实验二：加载执行COM格式用户程序的监控程序

..

..

1 实验题目

模拟单道批处理系统，编写一个简单的监控程序，使其能从键盘输入指令，并加载执行指定的 COM 格式的用户程序。

具体来说，共分为三个步骤：

1. 将实验一的引导扇区程序修改为 COM 格式的程序，缩小显示区域，分别在屏幕的四个 1/4 区域显示。显示一些信息后，程序会结束退出。在 1.44MB 软驱映像中存储这些程序；
2. 重写 1.44MB 软驱引导程序，利用 BIOS 调用，实现一个能执行 COM 格式用户程序的监控程序。解决加载用户程序和返回监控程序的问题。执行完一个用户程序后，可以执行下一个；
3. 设计一种命令，可以在一个命令中指定某种顺序执行若干个用户程序。可以反复接受命令。

2 实验目的

1. 了解监控程序执行用户程序的主要工作
2. 了解一种用户程序的格式与运行要求
3. 加深对监控程序概念的理解
4. 掌握加载用户程序方法
5. 掌握几个BIOS调用和简单的磁盘空间管理

3 实验要求

1. 知道引导扇区程序实现用户程序加载的意义
2. 掌握COM/BIN等一种可执行的用户程序格式与运行要求
3. 用户程序统一加载到内存 `0x0000:0x8100` 处

4 实验方案

工具与环境：纯汇编，NASM 编译器，bochs 虚拟机（1MB 内存、1.44MB 软驱）

执行用户程序分为两步：从外存中读取用户程序到内存、指令跳转到用户程序。难点在于用户程序执行完后回到监控程序。从外存读取内容到内存、键盘输入都使用 BIOS 中断服务完成。

结合布置的实验内容，与我自己上一次实验的基础，我将按照如下步骤进行：

1、修改实验一的弹射字母程序 包括加入数据内存的使用（原来的程序完全使用寄存器，为了使其更具有测试通用性，需加上数据内存的使用）、修改为 COM 格式、修改为有限执行（本实验模拟的是单道批处理系统，故需假设用户程序都能正常退出）、备份成四个不同区域射字母的程序；

2、编写可执行用户程序的监控程序 一步一步来，先使得监控程序能正常地执行用户程序并返回；

3、编写监控程序的命令输入 使得监控程序具备应有的基本功能。

实验过程部分将以流畅的方式叙述实验细节，更多试错过程写在实验总结部分。

5 实验过程

5.1 前置知识学习

实验一的程序由于只用了寄存器，因此没有 `org` 语句，也没有深入了解 x86 的寻址模式（这一点严重阻碍了后续程序理解与编写）。因此此处需要先学习。

阅读《x86汇编语言：从实模式到保护模式》2.5.3 内存分段机制[1]可知：在 16 位的 8086 汇编语言中，最大内存为 1MB（即 20 位内存地址），使用“(段地址<<4)+偏移地址”的方式来表示，例如当前指令地址（段地址寄存器 `cs`、偏移地址寄存器 `ip`）为 `(cs<<4)+ip`。段地址和偏移地址各是 16 位的，使用该方法计算则可得一个 20 位地址。

由此可得，该寻址模式的优点为内存分段非常自由，可以从任意 16 倍数的地址开始一个任意长度的段（长度在 2^{16} 以内）。缺点就是对内存管理要求高，处理不好的话会造成混乱与冲突。

而在编写汇编程序时所用到的数据地址，编译以后只是给出偏移地址。若无 `org` 语句，则所有偏移地址视为其与程序开头的偏移距离。这样一来，例如把引导程序加载到了 `0x7c00` 处（此时段地址为 `0`、偏移地址 `0x7c00`），理应把程序所有的数据地址偏移都加上 `0x7c00` 才对。在程序开头写上 `org 0x7c00`，就达到了这个效果。

5.2 修改弹射字母程序

5.2.1 使用数据内存

将输出学号姓名的部分由原来的每个字符手动输入显存，改成在数据内存中预存一个字符串，然后循环输入显存，修改部分代码如下：

```
mov ax, 0                           ; loop var, from 0 to 11
FOR_2:                                  ; ID & name start at (10,20)
        mov di, ax
        add di, 20
        add di, 800                     ; di=10*80+20+ax
        imul di, 2
        mov si, ax
        add si, ID_Name
        mov byte bh, [si]
        mov [es:di], bh                 ; char, es=0xB800
        mov byte [es:di+1], 0xA1        ; background
        inc ax
        cmp ax, 11
        jle FOR_2

datadef:
ID_Name db '18340083 KQP'
```

