实验二：加载用户程序的监控程序

18340225 钟婕

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

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、将自己实验一的引导扇区程序修改为3-4个不同版本的COM格式程序，每个程序缩小显示区域，在屏幕特定区域显示，用以测试监控程序，在1.44MB软驱映像中存储这些程序。

4、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。

5、设计一种简单命令，实现用命令交互执行在1.44MB软驱映像中存储几个用户程序。

6、编写实验报告，描述实验工作的过程和必要的细节，如截屏或录屏，以证实实验工作的真实性

实验内容

1、 将自己实验一的引导扇区程序修改为一个的COM格式程序，程序缩小显示区域，在屏幕第一个1/4区域显示，显示一些信息后，程序会结束退出，可以在DOS中运行。在1.44MB软驱映像中制定一个或多个扇区，存储这个用户程序a。

相似地、将自己实验一的引导扇区程序修改为第二、第三、第四个的COM格式程序，程序缩小显示区域，在屏幕第二、第三、第四个1/4区域显示，在1.44MB软驱映像中制定一个或多个扇区，存储用户程序b、用户程序c、用户程序d。

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。程序可以按操作选择，执行一个或几个用户程序。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。

4、在映像盘上，设计一个表格，记录盘上有几个用户程序，放在那个位置等等信息，如果可以，让监控程序显示出表格信息。

5、拓展自己的软件项目管理目录，管理实验项目相关文档

实验环境与工具

1、 实验环境：

1） 实验运行环境：Windows10

2） 虚拟机软件：VMware Workstation

2、 实验工具

1） 汇编编译器：NASM

2） 文本编辑器：Visual Studio 2017

3） 软盘操作工具：WinHex

实验方案与过程

- ## 总体思路

  由于实验一已经安装了无操作系统的裸机，因此本次实验可以直接使用实验一中安装的虚拟机，在此不赘述安装过程。将本次实验分为监控程序的设计与用户程序的设计两部分，监控程序负责根据操作选择加载相应的用户程序并执行，执行完成之后返回监控程序，再次进行操作选择。下面分别对这两部分的实验原理与设计过程进行描述。

- ## 监控程序的设计

  监控程序在本次实验中的主要职责就是：根据操作选择，正确无误地加载相对应的用户程序至内存，并跳转至内存相应位置，执行该用户程序。执行完成之后，要能够返回监控程序，并继续进行操作选择。

  考虑到每次执行完用户程序之后需要回到监控程序再次进行操作选择，因此通过查阅资料了解到BIOS调用中有清屏功能，因此学习使用该功能，方便在监控程序与用户程序之间的界面切换。

  具体如下：

```
;使用int10h的向上滚屏即清窗口功能
   mov ah,06h ;入口参数，功能号：ah=06h-向上滚屏，ah=07h-向下滚屏
   mov al,0h ;滚动行数（0-清窗口）
   mov ch,0 ;窗口的左上角位置（x坐标）
   mov cl,0 ;窗口的左上角位置（y坐标）
   mov dh,12 ;窗口的右下角位置（x坐标）
   mov dl,39 ;窗口的右下角位置（y坐标）
   mov bh,7 ;空白区域的缺省属性
   int 10h ;中断号
```

  在监控程序开始时，先初始化各个寄存器，将DS、ES寄存器的值赋值为CS的值，接着就调用上述的清屏功能。而为了方便用户了解调用各个用户程序的操作，设置了菜单内容，并在监控程序中显示，此时调用BIOS的10H号功能显示字符串即可，不过为了在监控程序与用户程序切换时不会互相影响其他区域的程序，达到其他三个程序可以同时出现的效果，因此仅将监控程序显示占领第一个用户程序的区域，将提示信息显示在第一个区域。

