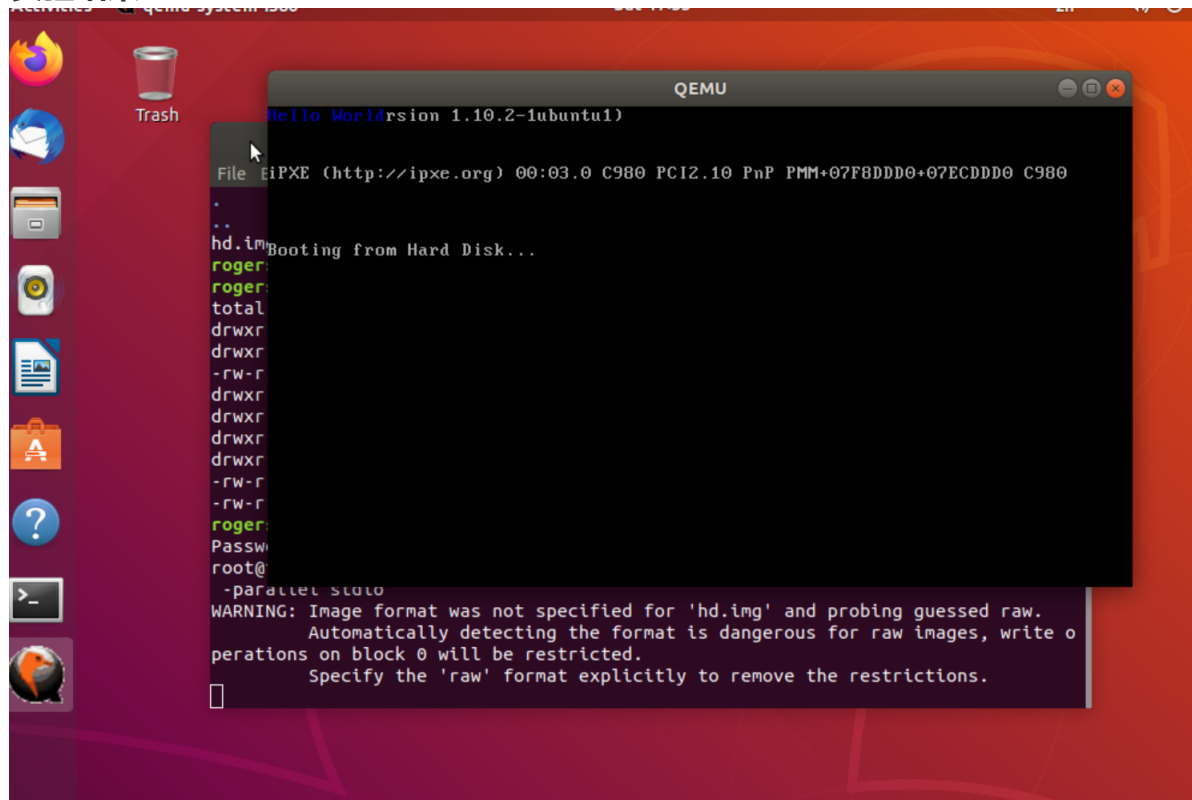
mov al, '['
mov [gs:2 * 9], ax

mov al, 'd'
mov [gs:2 * 10], ax

jmp $ ; 死循环

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

实验结果：



实验任务2

任务要求：

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. 请修改Hello World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。
3. 在1和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。

思路分析：

根据实验资料给出的关于实模式下中断的功能号的功能，包括了光标位置的获取，光标的移动和在光标处打印字符。具体地：应该先设置或者获取光标的初始位置，然后根据打印的字符数来设计光标的移动，注意显示屏上横纵坐标上限分别为[0,24],[0,79]；在必要的时候需要安排光标换行。

实验步骤：

(1) 利用中断实现光标的位置获取和光标的移动

代码展示：

```
org 0x7c00
[bits 16]
; get cursor position
mov ah, 0x03; get cursor position
mov bh, 0; page number
int 10h
```

```

; set cursor position
mov ah, 0x02; set cursor position
mov bh, 0; page number
inc dh; increment row
inc dl; increment column
int 10h

jmp $

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

```

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`

1. 获取光标的功能号为 `03h`，将获取的行号，列号写入 `dl, dh` 寄存器；
2. 移动字符的功能号为 `02h`，在这里我将行号和列号分别加一，即向右下角移动一次；