由于使用了数据内存，必须注意 org 语句的使用。例如将其写到格式化的空软驱的引导程序位置进行测试，引导程序在内存的起始位置为 0x7c3e，因此该程序开头要加上

```
org 0x7c3e
```

运行该软驱进行测试，效果如左图。若不加 org 或者 org 错了位置，则如右图，显示一堆乱码。

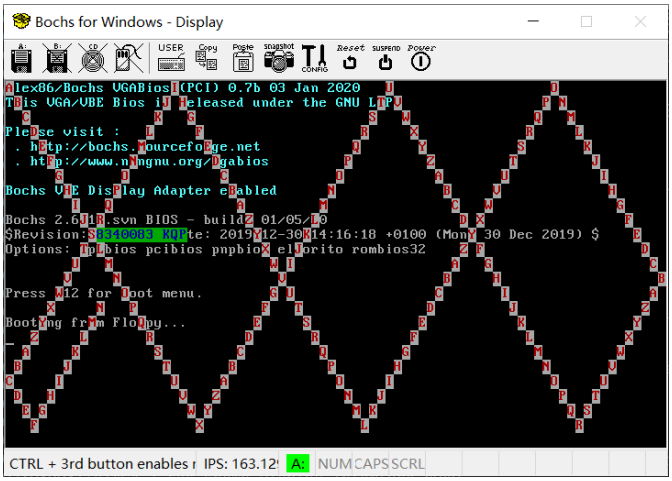


图 1: 测试成功

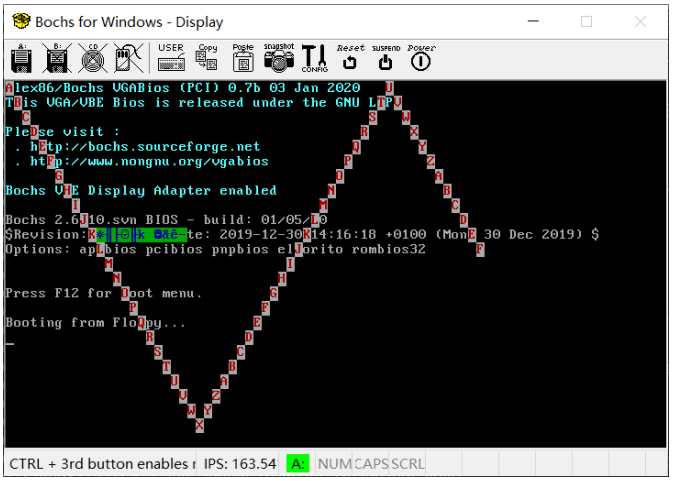


图 2: org错误导致乱码

5.2.2 修改弹射范围

要制作四个弹射程序，分别在屏幕的左上、右上、左下、右下弹射。只需修改按下表修改相关参数即可。修改过程比较琐碎，此处不赘述，效果如下图。

文件名	弹射范围	左上坐标	右下坐标
shoot1.asm	左上	(0,0)	(11,39)
shoot2.asm	右上	(0,40)	(11,79)
shoot3.asm	左下	(12,0)	(23,39)
shoot4.asm	右下	(12,40)	(23,79)