  特别注意地是，根据实验要求，要将此监控程序作为引导扇区程序，因此控制监控程序地大小在512字节之内，而且再在最后两个字节填充可引导标志：0X55、0XAA

  具体显示操作与菜单内容如下：

```
;调用int10h显示操作系统启动的相关提示信息
    mov bp,Message;偏移地址，采用段地址：偏移地址形式表示串地址
    mov ax,1301h ;ah=13h为功能号，al=01h指光标位于串尾
    mov bx,0007h ;bh=00h为页码，bl=07h为显示属性：黑底白字
    mov dx,0000h ;dh为行号，dl为列号
    mov cx,MessageLength
    int 10h
```

```
Message:
    db "Hello, welcome to My OS!",0AH,0DH ;字符数组，最后为换行符
    db "Here is the start Memu.",0AH,0DH,"Input ' ' to exit
UP.",0AH,0DH,"Please Enter the number:",0AH,0DH
    db "1.square",0AH,0DH
    db "2.single stone",0AH,0DH
    db "3.double stone",0AH,0DH
    db "4.sand clock",0AH,0DH
    db "5.DIY a special route.",0AH,0DH,"Please input '0' after every
run!",0AH,0DH
    db "6.display the table.",0AH,0DH
    MessageLength  equ ($-Message) ;将此监控程序作为首扇区引导程序
    times 510-($-$$) db 0 ;用‘0’填满首扇区剩余位置
    db 0x55,0xaa ;可引导标志
```

在做完清屏工作以及提示任务之后，接下来就需要根据输入进行调用用户程序地选择。通过查看相关博客等资料，发现BIOS调用中有从输入一个字符的功能，因此利用这个功能调用结合上述的菜单信息，可以进行用户程序的选择！

输入一个字符是06H中断号调用，功能号为0H，输入的字符存放在AL寄存器中。根据输入的字符值，借鉴实验一的代码，用类似于Switch语句，依次判断、选择。

```
;输入选择
Input:
    mov ah,0h
    int 16h ;调用16h号中断输入一个字符，ah为功能号，输入的字符存入al中
    cmp al,'0'
    jz Run
    cmp al,'1'
    jz UP
    cmp al,'2'
    jz UP
    cmp al,'3'
    jz UP
    cmp al,'4'
    jz UP
    cmp al,'5'
    jz DIY
    cmp al,'6'
    jz UP
    jmp Input ;确保输入的是0、1、2、3、4、5、6中的一个
```

根据输入选择之后，跳转到UP程序段，这段程序的功能主要是读取对应扇区的内容至内存相应位置，并跳转至内存相应位置执行。由于这四个COM格式的用户程序，我将之存放在第2个扇区~第5个扇区，因此根据输入的选择需要判断读取哪一个扇区。

加载软盘中存放相对应用户程序的扇区至内存中，需要用到BIOS的功能调用的int 13h功能，读取指定扇区至内存指定位置。为了代码编写的简洁性，定义OffsetofUP为8100H（监控程序中cs为0h，将初始偏移量ip设为8100H），代表用户程序加载至内存的偏移量，无论加载四个用户程序的哪一个，在int 13h读取扇区时均将其加载到cs：bx的内存位置，其中cs为统一的段地址，而bx则将其设定为数据常量OffsetofUP。

读取扇区结束之后，程序要跳转到用户程序中执行，由于用户程序是COM格式，规定初始偏移量为100h，而监控程序将用户程序加载到了0：8100h位置，因此在用户程序中，cs段基址并不为0h，而应该为800h，因此在监控程序利用jmp指令跳转到用户程序时，需要用段前缀形式描述跳转地址为：800h：100h，这样在用户程序中，段基址就为800h，程序的初始偏移量就为100h了。

读取软盘扇区需要BIOS的13h号功能调用，因此需要理解13h功能调用的各种入口参数的作用，具体如下：

AH:为功能号02H

AL：为读取的扇区数

DL：为驱动器的盘号（其中0和1表示软盘，80H和81H表示硬盘或U盘）

DH：磁头号

CH：柱面号的低8位

CL：0~5位为起始扇区号（特别注意扇区号从1开始），6~7位位硬盘柱面号高2位

ES：BX：为读入数据在内存中的存储位置

```
;调用int 13h读扇区加载用户程序至内存相应位置
;为了简便，将第一个用户程序存放在软盘的第二个扇区，以此类推
UP:
    sub al,47  ;输入的字符存放在al中，对应字符ASCII码值与实际扇区号相差47
    mov cl,al  ;读的起始扇区号，扇区号是从一开始的
    mov ax,cs
    mov es,ax ;置es与cs相等，为段地址
    mov bx,OffsetofUP ;数据常量，代表偏移地址，段地址:偏移地址为内存访问地址
    mov ah,02h ;功能号02h读扇区
    mov al,1   ;读入扇区数
    mov dl,0   ;驱动器号，软盘为0，U盘和硬盘位80h
    mov dh,0   ;磁头号为0
    mov ch,0   ;柱面号为0
    int 13h    ;中断号
    jmp 800h:100h ;第一个用户程序已经加载到内存0：8100h位置，由于时COM格式要写成800h：100h形式
```

