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

实验名称：MBR、实模式中断、汇编

专业名称：计算机科学与技术

学生姓名：数据删除

学生学号：数据删除

实验成绩：

报告时间：2025 年 2 月 26 日

1. 实验要求

Assignment 1 MBR

1. 复现 example 1。说说你是怎么做的，并将结果截图。
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

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 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。
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。说说你是怎么做的，并将结果截图。

Assignment 3 汇编

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

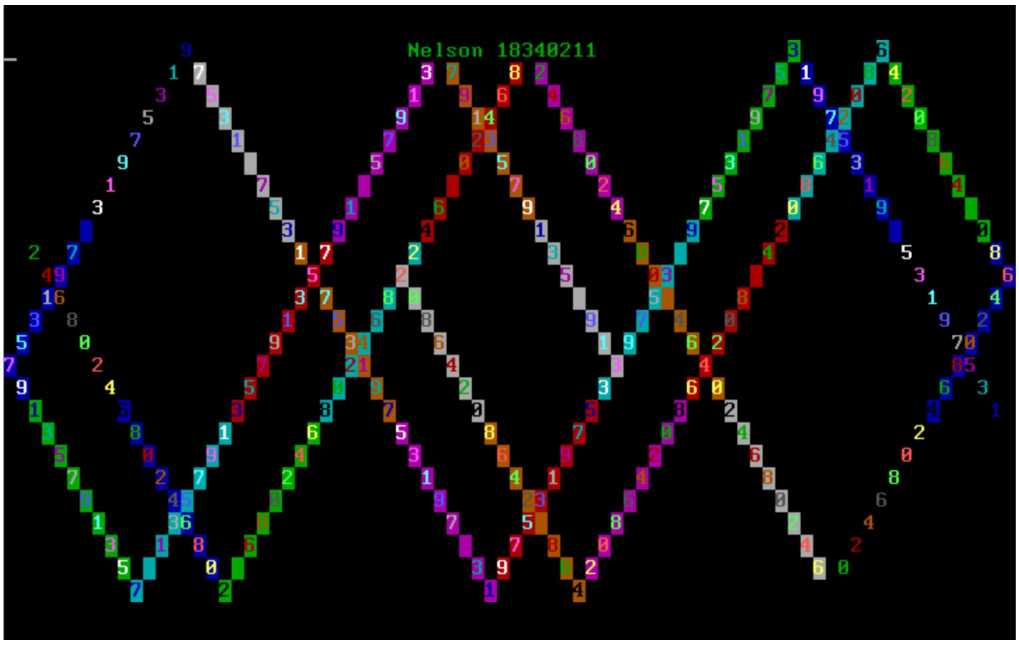
```
while a2 ⩾ 12 then
    call my_random          // my_random将产生一个随机数放到eax中返回
    while_flag[a2 - 12] = eax
    --a2
end
```

3. 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)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



2. 实验过程

Assignment 1 MBR

考虑用和思考题最后一题一样的用循环访问字符串数组的方式来显示字符。思考题在访问[si + string]的段号没有指定，只需要在代码最上方加上 org 0x7c00 或者手动指定一下 ds 为 0x07c0 即可

显示学号只需要改变字符串数组即可，要从(12,12)开始输出，只需在输出的显存地址上整体加一个 2*(12*80+12)=1944 的偏移量，这个偏移量直接加到 gs 上即可。