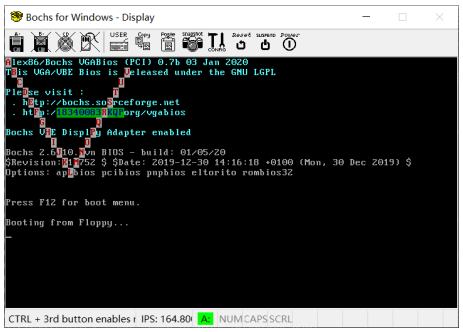


图 3: shoot1

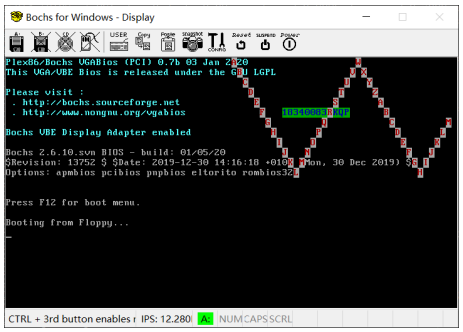


图 4: shoot2

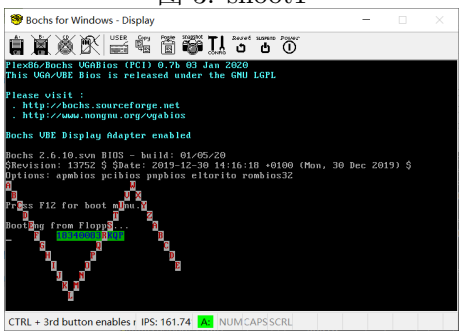


图 5: shoot3

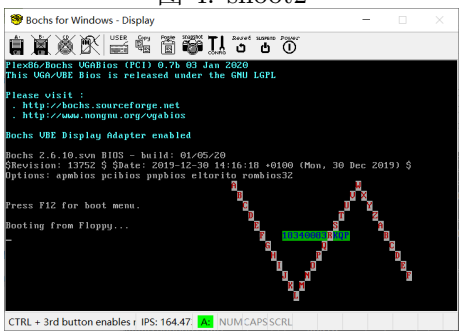


图 6: shoot4

5.2.3 改为有限执行

规定程序从左边界运动到右边界就结束。具体更改为 `foo` 过程中判断触碰右边界后:（以 `shoot1.asm` 为例）

```
...
IF_2: cmp bl, 39                ; rightmost
        jne IF_3
        ret
...
```

5.3 编写简单监控程序

5.3.1 BIOS中断

大多数与硬件相关的操作将通过 BIOS 中断服务实现。调用方法为，先在指定的寄存器中设置好参数，然后在寄存器 `ah` 设置功能号，最后 `int` 中断号 即会调用 BIOS 中断。可百度查询常用的 BIOS 中断功能及其参数[2]。例如封装一个清屏过程和输出欢迎信息过程如下：

```
clear_screen:
mov al,0                                ; clear screen
mov ch,0                                ; upper-left row
mov cl,0                                ; upper-left column
mov dh,23                               ; down-right row
mov dl,79                               ; down-right column
mov bh,0x07                             ; color&background
mov ah,0x06                             ; function: clear screen
int 0x10                                ; interrupt
ret

init_print:
mov al, 0x01                            ; pointer at the end of string
mov bx, 0x07                            ; page: bh=0,   color&background: 07h
mov dx, 0                               ; string position: (0,0)
mov cx, init1_len                       ; string length
mov ax, cs                              ; string address
mov es, ax
mov bp, init1
mov ah, 0x13                            ; function: display string
int 10H                                 ; interrupt
ret

datadef:
init1 db 'This is KQP_OS.'
init1_len equ ($-init1)
```

5.3.2 加载执行用户程序

分两步，第一步是使用 BIOS 中断来把用户程序从外存加载到内存。本实验中指定加载到内存 0x0000:0x8100 处，封装过程如下：（cl 表示从哪个扇区开始加载用户程序，即用户程序存放的扇区）

```
client_pos equ 8100h
client_pos_seg equ 800h

load_client:
mov ax, cs                          ; position in memory
mov es, ax
mov bx, client_pos
mov al, 1                           ; number of sectors to load
mov dl, 0                           ; driver ID
mov dh, 0                           ; head ID
mov cl, 2                           ; which sector to start
mov ch, 0                           ; cylinder ID
mov ah, 0x02                        ; function: read
int 13H                             ; interrupt
ret
```

第二步，将地址转移到用户程序处。难点在于用户程序执行完后如何回到监控程序。参考 DOS 系统运行 COM 格式程序的做法[3][4]，DOS 将用户程序加载到内存时会在程序头部留空 256 字节，用于写一个“返回监控程序”指令及文件相关信息，用户程序只需在结束后加上一句“返回程序头部”的指令，这样就能返回监控程序了。

我仿照该协议并简化，监控程序使用 call 指令跳到用户程序所在位置，并且要求用户程序结束以后必须加上 ret，这样就跳回来了。这样做，等价于把用户程序视作一个子过程调用之。

因此，load_client 过程在执行完 BIOS 中断之后，加上以下语句

```
mov ax, client_pos_seg
mov ds, ax
mov es, ax
call client_pos
```

就成为一个完整的“加载执行用户程序”过程了。

此处有一个细节，就是监控程序要为用户程序指定好段地址，存放在 ds,es 中。

5.3.3 修改用户程序为COM格式

刚才的弹射程序要改为 COM 格式也要做两点修改。第一，由于 COM 程序加载到内存后会留空 256 字节，因此用户程序头部应该 org 这个偏移：

```
org 100h
```

