

# 操作系统实验报告

2024 年 4 月 11 日

## 一、实验要求

在 PA1 完成的基础上，使用 nasm 实现一个类似 LILO 的交互式引导程序用于替换原 MBR 中的引导代码，其主要功能：

- 1) 列出可供选择的启动分区（windows 分区、linux 分区）列表；
- 2) 用户可用上下键或数字键进行选择（回车键完成选择）；
- 3) 根据用户选择加载指定分区上的操作系统（或引导程序）。

提交 nasm 格式的汇编代码、修改完成的 MBR 和实验报告！

提示：

- 1) MBR 中的代码采用的是 i8086 体系架构上的 16 位指令（另需要注意代码尺寸，不能超过分区表!!!）
- 2) 合理利用 BIOS 提供的基本输入输出功能：int10（屏幕输出），int 13（磁盘访问），int 16（键盘输入）

## 二、实验环境

- 操作系统：Windows 11
- 编程语言：汇编
- 使用工具：qemu;vscode Hex Editor
- 虚拟系统版本:windows-7;linux-20-desktop

## 三、实验原理

i8086 体系下，计算机开机后会首先读取 BIOS 中的程序，接下来进行硬件自检，自检完成后 BIOS 将控制权交给下一阶段的启动阶段。我们需要修改 MBR 中的代码，在启动到

一阶段的时候，实现能够选择引导 Windows 和 linux 的二阶段加载。我们只需要找出这两者的二阶段分别在哪里，利用一些指令来引导它们即可。

## 四、实验过程

实验主要分为以下几个步骤：

1. 获取虚拟磁盘原始 MBR 数据：这是重要的参考
2. 搭建框架：编写汇编代码框架，利用 int 13h 读磁盘, int 10h 打印, int 16h 输入
3. 编写引导 windows 二阶段的代码
4. 编写引导 linux 二阶段的代码
5. 写入磁盘，查看运行效果
6. 优化 UI 界面，增加部分功能

### 4.1 获取虚拟磁盘原始 MBR 数据

这个步骤在实验 1 已经完成，与之前不同的是，我采取了 ndisasm 来进行反汇编，这个结果更准。相应获取的反汇编代码将在后面步骤上利用到。

### 4.2 搭建框架

首先搭建起整套汇编代码框架，实现屏幕能显示 1.windows 和 2.linux，确保基础流程正确

```
; 设置代码段的起始位置
BITS 16
SECTION MBR vstart=0x7c00

; 初始化堆栈(只安装了 windows 系统的 mbr 中最前面就是这些指令)
mov ax, cs
mov ds, ax
mov es, ax
mov ss, ax
mov sp, 0x7c00

_start:
; 清屏
```

```
mov ax, 0x600
mov bx, 0x700
mov cx, 0
mov dx, 0x184f
int 10h

; 打印引导标签
mov si, lead_label
call print
mov si, windows_label
call print
mov si, linux_label
call print

; 输入
call input

; 重新显示标签, 也可以写成jmp $原地等待
jmp _start

__exit:
ret

; 打印字符串
print:
lodsb
or al, al
jz __exit
mov ah, 0x0E
int 10h
jmp print

; 用户输入
input:
mov ah, 0
int 16h
```

```
cmp ah, 0
je input

cmp al, '1'
je windows_selected
cmp al, '2'
je linux_selected
jmp input

; 进入 Windows
windows_selected:
    (to do)

; 进入 Linux
linux_selected:
    (to do)

; 定义字符串
lead_label db "press 1 for windows, 2 for linux", 0x0D, 0x0A, 0
windows_label db "1. windows", 0x0D, 0x0A, 0
linux_label db "2. linux", 0x0D, 0x0A, 0
selected_partition db 0 ; 可在这里修改默认选择的分区

; MBR 结束标志
times 510-($-$$) db 0
dw 0xAA55
```

### 4.3 编写引导 windows 二阶段的代码

参考[windows 反汇编 MBR 代码解析](#)，我们可以知道 windows 本身引导二阶段是在做哪些事情。关键部分如下：

将 mbr 从 0x7c00 移走到 0x600

00000000	33C0	xor ax, ax	// 设置 ax 寄存器的值为 0
00000002	8ED0	mov ss, ax	

```

00000004  BC007C          mov  sp,0x7c00 // 设置栈值

//把MBR从0x7c00 移动到 0x600
00000007  8EC0          mov  es,ax
00000009  8ED8          mov  ds,ax
0000000B  BE007C        mov  si,0x7c00
0000000E  BF0006        mov  di,0x600
00000011  B90002        mov  cx,0x200
00000014  FC           cld   // 设置寄存器为自增
00000015  F3A4         rep  movsb //拷贝
00000017  50           push ax //入栈 16bit
00000018  681C06       push word 0x61c //入栈
0000001B  CB           retf //调到0x61c继续执行
0000001C  FB           sti