另外，注意到实验内容中要求设计一种命令，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。因此设计为‘5’字符对应的DIY功能，考虑用一个字符数组存储用户输入的4个字符，即这四个用户程序的执行顺序。但在实验初期，我仅仅利用数组的元素值来控制顺序，发现不行，原因在于当监控程序跳转到用户程序执行之后返回时，并不会保存有原来监控程序执行值存储在数组内的值，相反数组中是初始值，因此这达不到效果。

冥思苦想一番之后，我想到利用栈来保存这四个顺序，在输入‘5’并输入四个字符之后，便将这四个字符逆序入栈，然后出栈一个，调用读扇区程序段UP并跳转至内存相应位置执行。执行完成之后，回到监控程序，用户输入‘0’提示现在仍处于按指定顺序执行用户程序的过程，因此输入‘0’之后，监控程序再次出栈一个字符，调用相应字符对应的用户程序。依次进行，直至4个指定顺序的用户程序执行完成。

```
;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。
DIY:
```

```
    mov ah,0h
    int 16h ;调用16h号中断输入一个字符，ah为功能号，输入的字符存入al中
    mov byte[arr],al
    mov ah,0h
    int 16h ;调用16h号中断输入一个字符，ah为功能号，输入的字符存入al中
    mov byte[arr+1],al
    mov ah,0h
    int 16h ;调用16h号中断输入一个字符，ah为功能号，输入的字符存入al中
    mov byte[arr+2],al
    mov ah,0h
    int 16h ;调用16h号中断输入一个字符，ah为功能号，输入的字符存入al中
    mov byte[arr+3],al
    mov bh,0
    mov bl,byte[arr+3]
    push bx;逆序依次入栈
    mov bl,byte[arr+2]
    push bx
    mov bl,byte[arr+1]
    push bx
    mov bl,byte[arr]
    push bx
    jmp Run
```

```
Run:
    pop ax;出栈
    jmp UP;跳转至读扇区程序段，读取对应扇区调用对应用户程序
```

特别说明，输入'0'字符是为了提示系统现在仍处于按照某种指定顺序执行用户程序的过程，因此在上述Input程序段中，用户输入'0'，程序会跳转至Run执行，继续出栈、加载用户程序。

当按照某种指定顺序的用户程序执行完成之后，回到Input程序段，继续进行操作选择。

根据上述所述，即可以完成监控程序的设计，实现监控程序按照操作选择加载用户程序的功能。

- ## 用户程序的设计

根据实验要求，用户程序是将实验一中的引导扇区程序修改为一个COM格式的程序，并对实验一的程序做出一定的修改，形成四个COM格式的程序，这四个程序将显示区域分成四部分，每个程序在一部分上显示运行。我将屏幕分为大致四部分为:(x:0~12,y:0~39);(x:0~12,y:40~79);(x:13~24,y:0~39);(x:13~24,y:40~79).

由于用户程序是COM格式，所以每个用户程序前都要有语句org 100H，将程序加载到内存中初始偏移量为100H的位置执行。在监控程序的设计中，将读取每个用户程序扇区时加载到内存位置的偏移量设为统一的，因此每个用户程序都需要加载到这个内存地址上。

下面分别叙述这四个用户程序:

1. ### 用户程序1

   用户程序1主要用于显示字符运动的矩形轨迹并且在矩形中央显示个人的信息（包括学号与姓名），根据本人划分的四个区域，用户程序1在第一个区域（x:0~12,y:0~39)上显示。

   由于在监控程序与第一个用户程序的切换过程中，也涉及到清屏的操作，因此参照监控程序的清屏功能调用，对显示区域第一个区域13×40范围进行清屏。

   然后，利用BIOS的10h功能调用显示字符串，cx寄存器存放显示字符串长度、bl存放显示属性、dx存放行列位置，bp存放字符串的偏移地址，段基地址有es寄存器给出。具体代码如下:

```
;调用int 10h实现显示字符串功能
show:
   mov ax,cs
   mov es,ax
   mov ax,myName
   mov bp,ax;es:bp=串基地址:偏移地址=串地址
   mov cx,14;串的长度
   mov ax,1301h;ah=13h显示字符串;al=01h光标位于串尾
   mov bx,001dh;bl为1dh 蓝底浅品红无闪烁;bh为00h 页号为0
   mov dx,060Ah;列位置
   int 10h;调用10h号中断
   mov ax,myNumber
   mov bp,ax;es:bp=串基地址:偏移地址=串地址
   mov cx,15;串的长度
   mov ax,1301h;ah=13h显示字符串;al=01h光标位于串尾
   mov bx,001dh;bl为1dh蓝底浅品红无闪烁;bh为00h页号为0
   mov dx,070Ah;列位置
   int 10h;调用10h号中断
```