Assignment 2 实模式中断

根据 int 10h 的文档，获取光标和光标移动可以分为以下两步：

1. 将 ah 设为 03h，然后调用 int 10h，即获取光标位置，光标的行号和列号分别存储在 dh 和 dl 中；

2．要移动光标，只需要对 dh 和 dl 进行算数操作，移动到所需的位置后，将 ah 设为 02h，然后调用 int 10h，即完成移动操作。

在此基础上，要输出学号，只需要把 1.2 中把学号传入显存的步骤换成调用中断即可。

除此之外，要实现键盘输入并回显，也分以下两步：

1．调用 int 16h 的 0 号功能，等待键盘输入

2．调用 int 10h 的 09h 号功能，将输入显示

第一步会把字符的 ASCII 码存到 al，所以第二步可以直接输出。

Assignment 3 汇编

汇编中写分支语句大致遵循以下方法：

1．设置一个分支结尾 if_end

2．每个分支的头部定义标签，结尾写 jmp if_end，这样每个分支就不会互串

3．分支开头判断条件，引导到各个分支

循环语句大致遵循以下方法：

1．定义标签 for_begin,for_end 或者 while_begin,while_end

2．循环头部判断循环条件（如这里的 a2>=12），不满足则跳转到 end

3．循环块（如 call my_random 和写入 while_flag）

4．更新循环变量（如果有必要），jmp 到 begin

除此之外还有其他的实现方式，比如在循环结尾才判断条件（类似 C 语言的 do-while）要视情况而定。

函数的实现见关键代码。

Assignment 4 汇编小程序

程序要实现的功能分为两部分：渲染反弹的数字、渲染标题。从示例视频看标题不会被反弹的数字覆盖，所以为了方便起见，每次都完整渲染一次标题。

渲染反弹的数字在反弹逻辑设计上如果直接分四个状态做会比较困难。不妨拆分成水平方向速度和垂直方向速度，独立地检测水平和垂直是否即将要出界，如果是，则将对应方向的速度取负数即可。

剩下的部分比较简单：要渲染变化的数字，直接用字符串“1357924680”加上一个循环的下标即可；渲染变化的颜色直接用一个 byte，每次加一即可。

3. 关键代码

Assignment 1 MBR

复现 Example 1，使用循环方式：

```
[bits 16]
xor ax, ax
; 初始化段寄存器
mov cx, 0x07c0
mov ds, cx
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax

; 初始化栈指针
mov sp, 0x7c00

; 初始化 gs 到 0xb800
mov ax, 0xb800
mov gs, ax

mov ah, 0x02 ;绿色
mov cx, 11  ; 11 是'hello world'的长度
```

```
mov si, 0
mov bx, 0

print_hello_world:
    mov al, [si + string]
    mov [gs:bx], ax
    inc si
    add bx, 2
    loop print_hello_world

string db 'Hello World'

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

显示学号，将 ah 改为 0x12，这样就变成了前景绿色、背景蓝色；bx 初始化为 1944，就可以改变输出位置；将 cx 改为 8，string 改为 23336223，可以改变输出内容。

Assignment 2 实模式中断

光标移动：

```
; 获取光标位置
mov ah, 03h
int 10h

; 设置光标位置
add dh, 1
add dl, 2
mov ah, 02h
int 10h
```

该代码会获取初始光标位置，并依次向右一格、向下一格移动。

输出学号：

```
; 设置光标位置为(12,12)
mov dh, 12
mov dl, 12
mov ah, 02h
int 10h
```

```
mov bl, 0x12; 颜色
mov si, 0
mov cx, 1

print_hello_world:
    ; 写字符
    mov al, [si + string]
    mov ah, 09h
    int 10h

    ; 移动光标
    inc dl
    mov ah, 02h
    int 10h

    ; 当 si<8 时继续循环
    inc si
    cmp si, 8
    jb print_hello_world

count dw 8
string db '00000000'
```

因为 int 10h 的 09h 号功能调用会用到 cx 寄存器，和 loop 指令有冲突，所以不使用 loop 指令。

获取键盘输入并回显：

```
; 获取键盘输入
mov ah, 0
int 16h

; 回显
mov bl, 07h
mov cx, 1
mov ah, 09h
int 10h

jmp $
```

Assignment 3 汇编

分支语句：

```
your_if:
    cmp dword [a1], 12
    jge else_if
    mov eax, dword [if_flag]
    mov ecx, 2
    xor edx, edx
    div ecx
    inc eax
    mov [if_flag], eax
    jmp your_if_end
else_if:
    cmp dword [a1], 24
    jge else
    mov eax, 24
    sub eax, dword [a1]
    mov ecx, dword [a1]
    xor edx, edx
    mul ecx
    mov [if_flag], eax
    jmp your_if_end
else:
    mov eax, dword [a1]
    shl eax, 4
    mov [if_flag], eax
your_if_end:
```

循环语句：

```
your_while:
    cmp dword [a2], 12
    jl your_while_end

    call my_random
    mov ecx, dword [a2]
    sub ecx, 12
    mov ebx, dword [while_flag]
    mov [ebx+ecx], al
    dec dword [a2]
    jmp your_while
your_while_end:
```

函数调用：