```

在这一步骤，需要格外注意 `push word 0x61c` 这个语句需要进行修改，为了能继续执行我们手写 MBR 的代码，我们需要将手写的 MBR 的 `asm` 文件编译再反编译，看到 `sti` 这个语句的地址是什么，填入这个地址。以后每次修改都需要重复这次操作。

#### 找 windows 活动分区跳转执行

```

;CHS法
mov  bp, 0x7be
mov  ax, 0x0201
mov  bx, 0x7c00
mov  dl,[bp+0x0]
mov  dh,[bp+0x1]
mov  cl,[bp+0x2]
mov  ch,[bp+0x3]
int  0x13

jmp  0x0000:0x7c00

```

通过分区表我看出，我的 windows 的加载器在第一个分区的活动扇区，0x7be 就是分区表第一个分区的入口，所以省略了找活动分区的步骤。采用的和是 windows-7 本身一样的读取磁盘方式:CHS 方法。至此已经完成了引导 windows 二阶段的编写。

#### 4.4 编写引导 linux 二阶段的代码

阅读我的虚拟磁盘的反汇编代码，可以知道原本 linux 的引导过程采用的是 LBA 方法。通过查询资料，我得知我的 linux 二阶段就在 1 号扇区。于是操作如下：

```
mov dl, 0x80; 必要语句，设置驱动器号
```

```
;LBA法，读到0x7000
```

```
xor ax, ax
```

```
mov [si+0x4], ax
```

```
inc ax
```

```
mov [si-0x1], al
```

```
mov [si+0x2], ax
```

```
mov word [si], 0x10
```

```
mov ebx, 0x00000001
```

```
mov [si+0x8], ebx
```

```
mov ebx, 0x00000000
```

```
mov [si+0xc], ebx
```

```
mov word [si+0x6], 0x7000
```

```
mov ah, 0x42
```

```
int 0x13
```

```
;将start从0x7000移动到0x8000启动
```

```
mov bx, 0x7000
```

```
mov es, bx
```

```
xor bx, bx
```

```
mov ax, 0x201
```

```
int 0x13
```

```
mov bx, es
```

```
pusha
```

```
push ds
```

```
mov cx, 0x100
```

```
mov ds, bx
```

```
xor si, si
```

```
mov di, 0x8000
```

```
mov es, si
cld
rep movsw
pop ds
popa
jmp 0x8000
```

注意第一句为必要语句, 这是在设置驱动器号, 省略了就没法运行。

#### 4.5 写入磁盘, 查看运行效果

编译 **asm** 文件 (我的 **asm** 叫 **newnew.asm**, 编译生成的 **bin** 叫 **win.bin**)

```
nasm -f bin C:\Users\86181\Desktop\Program\
VscodeData\markdown\newnew.asm -o win.bin
```

反汇编 **win.bin**(这步是为了找 **sti** 指令地址)

```
ndisasm win.bin
```

##### 写入虚拟磁盘

注意: 需要将原始 MBR 从 441 位到 512 位的数据复制过来, 多复制 6 位是因为要告知磁盘名称信息

```
dd if=win.bin of=disk2.raw bs=512 count=1
```

##### 运行

```
qemu-system-x86_64 -bios D:\MyOwnFiles\qemu\share\bios.bin -drive
file=disk2.raw,format=raw -m 2G -smp 2
```

打开后的界面分别如下图 (后续选 **windows** 能拉起自己的二阶段, 选 **linux** 能打开 **grub**, 选择进入 **windows** 或者 **Linux** 且均能成功)