接下来说明，字符按矩形运动轨迹部分。类似于实验一的汇编，只不过字符运动方向需要更改为上下左右四个方向，且字符运动区域缩小为显示区域的第一个部分，其余部分与实验一的汇编高度相似。下面摘取向右运动的部分代码：

```
Rt:
   inc word[y]
   mov bx,word[y]
   mov ax,36;与边界作比较
   sub ax,bx
   jz Rt2Dn;到达边界，变换方向
   jmp display

Rt2Dn:
   mov word[y],35
   mov byte[initial],Dn_;变换方向
   jmp display;显示字符
```

最后需要考虑，用户程序如何返回监控程序的问题。在监控程序的设计中对于操作选择的学习过程了解到BIOS输入一个字符的功能调用，因此在用户程序返回监控程序的设计中，也利用输入一个字符作为返回标志。当用户输入一个空格时，程序用jmp 0:7C00H指令返回到监控程序，若输入的不是空格则用户程序继续执行。因为用户程序中的cs寄存器的值与监控程序的不同，因此使用jmp指令时，要写明段前缀。

具体代码如下所示：

```
;程序的终止由用户输入空格键终止
;调用int 16h输入一个字符判断
Judge:
  mov ah,01h
  int 16h;输入一个字符，存放在al中
  mov bl,20h;空格的ASCII码值为32
  cmp al,bl;比较输入的是否为空格
  jz Quit;若是则返回监控程序
jmp Loop;不是，则继续执行
Quit:
jmp 0:7C00H;返回监控程序
```

2. 用户程序2

用户程序2是与实验一相同的字符45°弹射的程序，只不过将字符的的运动轨迹的显示区域缩小为四分之一个区域，即第二个部分中（x:0~12,y:40~79）。因此只需要在实验一的汇编代码更改弹射时方向变化判断时的边界值即可，同时类似于用户程序1，为了整洁美观，需要加入清屏操作（清除本区域的屏幕内容即可）以及根据输入字符判断程序返回与否的操作。由于这两个操作在用户程序1中已经说明，且本程序也在实验一中详细叙述了，因此在此不再赘述。

下面为方向转换的部分代码：

```
DnRt:
    inc word[x]
    inc word[y]
    mov bx,word[x]
    mov ax,13 ;边界判断
    sub ax,bx
    jz dr2ur
    mov bx,word[y]
    mov ax,80 ;边界判断
    sub ax,bx
    jz dr2dl
    jmp display
;方向转换
dr2ur:
   mov word[x],11
   mov byte[initial],Up_Rt
   jmp display
;方向转换
dr2dl:
   mov word[y],78
 mov byte[initial],Dn_Lt
   jmp display
```

3. 用户程序3

用户程序3是在用户程序2也即实验一的基础上拓展实现的，即实验一中设计了单条字符的运动轨迹，而在用户程序3中拓展为两条字符的运动轨迹。

改进方面主要在于初始时设定这两个字符运动的起始位置的y坐标不同，用一个含两个元素的数组描述这两个y坐标。开始时，令第一个元素代表的y坐标设定的位置的字符先运动，然后在经历了（50000*580）时间之后，切换为数组第二个元素代表的y坐标进行一个时间单位的字符运动。特别注意的是，每次要记得将一次运动后的y坐标值写回相应的数组位置！虽然在程序设计中，这两个起始位置不同的字符是交替运动的，但由于人的视觉暂留等原理，看起来是两条同时运动的字符轨迹。因此完成了实验设计。

下面展示改进部分的代码：

```
Loop:
    dec word[num]
    jnz Loop
    mov word[num],delay
    dec word[dnum]   ;产生50000*580的一次时间单位延迟
    jnz Loop
```

```
    mov word[num],delay
    mov word[dnum],ddelay
    mov ax,0
    cmp ax,word[standard];standard的值为0是第一条轨迹，为2第二条轨迹
        jz ch1
    jmp ch0
;选择第二条轨迹
ch0:
    mov word[standard],0;更改standard的值
    mov ax,[arr+2];当前运动字符为第二条轨迹
    mov word[y],ax ;更改y坐标的值
    jmp Swi
;选择第一条轨迹
ch1:
    mov word[standard],2
    mov ax,[arr];当前运动字符为第一条轨迹
    mov word[y],ax
    jmp Swi
```