成果展示：

(2) 请修改Hello World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。

```

my_id db "22336173_LHJ";
org 0x7c00
[bits 16]
; print "22336173" on the screen

set_cursor_first:
    mov bx, 0
    mov dh, 5; row5
    mov dl, 9; col8
    mov ah, 2
    int 10h
set_color:
    mov bl, 0x40 ; back red front black
    mov ecx, 12; loop 12 times
    mov si, my_id

print_loop:
    mov al, [si]; read a char from si address
    push cx; save the loop times
    mov cx, 0x0001; set number of char to print
    mov ah, 9
    int 10h
    call cursor_inc ; cursor movement
    pop cx
    add si, 1; next char
    loop print_loop

    jmp $

cursor_inc:
    inc dl

```

```

    cmp dl, 80
    jne add_cursor
    mov dl, 0; 换行
    inc dh
add_cursor:
    mov ah, 2; set cursor +1
    int 10h
    ret

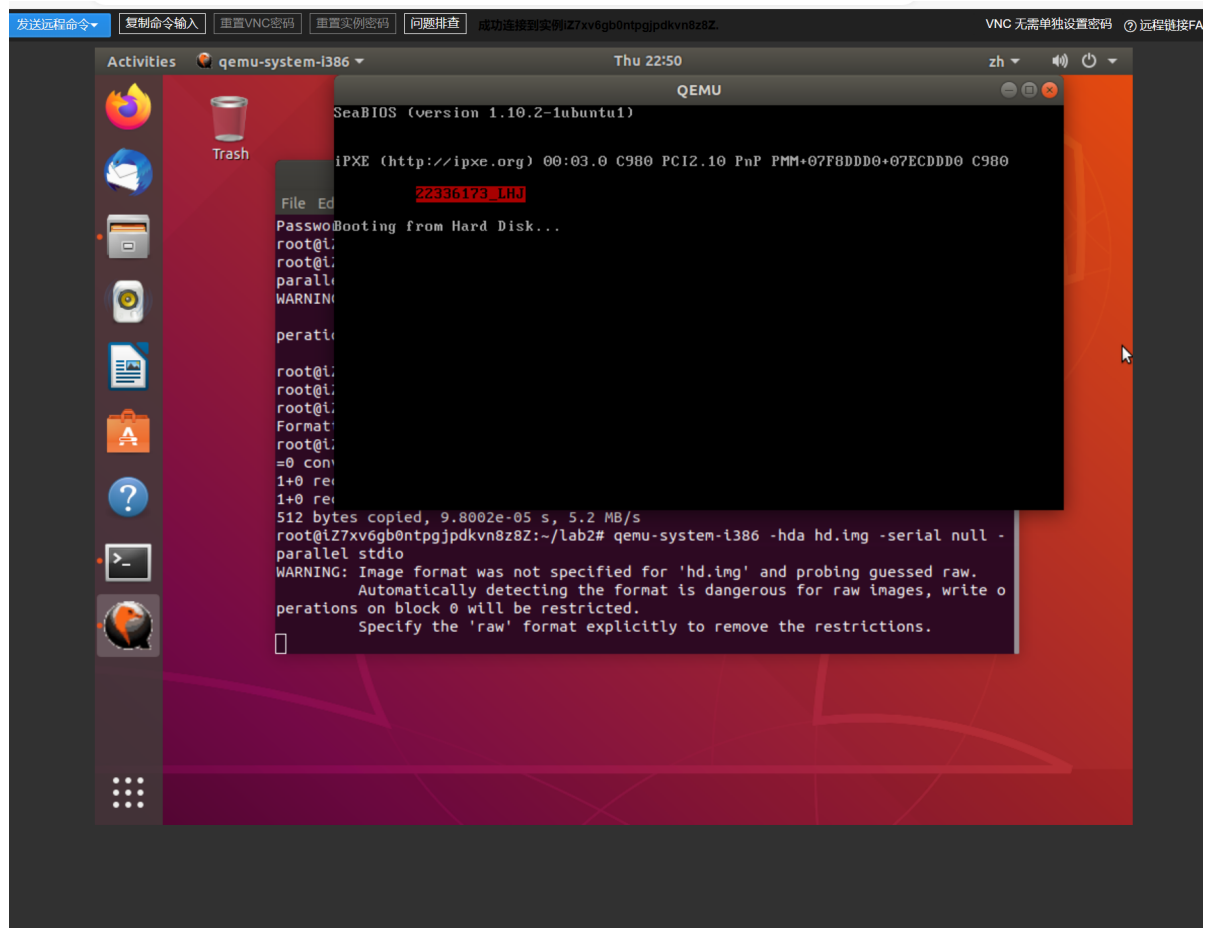
times 510-($-$$) db 0
db 0x55, 0xaa

```

代码分析:

使用循环输出的思路，首先将我的学号放在字符串数据段，然后将地址放入si寄存器，循环次数放入ecx寄存器，每次输出完以后，ecx-1，si加一以打印下一个字符；注意光标移动需要添加换行条件机制。