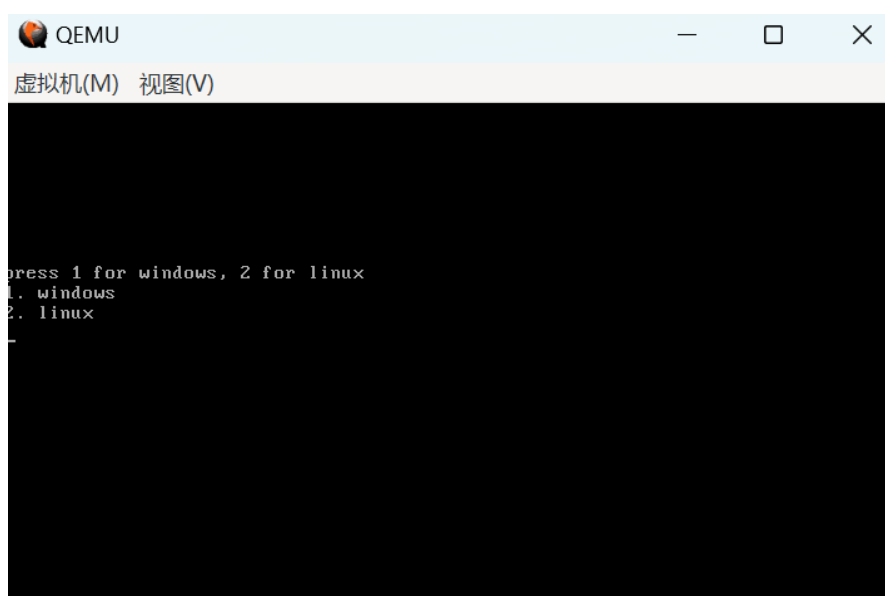


图 1: 菜单界面

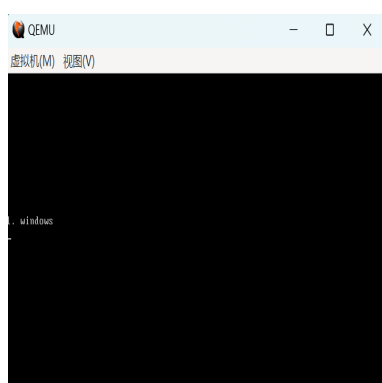


图 2: 输入 1, 快速打印一个 1.windows 并且拉起 windows 二阶段

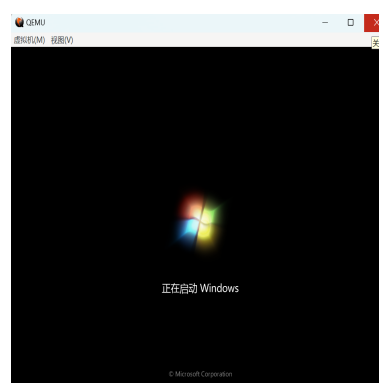


图 3: 输入 1, 快速打印一个 1.windows 并且拉起 windows 二阶段



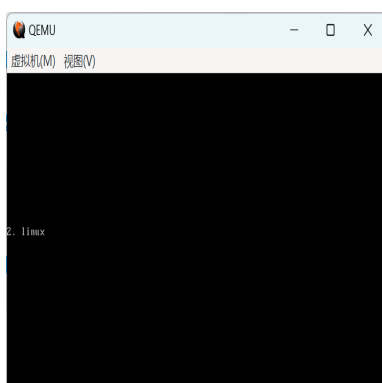


图 4: 输入 2, 快速打印一个 2.linux 并且拉起 linux 二阶段

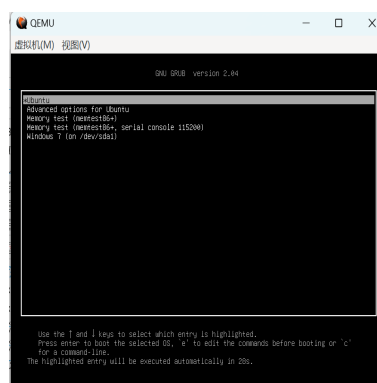


图 5: 输入 2, 快速打印一个 2.linux 并且拉起 linux 二阶段

## 4.6 优化 UI 界面, 增加部分功能

### 4.6.1 增加等待 enter 的功能

由于输入 1 或者 2 后立马进入了各自的二阶段, 这样不好, 容易误操作, 所以增加了需要等待 enter 按下才进入的功能。

### 4.6.2 增加多次选择功能

由于老师需要的是可以切换选择, 为了实现类似功能, 我增加了可以修改选择的功能, 即可以先选 1 再选 2 再选 1 等等, 选中的会打印出来提示用户, 按下 enter 后才进入二阶段。