除了上述的改进部分，其余方向跳转的变化、判断均与用户程序2的设计类似。当然同样需要加上BIOS功能调用中的清屏操作以及输入字符判断是否返回监控程序的操作，在此就不一一赘述了。

4. ## 用户程序4

用户程序4是在实验一的基础上，思考变化字符的运动轨迹的形状，除了矩形、45°弹射之外考虑三角形等形状，但由于三角形显得不够美观、对称，因此选择实现用两个三角形组成的沙漏形状的运动轨迹。其实不论是什么形状，设计的思想都与实验一十分类似，都是在间隔一个时间单位之后，更改字符的x、y坐标，并且判断是否到达形状的边界，如果到达根据形状的特点改变方向，并将在该位置的字符显示出来即可。

沙漏图像的轨迹设计分为右、左下、左上三个方向，初始时设定为向右运动，在向右到达y坐标边界时需要判断x坐标是否达到边界，如果x、y均到达边界那么方向转换为左上方向，如果y坐标到达边界而x坐标没有到达边界那么方向转换为左下方向。基于上述判断即可，至于间隔时间单位的选择以及显示x、y坐标位置的字符均与实验一中的类似。

下面选取方向转换的部分代码：

```
;向右运动
Rt:
   inc word[y]
   mov bx,word[y]
   mov ax,56
   sub ax,bx ;判断y坐标是否到达边界
   jz Rt2DnLt
   jmp display

Rt2DnLt:
   mov bx,word[x]
   mov ax,24
   sub ax,bx ;判断x坐标是否到达边界
   jz Rt2UpLt ;若是，向右转换为左上
   mov word[y],55
   mov byte[initial],Dn_Lt ;否则为左下
   jmp display

Rt2UpLt:
```

```
  mov word[y],55
    mov byte[initial],Up_Lt
  jmp display
```

与前面三个用户程序类似，用户程序4也需要进行清屏操作与返回判断操作。

- ## 用户程序显示表格信息

根据实验要求，需要设计一个表格，记录软盘上有几个用户程序，放在软盘的哪个位置等信息，并让监控程序能够显示表格信息。基于上述要求，其实就是再设计一个“用户程序”，这个程序显示前四个用户程序的位置等相关信息，监控程序可以像加载用户程序一样加载这个程序，并能够返回。

根据上述分析，我简单理解为，创建一个程序，这个程序的组织其实很类似于前面四个用户程序。需要利用BIOS的10h显示字符串功能调用显示表格信息，并类似于前四个用户程序判断输入字符是否为空格的方式，用jmp指令返回监控程序中。同样地，监控程序也能够通过输入‘6’选择字符，选择跳转到此程序。

```
showInfo:
    ;调用int10h显示有关用户程序的相关提示信息
    mov bp,Message;偏移地址，采用段地址：偏移地址形式表示串地址
    mov ax,1301h ;ah=13h为功能号，al=01h指光标位于串尾
    mov bx,0007h ;bh=00h为页码，bl=07h为显示属性：黑底白字
    mov dx,0000h ;dh为行号，dl为列号
    mov cx,MessageLength
    int 10h
```