成果展示:



(3) 在1和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。

代码展示:

```

org 0x7c00
[bits 16]

mov dh, 5      ; row 5
mov dl, 9      ; col 8

```

```

kbIO:
mov ah, 0x00 ; function 1 of int 0x16 (keyboard input)
int 0x16
or al, 0x00 ; Test if the ASCII code of the key pressed is zero
jnz print_key ; If not zero, jump to print_key
jmp kbIO ; If zero, continue waiting for input

print_key:
call cursor_inc

mov ah, 0x09 ; print character
mov bl, 0x40 ; color
mov cx, 0x01 ; number of characters
int 0x10
jmp kbIO ; repeat

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

```

代码分析:

考虑使用中断号16h, 功能号00h, 资料如下:

2、INT 16H (键盘I/O中断)

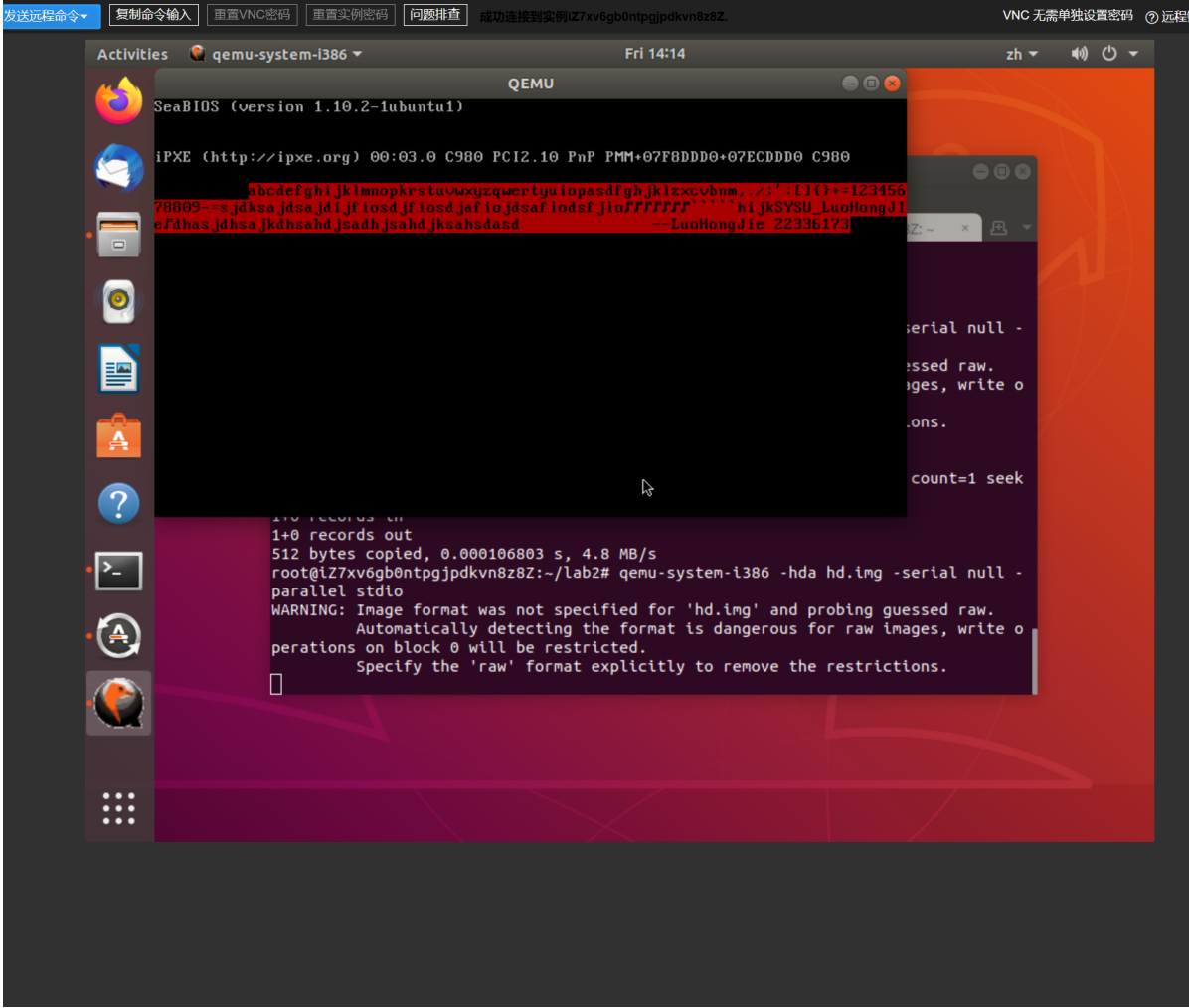
AH=0: 从键盘读入ASCII字符, 放在AL中。

AH=1: 测试有无键被按下。ZF=0, 表示按过任意键, 并在AL中获得该键的ASCII码。ZF=1, 未按过键。

AH=2: 读取特殊功能键的状态至AL中。

每次都测试键盘有无输入, 通过al寄存器与0相或来判断, 无输入则循环等待输入, 否则就进入输出模块, 依然要考虑换行机制, 输出完以后, 返回等待输出。

成果展示:



实验任务3

任务要求：

17. 汇编代码的编写

- 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。

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
```

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. 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串 `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
```

思路分析：

本题要求我们熟悉x86汇编的条件判断和循环和函数的结构。条件： `cmp, je, jne, jz, jnz` 等等；循环：使用 `cx` 寄存器存储循环次数，`loop` 指令更新循环次数；函数： `call, ret` 注意需要使用栈寄存器存储原函数环境，可以使用 `pushad, popad` 指令来快速保存和回复环境。

代码分析：

```
[bits 32]
#include "head.include"

mov eax, [a1]
mov ebx, [a2]
your_if: ;条件循环
    cmp eax, 12
    jl lt12 ; if eax < 12 then jump to lt12
    cmp eax, 24
    jl lt24 ; if eax < 24 then jump to lt24
    shl eax, 4; a1*=16
```

```

    mov [if_flag], eax; change if_flag to a1
    jmp your_while

lt24:
    mov ecx, eax ; ecx = eax
    sub ecx, 24 ; ecx = ecx - 24
    neg ecx ; ecx = -ecx
    imul ecx, eax; ecx = ecx * eax
    mov [if_flag], ecx
    jmp your_while

lt12:
    sar eax, 1 ; a1/=2
    inc eax ; a1++
    mov [if_flag], eax
    jmp your_while

your_while:    ;循环
    cmp dword[a2], 12
    jl lt_12; if ebx < 12 then jump to lt12
    call my_random
    mov ebx, [a2]
    mov dword[ecx + ebx - 12], eax
    dec dword[a2]
    jmp your_while

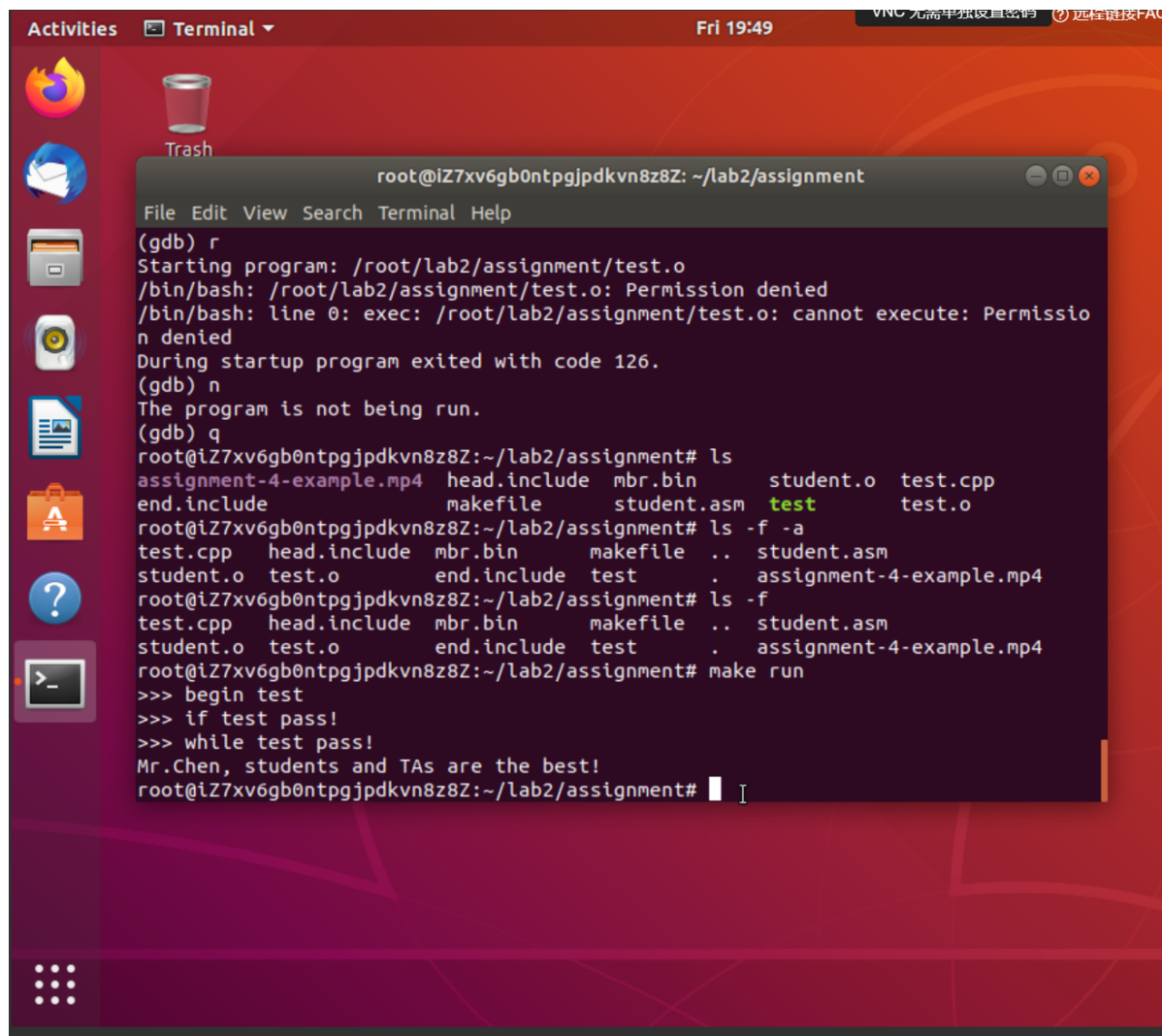
lt_12:
#include "end.include"

your_function:    ;函数
    pushad
    mov eax, 0
loop:
    mov ecx, [your_string]
    cmp byte[ecx+eax], 0
    je loopend
    pushad
    mov ebx, dword[ecx+eax]
    push ebx
    call print_a_char
    pop ebx
    popad
    add eax, 1;
    jmp loop;

loopend:
    popad
    ret