```
your_function:
    mov eax, dword [your_string]
    xor ebx,ebx
```

```
    for:
        cmp byte [eax], 0
        je for_end
        pushad
        mov bl, byte [eax]
        push ebx
        call print_a_char
        pop ebx
        popad
        inc eax
        jmp for
    for_end:
    ret
```

Assignment 4 汇编小程序

将两支数字的位置和方向向量以及其他数据保存到内存：

```
title db 'test 00000000',0
string db '1357924680',0

string_index_1 dw 0
color_1 db 0
pos_1 db 2,0
velocity_1 db 1,1

string_index_2 dw 0
color_2 db 0x21
pos_2 db 3,79
velocity_2 db -1,-1
```

渲染反弹数字（其中一支）：

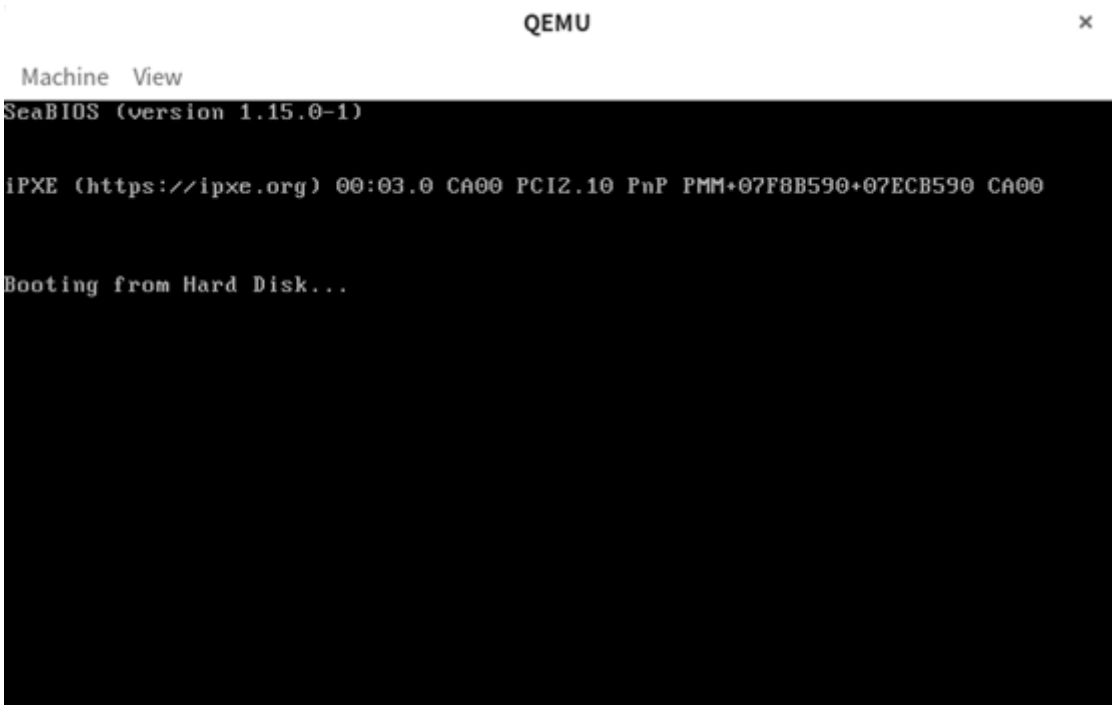
```
; 移动光标
mov dh, byte [pos_1]
mov dl, byte [pos_1 + 1]
mov bh, 0x00
mov ah, 0x02
int 10h

; 获取当前位置的字符
movzx si, byte [string_index_1]
mov al, byte [string + si]
mov bl, byte [color_1]
mov ah, 0x09
```

```
mov cx, 1
int 10h

inc bl
mov byte [color_1], bl

inc si
mov al, byte [string + si]
cmp al, 0
jne reset_index_end_1
mov si, 0
reset_index_end_1:
mov [string_index_1], si

; 假设不反弹，计算新的位置
mov ah, byte [velocity_1]
mov al, byte [velocity_1 + 1]
add ah, dh
add al, dl