### 4.6.3 增加可以默认选择的功能

只需要修改标签默认值即可, 比如: `selected_partition db 1` 就可以默认选择 windows。

## 五、实验结果

本人完全完成了本次实验的要求, 实现了一个二阶段加载引导器, 列表体现在开机界面上, 使用数字可以选择进入 windows 或者 linux, 进入后能正确引导二阶段, 并且增加了附加的优化功能。为了严格保证我的作业完成度, 可以查看我的作业视频 (必须下载后才能看到): [视频链接](#)

## 六、总结与思考

- 理论好理解不代表实验好做，实验需要非常多的细节，比如说 retf 语句找 sti 地址继续执行这一句，就不能从原始的 MBR 照抄，因为新写的 MBR 文件下该语句地址会变，在这一步上我耗费了比较多的时间
- STFW,RTFM 是计算机大类学生必备的素养，知识需要主动检索和学习，我检索到的有用的资料见第八节
- 通过本次实验，我对 MBR 的理解更深了一步。在手写 MBR 的过程中，对操作系统起到的引导和控制作用有了更直观的体验，int 中断指令也不那么抽象了。我也享受到了制作 MBR，不断优化的快乐。

## 七、完整源代码以及完整注释

； 设置代码段的起始位置

`BITS 16`

`SECTION MBR vstart=0x7c00`

； 初始化堆栈 ( 只安装了 windows 系统的 mbr 中最前面就是这些指令 )

`mov ax, cs`

`mov ds, ax`

`mov es, ax`

`mov ss, ax`

`mov sp, 0x7c00`

`__start:`

； 清屏

`mov ax, 0x600`

`mov bx, 0x700`

`mov cx, 0`

`mov dx, 0x184f`

`int 10h`

； 打印引导标签

`mov si, lead_label`

`call print`

```
mov si, windows_label
call print
mov si, linux_label
call print

; 输入
call input

; 重新显示标签，也可以写成jmp $原地等待
jmp __start

__exit:
ret

; 打印字符串
print:
lodsb
or al, al
jz __exit
mov ah, 0x0E
int 10h
jmp print

; 用户输入
input:
mov ah, 0
int 16h
cmp ah, 0
je input

cmp al, '1'
je select_windows ; 想选 windows
cmp al, '2'
je select_linux ; 想选 linux
jmp input
```

```
; 选择 Windows 分区
select_windows:
; 清屏
mov ax, 0x600
mov bx, 0x700
mov cx, 0
mov dx, 184fh
int 10h
; 等 enter
mov byte [selected_partition], '1'
mov si, windows_label
call print
jmp wait_enter

; 选择 Linux 分区
select_linux:
; 清屏
mov ax, 0x600
mov bx, 0x700
mov cx, 0
mov dx, 184fh
int 10h
; 等 enter
mov byte [selected_partition], '2'
mov si, linux_label
call print
jmp wait_enter

; 进入 Windows
windows_selected:
; 基本照着只装了 windows 的原始的MBR来写就行

; 将MBR从0x7c00移到0x600
xor ax, ax                ; 清零 ax 寄存器
mov ss, ax                ; 将 ax 的值移动到 ss 寄存器
mov sp, 0x7c00            ; 将栈指针 sp 设置为 0x7c00
```

```

mov es, ax          ; 将ax的值移动到es寄存器
mov ds, ax          ; 将ax的值移动到ds寄存器
mov si, 0x7c00       ; 将源地址0x7c00移动到si寄存器
mov di, 0x600        ; 将目的地址0x600移动到di寄存器
mov cx, 0x200        ; 将数据大小0x200移动到cx寄存器
cld                 ; 清除方向位
rep movsb           ; 重复执行movsb指令，将数据从si复制到di

```

； 这步需要注意将push word后的地址找出来，不是原本的地址，  
 ； 而是这个新写的MBR汇编成的二进制代码再反汇编生成对应的顺序地址  
 ； （加上0x600偏移量）

```

push ax             ; 将ax寄存器的值入栈
push word 0x6A3     ; 将0x6A3入栈
retf                ; 返回前先弹出标志寄存器，然后根据指定的代码段和偏移地址