```

结果展示:



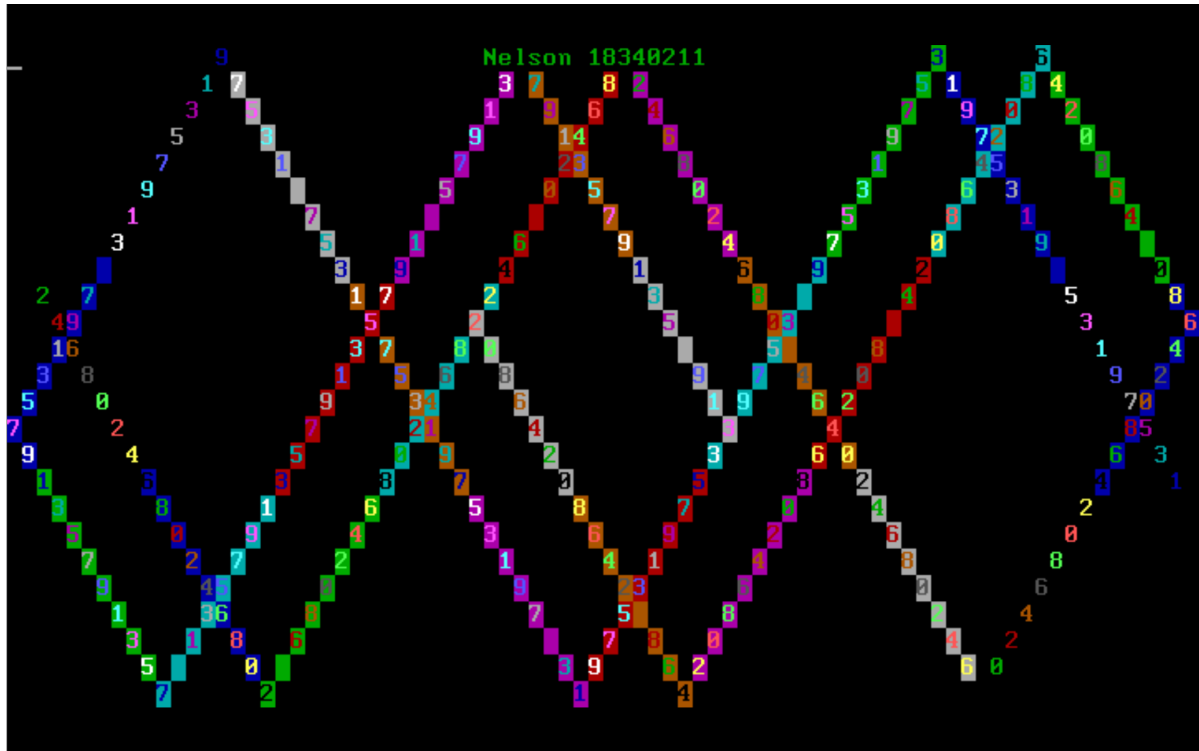
```
root@iZ7xv6gb0ntpgjpdv8z8Z: ~/lab2/assignment
File Edit View Search Terminal Help
(gdb) r
Starting program: /root/lab2/assignment/test.o
/bin/bash: /root/lab2/assignment/test.o: Permission denied
/bin/bash: line 0: exec: /root/lab2/assignment/test.o: cannot execute: Permission denied
During startup program exited with code 126.
(gdb) n
The program is not being run.
(gdb) q
root@iZ7xv6gb0ntpgjpdv8z8Z:~/lab2/assignment# ls
assignment-4-example.mp4  head.include  mbr.bin      student.o  test.cpp
end.include               makefile      student.asm  test       test.o
root@iZ7xv6gb0ntpgjpdv8z8Z:~/lab2/assignment# ls -f -a
test.cpp  head.include  mbr.bin  makefile  ..  student.asm
student.o  test.o        end.include  test  .  assignment-4-example.mp4
root@iZ7xv6gb0ntpgjpdv8z8Z:~/lab2/assignment# ls -f
test.cpp  head.include  mbr.bin  makefile  ..  student.asm
student.o  test.o        end.include  test  .  assignment-4-example.mp4
root@iZ7xv6gb0ntpgjpdv8z8Z:~/lab2/assignment# make run
>>> begin test
>>> if test pass!
>>> while test pass!
Mr.Chen, students and TAs are the best!
root@iZ7xv6gb0ntpgjpdv8z8Z:~/lab2/assignment#
```