; 检查垂直方向上是否要反弹
cmp ah, 0
jl reverse_velocity_x_1
cmp ah, 25
jge reverse_velocity_x_1
jmp reverse_velocity_x_end_1

reverse_velocity_x_1:
    neg byte [velocity_1]
    mov ah, byte [velocity_1]
    add ah, dh
reverse_velocity_x_end_1:

; 检查水平方向上是否要反弹
cmp al, 0
jl reverse_velocity_y_1
cmp al, 80
jge reverse_velocity_y_1
jmp reverse_velocity_y_end_1

reverse_velocity_y_1:
    neg byte [velocity_1 + 1]
    mov al, byte [velocity_1 + 1]
    add al, dl
```

```
reverse_velocity_y_end_1:

; 保存新的位置
mov byte [pos_1], ah
mov byte [pos_1 + 1], al
```

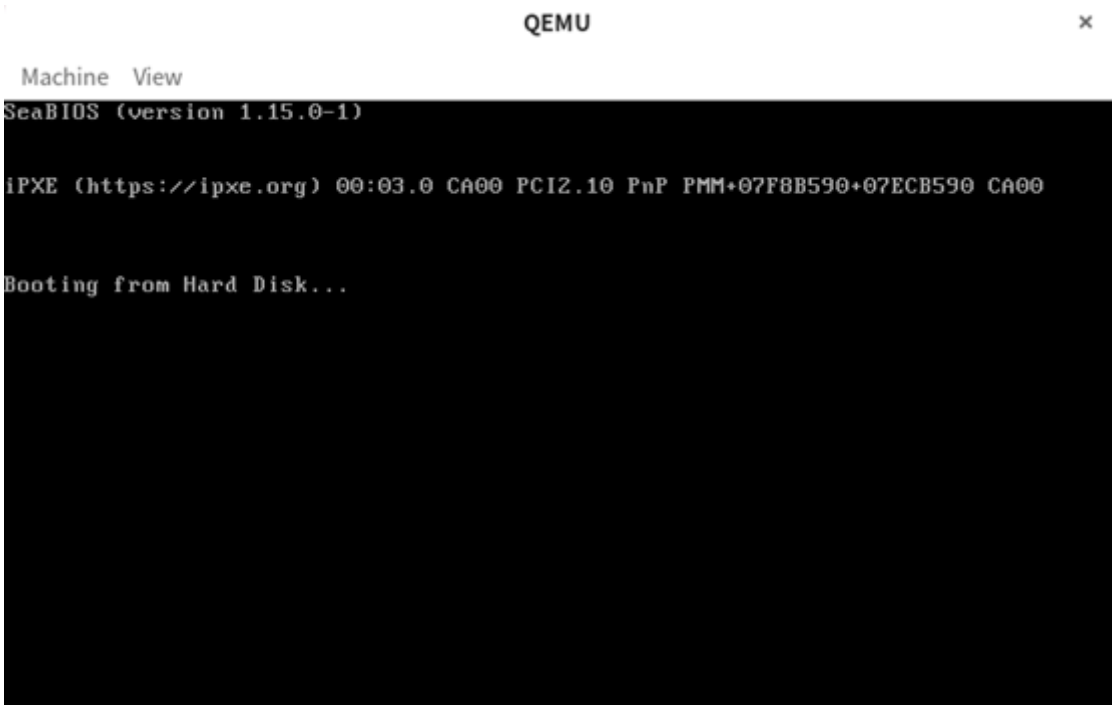
4. 实验结果

Assignment 1 MBR



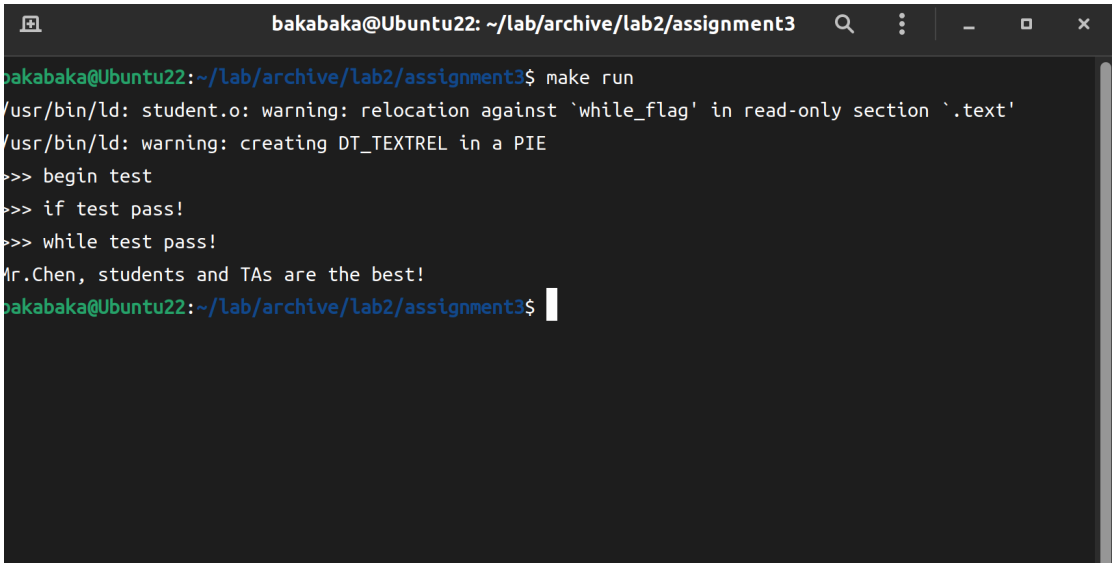
Assignment 2 实模式中断

和 Assignment 1 中的光标相比，确实向右下方移动了指定的距离

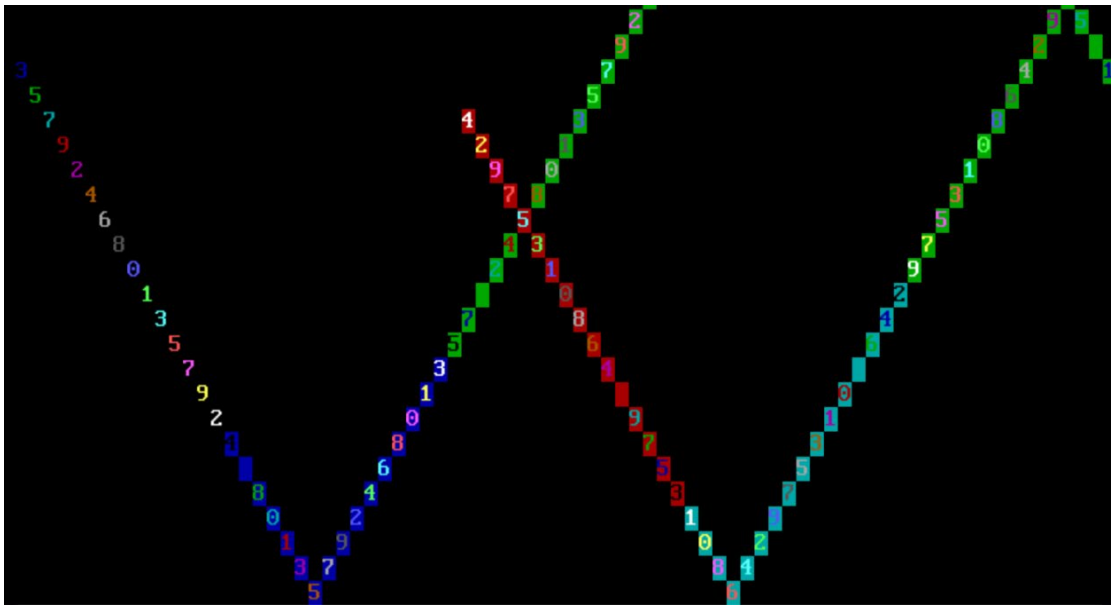


和 1.2 的效果相同。

Assignment 3 汇编



Assignment 4 汇编小程序



5. 总结

写 Assignment 3 的时候遇到了很大困难：while test 时一直遇到段错误并且找不到原因，后来发现 while_flag 是一个函数指针，在汇编里，应该先解引用获得数组头地址，加上偏移量后二次解引用，才是真正的要存放的位置。如果缺少第一次解引用，就很可能改变指针的值，之后的读取就可能读到未分配的内存而发生段错误。