```

； 我的Windows的加载器在第一个分区的活动扇区0x7be  
 ； 使用CHS法

```

mov bp, 0x7be       ; 将0x7be移动到bp寄存器，该地址指向活动标志的位置
mov ax, 0x0201       ; 将0x0201移动到ax寄存器，指定扇区数和扇区号
mov bx, 0x7c00       ; 将0x7c00移动到bx寄存器，指定数据的加载地址
mov dl, [bp+0x0]     ; 将bp寄存器偏移0x0处的值移动到dl寄存器，指定驱动器号
mov dh, [bp+0x1]     ; 将bp寄存器偏移0x1处的值移动到dh寄存器，指定磁头号
mov cl, [bp+0x2]     ; 将bp寄存器偏移0x2处的值移动到cl寄存器，指定扇区号的低8
mov ch, [bp+0x3]     ; 将bp寄存器偏移0x3处的值移动到ch寄存器，指定扇区号的高2
int 0x13             ; 调用int 0x13中断
jmp 0x0000:0x7c00    ; 跳转到0x7c00地址

```

； 进入 Linux

linux\_selected:

； 大多数指令来自原汇编代码

```

mov dl, 0x80        ; 将驱动器号设为0x80

```

； 使用LBA法，读到0x7000

```

xor ax, ax          ; 清零ax寄存器
mov [si+0x4], ax     ; 将0写入si+0x4处，即设定LBA高字节为0

```

```
inc ax ; ax加1, 此时ax=1
mov [si-0x1], al ; 将al的值写入si-0x1处, 即设定LBA低字节为1
mov [si+0x2], ax ; 将ax的值写入si+0x2处, 即设定LBA的低16位为1
mov word [si], 0x10 ; 将0x10写入si处, 即设置读取的扇区数量为16
mov ebx, 0x00000001 ; 将1写入ebx寄存器
mov [si+0x8], ebx ; 将ebx的值写入si+0x8处, 即设定读取的磁头号为1
mov ebx, 0x00000000 ; 将0写入ebx寄存器
mov [si+0xc], ebx ; 将ebx的值写入si+0xc处, 即设定读取的柱面号为0
mov word [si+0x6], 0x7000 ; 将0x7000写入si+0x6处, 即设置读取
; 的起始扇区号为0x7000
mov ah, 0x42 ; 将0x42写入ah寄存器, 即设定读取扇区的子功能号
int 0x13 ; 调用0x13中断

; 将start从0x7000移动到0x8000启动
mov bx, 0x7000 ; 将0x7000写入bx寄存器
mov es, bx ; 将bx的值写入es寄存器, 即将es寄存器设定为0x7000
xor bx, bx ; 将bx寄存器清零
mov ax, 0x201 ; 将0x201写入ax寄存器, 指定读取1个扇区到es:bx
int 0x13 ; 调用0x13中断
mov bx, es ; 将es的值写入bx寄存器
pusha ; 将通用寄存器入栈
push ds ; 将ds寄存器入栈
mov cx, 0x100 ; 将0x100写入cx寄存器, 指定移动的字节数
mov ds, bx ; 将bx的值写入ds寄存器
xor si, si ; 将si寄存器清零
mov di, 0x8000 ; 将0x8000写入di寄存器, 即设置目的地址为0x8000
mov es, si ; 将si的值写入es寄存器, 即将es寄存器设定为0x8000
cld ; 清除方向位, 确保字符串传送操作作为正向传送
rep movsw ; 重复执行movsw指令, 将ds:si指向的字符串复制
; es:di指向的位置

pop ds ; 弹出ds寄存器的值
popa ; 弹出通用寄存器的值
jmp 0x8000 ; 跳转到0x8000

; 等待用户按下 Enter 键
wait_enter:
```

```
mov ah, 0
int 16h
cmp al, 0x0D ; Enter 键的 ASCII 码为 0x0D
je execute_selected_partition
; 使用户可以切换 1, 2
cmp al, '1'
je select_windows
cmp al, '2'
je select_linux
jmp wait_enter

; 执行用户选择的分区
execute_selected_partition:
cmp byte [selected_partition], '1'
je windows_selected
cmp byte [selected_partition], '2'
je linux_selected
jmp __start

; 定义字符串
lead_label db "press 1 for windows, 2 for linux", 0x0D, 0x0A, 0
windows_label db "1. windows", 0x0D, 0x0A, 0
linux_label db "2. linux", 0x0D, 0x0A, 0
selected_partition db 0 ; 可在这里修改默认选择的分区
```

## 八、其它参考文献

除了正文给出的参考文献，我参考的文献还有：

MBR 分区表详解

Linux 磁盘管理和文件系统

Windows 操作系统引导过程

读取磁盘：CHS 方式

磁盘寻址方式-CHS 和 LBA 寻址方式