通过了测试。

----- 实验任务4 -----

任务要求:

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。



思路分析：

依然使用存储显存的办法，设置初始坐标为（2，0）。然后设置4个移动函数，分别对应左上，左下，右下，右上；在每个函数中都设置碰壁判断函数（x:-1和25，y:-1和80），一旦碰撞就触发反弹函数。每次移动以后加入字符添加函数，加入颜色变化函数，每次存储一个字符，循环存储的是字符串“22336173_LHJ”

代码分析：

```
org 0x7c00
; 初始化规定变量
_DR equ 1
_UR equ 2
_UL equ 3
_DL equ 4 ; 这四个分别是四个方向的代号
delay equ 2000
ddelay equ 100 ; 循环延迟的量 总共是2000*100个循环以延迟字符输出的间隔

START:
    mov ax, cs
    mov es, ax
    mov ds, ax
    mov ax, 0b800h
    mov es, ax
    mov si, 0
    mov di, 0

PRINT1:
    mov bx, name
    mov al, [bx+si]
    cmp al, 0
```

```

    jz Loop
    mov bx, 52
    mov byte[es:bx+di], al
    mov byte[es:bx+di+1], 1
    inc si
    add di, 2
    jmp PRINT1

    mov si, name
Loop:
    dec word[count]
    jnz Loop

    mov word[count], delay
    dec word[dcount]
    jnz Loop

    mov word[count], delay
    mov word[dcount], ddelay

    mov al, 1
    cmp al, byte[rdu1]
    je D_R                ;判断当前移动方向，下同

    mov al, 2
    cmp al, byte[rdu1]
    je U_R

    mov al, 3
    cmp al, byte[rdu1]
    je U_L

    mov al, 4
    cmp al, byte[rdu1]
    je D_L

    jmp $

D_R:                ;进入判断碰壁函数，若碰壁则相应的改变方向，并改变当前的移动方向
    inc byte[x]
    inc byte[y]

    mov bh, byte[x]
    mov bl, 25
    cmp bh, bl
    je dr2ur

    mov bh, byte[y]
    mov bl, 80
    cmp bh, bl
    je dr2dl

    jmp display        ;结束后进入输出阶段

dr2ur:
    mov byte[x], 23

```

```
mov byte[rdu1], _UR
jmp display
```

dr2dl:

```
mov byte[y], 78
mov byte[rdu1], _DL
jmp display
```

D_L:

```
inc byte[x]
dec byte[y]
```

```
mov bh, byte[x]
mov bl, 25
cmp bh, bl
je dl2ul
```

```
mov bh, byte[y]
mov bl, -1
cmp bh, bl
je dl2dr
```

```
jmp display
```

dl2ul:

```
mov byte[x], 23
mov byte[rdu1], _UL
jmp display
```

dl2dr:

```
mov byte[y], 1
mov byte[rdu1], _DR
jmp display
```

U_R:

```
dec byte[x]
inc byte[y]
```

```
mov bh, byte[x]
mov bl, -1
cmp bh, bl
je ur2dr
```

```
mov bh, byte[y]
mov bl, 80
cmp bh, bl
je ur2ul
```

```
jmp display
```