配合监控程序设好的段地址 ds,es，访问数据内存就没问题了。

第二，用户程序结束后加上 ret 语句：

```
main:
...
call foo        ; shoot procedure
ret
```

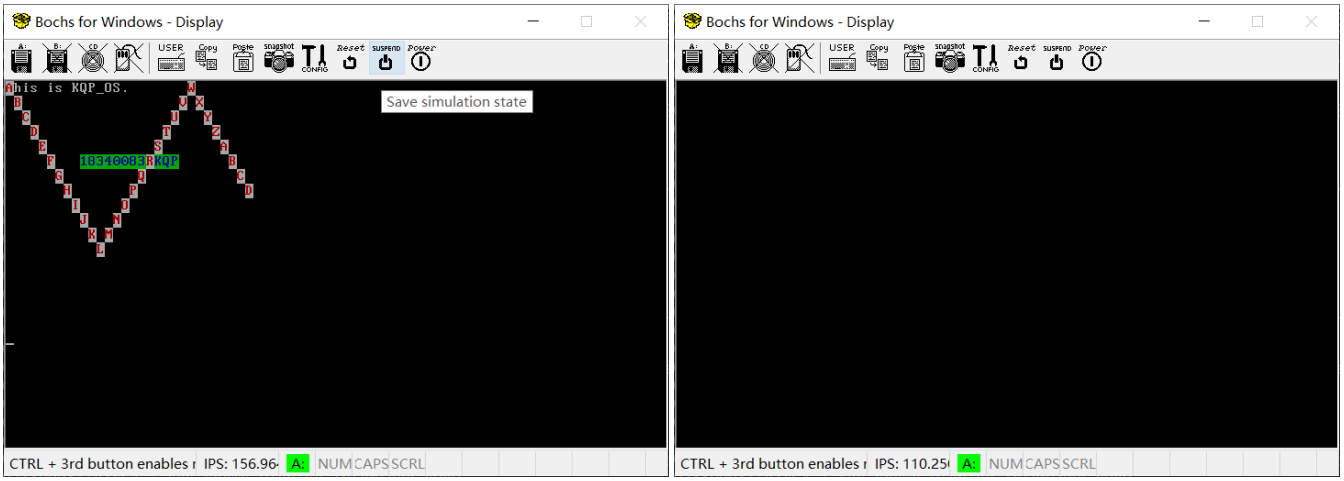
5.3.4 测试

监控程序的主函数写为

```
mbr:
call clear_screen
call init_print
call load_client
jmp $
```

即应该看到的效果为：先清屏并写入欢迎信息，然后执行左上射字母，最后再清屏死机。

使用 HexEditor 工具将监控程序写入首扇区引导程序的位置，将 shoot1 写入第 2 号扇区开头。运行结果如下：（助教可运行 init_monitor.bxrc 查看动态效果）



可以看到，运行结果符合期望，该步骤成功。

5.4 设计命令输入

通过汇编能实现的命令较为有限，若指令复杂则编程复杂度剧增，我认为更繁多的命令应留到后面用 C 语言实现，此处仅实现一个较为简单的命令——键盘输入 1 至 4 分别代表运行 4 个射字母程序。

这里要用到的就是 BIOS 中断服务中的键盘读入，代码如下：

```
read_program_id:
mov ah, 0                               ; function: input from keyboard
int 16H                                 ; interrupt
cmp al, 0
mov cl, al
xor cl, 48
inc cl
ret
```

读入的 ASCII 码会放在 `al`，将其异或 48 得到实际数字，再加 1 即为需要加载的扇区号，直接将其存入寄存器 `cl`，那么接下来调用 `load_client` 就不用对 `cl` 重新赋值了。

将监控程序主程序修改为

```
mbr:
        call clear_screen
        call init_print
        call read_program_id
        call load_client
jmp mbr
```