ur2dr:

```
mov byte[x], 1
mov byte[rdu1], _DR
jmp display
```

ur2ul:

```
mov byte[y], 78
```

```

    mov byte[rdu1], _UL
    jmp display

U_L:
    dec byte[x]
    dec byte[y]

    mov bh, byte[x]
    mov bl, -1
    cmp bh, bl
    je u12d1

    mov bh, byte[y]
    mov bl, -1
    cmp bh, bl
    je u12ur

    jmp display

u12d1:
    mov byte[x], 1
    mov byte[rdu1], _DL
    jmp display

u12ur:
    mov byte[y], 1
    mov byte[rdu1], _UR
    jmp display

display:
    cmp byte[num], 12 ;12是我的字符串的个数
    je swap
next:
    xor ax, ax
    mov ax, word[x] ; 获取横纵坐标，根据显存计算公式： $es \leftarrow 4 + 2 * (x * 80 + y)$  来获得当前存储的位
置
    mov bx, 80
    mul bx
    add ax, word[y]
    mov bx, 2
    mul bx
    mov bx, ax
    mov ah, byte[color];
    mov al, byte[si] ;si存储的是当前应该输出的字符的位置
    mov [es:bx], ax
    inc si ;准备输出下一个字符

change_color: ;颜色变化，恒定背景为黑色，前景不断变化颜色
    inc byte[color]
    cmp byte[color], 0x0f
    jnz Loop
    mov byte[color], 0x02
    jmp Loop

swap: ;若输出到结尾则变回字符串的头部
    sub si, 12
    jmp next

```



```

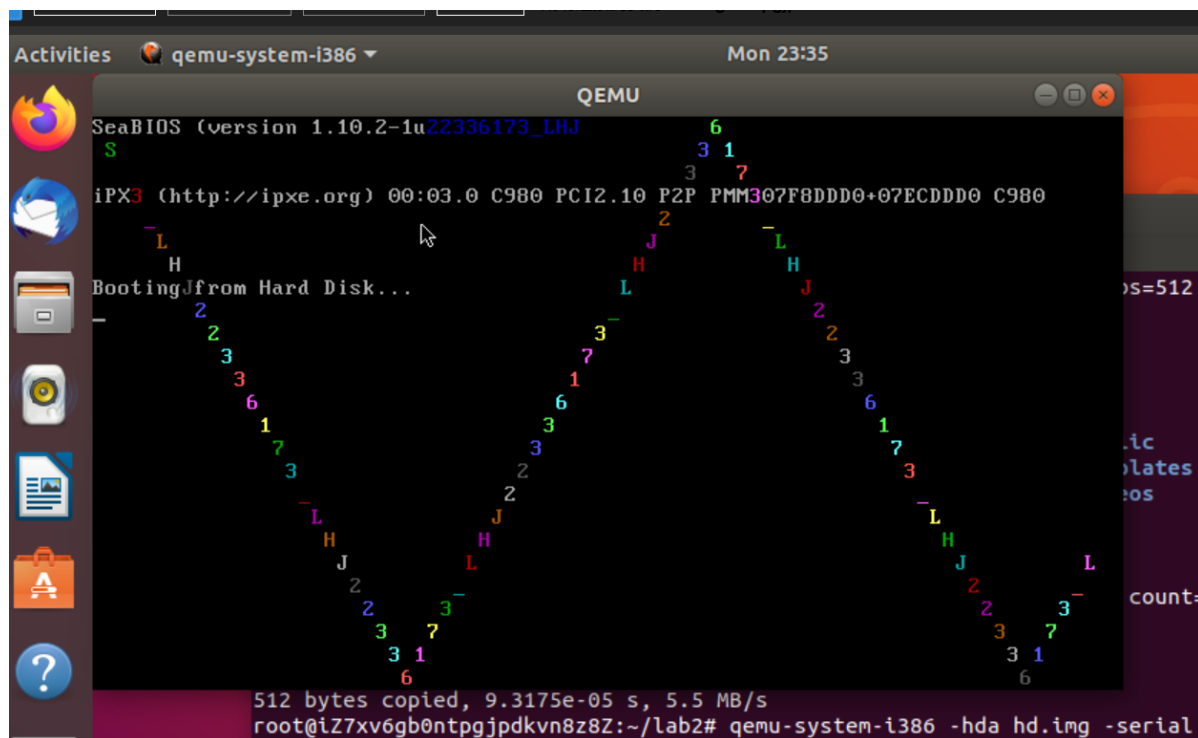
end:
    jmp $

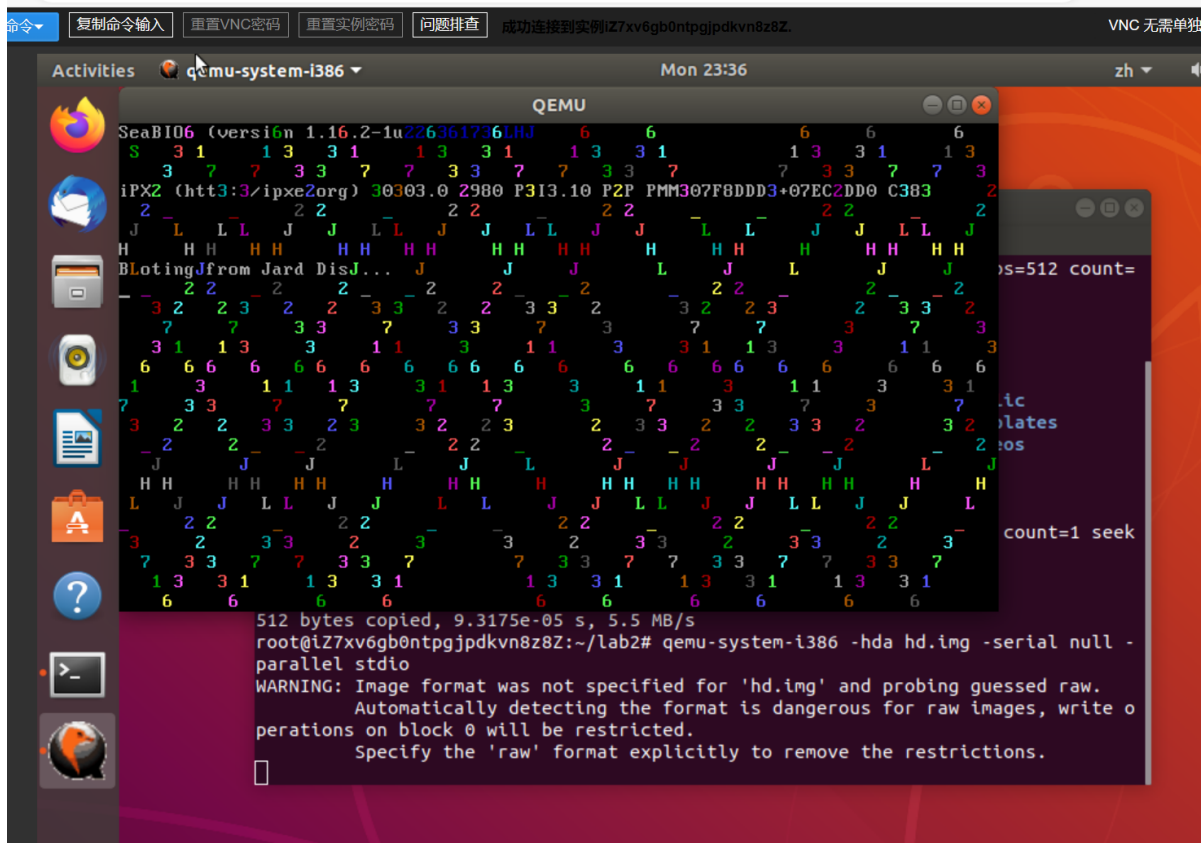
count dw delay    ; count to loop for latency
dcount dw ddelay
rdul db _DR    ; current movement
color db 0x02    ; initial color
x dw 0    ; position code
y dw 0
name db '22336173_LHJ',0    ; my id
num db 0    ; time that have printed

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

```

成果展示:





实验结果展示：通过执行前述代码，可得下图结果：

Section 5 实验总结与心得体会

棘手的问题：linux内核版本不匹配，需要更新比较新的内核版本才可以正常进行qemu和gdb调试；

心得体会：初步认识到linux系统的启动过程，内核（kernel）初始化结束后，initramfs作为临时根文件系统被挂载，helloworld是最简单的一类，还没有实现根文件的系统，mybusybox中基于已有程序给出了根文件系统，使得操作系统可以正常启动。

Section 6 对实验的改进建议和意见

1. 实验中的推荐使用linux-3的版本是否和最新的gdb和qemu平台不兼容？
2. 实验中的原理还需要阐释的更明白一些，很多命令行原理和Linux系统启动知识需要上网搜索，对于一些同学可能是不友好的。

Section 7 附录：参考资料清单

本节为可选章节，可以列出自己在实验过程中的一些重要参考书籍、博客网站等等，为将来的实验提供帮助。

[linux启动流程——initrd和initramfs initrd-switch-root" "initramfs-CSDN博客](#)

Section 8 附录：代码清单

【实验任务3】完整的c程序如下：