以实现循环调用。

将四个射字母程序分别写入 2 至 5 号扇区，监控程序写入首扇区引导程序位置，运行结果如下：（助教可运行 `myOS.bxrc` 查看动态效果）

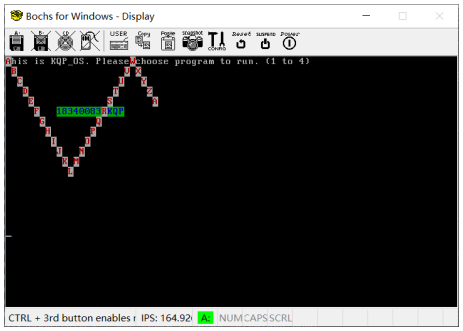


图 7: 输入1

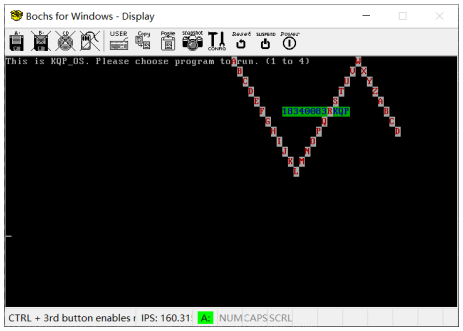


图 8: 输入2

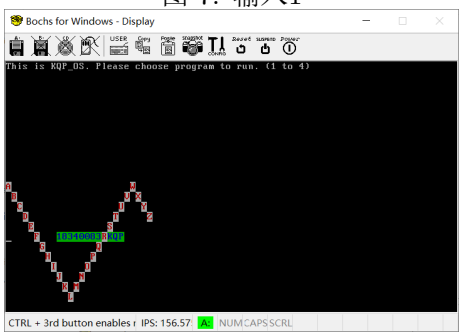


图 9: 输入3

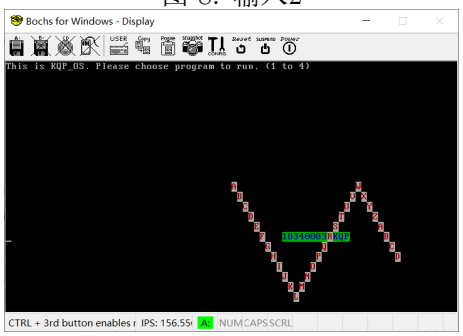


图 10: 输入4

可以看到，结果符合预期，该步骤成功。

6 实验总结

这次实验虽然操作比上次熟练，但花的时间更久了，原因是难度的提升、资料的缺失，以及部分资料的错误。

6.1 思考三个问题

一开始我就思考三个问题：1、大家都很强调的 `org` 到底是怎么用的？2、看 `org` 的介绍很混乱，x86 的寻址到底是怎么回事？3、监控程序去到用户程序很简单，问题是怎么回来？

前两个问题不理解，起源于对 `org 100h` 的不理解。计组学习的 mips、cache、虚拟内存段页式管理等内容，给我造成了 stereotype，让我觉得所谓“段地址+偏移地址”指的就是两者分别代表地址的高若干位和低若干位，根本没想过它们负责的二进制位居然会重叠。而同时，由于 ppt 上错将 `0x8100` 写为段地址，代码中 `[si]` 等隐式调用段地址的写法使我混乱，网上有些博客也写得很不负责任，一度使我迷惑不得前行。

最后我放弃了所有这些资料，阅读看起来像语法参考书的《x86汇编语言-从实模式到保护模式》。这下就看懂了，于是就有了本文 5.1 部分。所以有时候老老实实看书还是比上网搜乱七八糟的划得来。

至于第三个问题，我也思考了将近一天。课件及许多资料（包括《x86汇编语言》[5]）都是直接 `jmp` 过去的，但都没有讲怎么怎么回来。我想监控程序必须给用户程序指定返回的路径，也想过利用头部的 256 字节，后来同学给了 DOS 参考资料也确实是这样的。然而我认为让用户程序来 `return` 是不恰当的，这本应是系统的事，难道简简单单的一句话 `a=a+b;` 就没有资格作为用户程序了吗？

后来我是妥协了，接受了这个“协议”的办法，即规定作为应用程序必须具有这种格式（包含 `ret`），这相当于源代码编译成可执行文件要加很多额外的信息一样。

6.2 其他小错误

基本都跟地址有关。比如一开始直接按照各参考资料，给引导程序加的 `org 0x7c00` 发现输出字符串是乱码，后来发现我按正规的格式放置引导程序，其起始地址应该是 `0x7c3e`。又比如做 `init_monitor` 的时候，用户程序的学号没了，这是因为给 `ds` 指定了错误的段地址（本应 `0x800`，写成 `0x8000`）。

6.3 未解决问题

用 BIOS 中断输出字符串以后光标的位置不对，设遍所有参数都不对。结合不清屏的情况来看，光标位置总是位于 bochs 所有输出信息的最后。这点仍需长时间调试观察。

无法使用小键盘。小键盘的数字键跟横着的数字键是不同的效果，这个坑了我一晚上。后来观察 bochs 的键盘配置文件里是没有小键盘的，同学提醒是 BIOS 对于小键盘有特殊编码。

参考文献

[1] 李忠等,x86汇编语言：从实模式到保护模式,北京,电子工业出版社,30页

[2] 百度搜索“BIOS中断服务”“INT 16H”

[3] 赵永生,汇编语言返回DOS系统方法,河北理工学院学报（社会科学版）,2003年8月

[4] How does DOS load a program into memory,https://stackoverflow.com/questions/3715618/how-does-dos-load-a-program-into-memory

[5] 李忠等,x86汇编语言：从实模式到保护模式,北京,电子工业出版社,131页