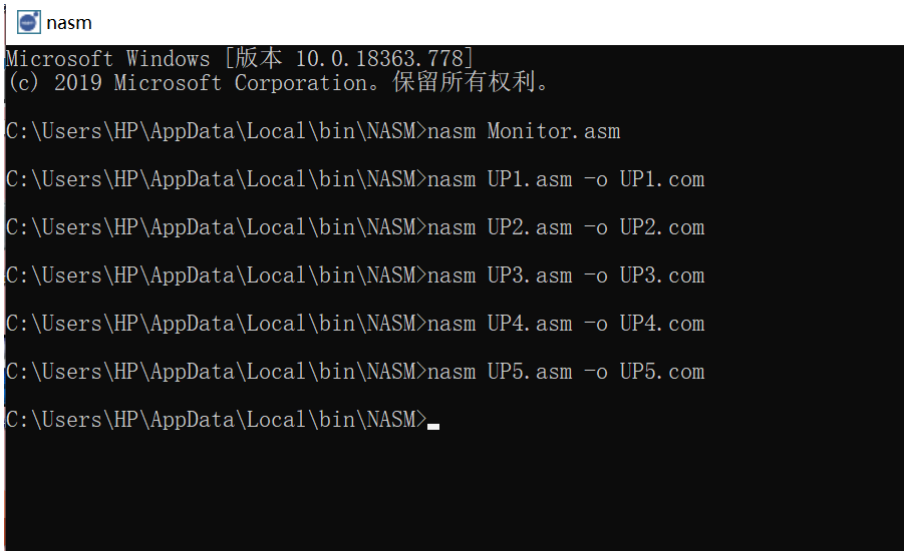
```
Message:
    Info:db 'There are four programs of user!',0Dh,0Ah;用户程序数量
    UP1name:db 'Square'
    UP1addr:db '  addr:0200h~0400h:second section',0DH,0AH;第一个用户程序的名称、位置
    UP2name:db 'Single stone'
    UP2addr:db '  addr:0400h~0600h:third section',0DH,0AH;第二个用户程序的名称、位置
    UP3name:db 'Double stone'
    UP3addr:db '  addr:0600h~0800h:fourth section',0DH,0AH;第三个用户程序的名称、位置
    UP4name:db 'Sand clock '
    UP4addr:db '  addr:0800h~0A00h:fifth section',0DH,0AH;第四个用户程序的名称、位置
    MessageLength equ ($-Message)
```

其中，本程序返回监控程序的判断、跳转方式与前四个用户程序相同，而且本程序的显示为了整洁美观需要对全屏进行清屏，在此就不赘述。

- ## 实验操作过程

1. ### NASM编译

打开NASM，在框内输入指令，对于监控程序Monitor.asm，由于不限定二进制文件格式，因此输入nasm+输入文件名即可，生成同名的二进制文件。而对于用户程序要生成COM格式程序，因此需要输入nasm + 输入文件名 -o + 输出文件名，生成COM格式的二进制文件。



Monitor	2020/5/4 21:27	文件	1 KB
Monitor.asm	2020/5/4 18:30	Assembler Source	3 KB

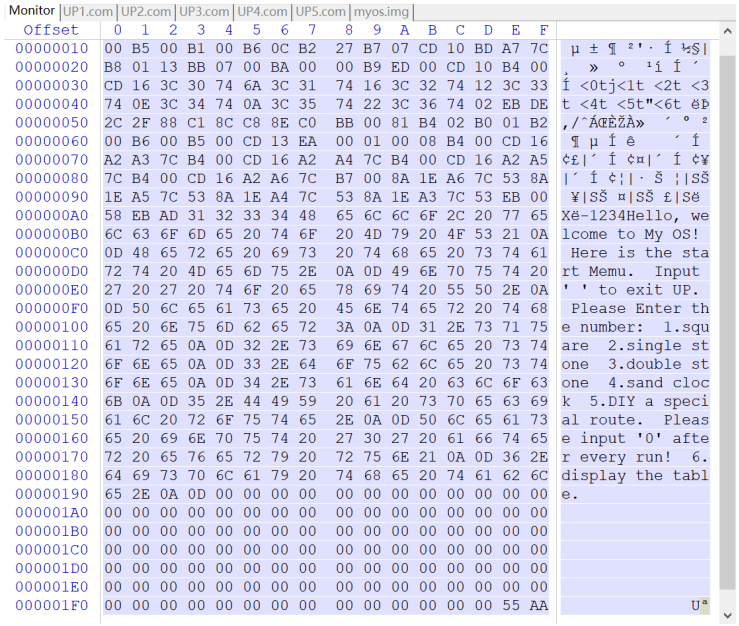
UP1.asm	2020/5/4 17:42	Assembler Source	4 KB
UP1.com	2020/5/4 21:27	MS-DOS 应用程序	1 KB
UP2.asm	2020/5/4 17:42	Assembler Source	4 KB
UP2.com	2020/5/4 21:27	MS-DOS 应用程序	1 KB
UP3.asm	2020/5/4 17:42	Assembler Source	4 KB
UP3.com	2020/5/4 21:27	MS-DOS 应用程序	1 KB
UP4.asm	2020/5/4 17:42	Assembler Source	3 KB
UP4.com	2020/5/4 21:28	MS-DOS 应用程序	1 KB
UP5.asm	2020/5/4 17:42	Assembler Source	2 KB
UP5.com	2020/5/4 21:28	MS-DOS 应用程序	1 KB

2. winHex工具编辑软盘

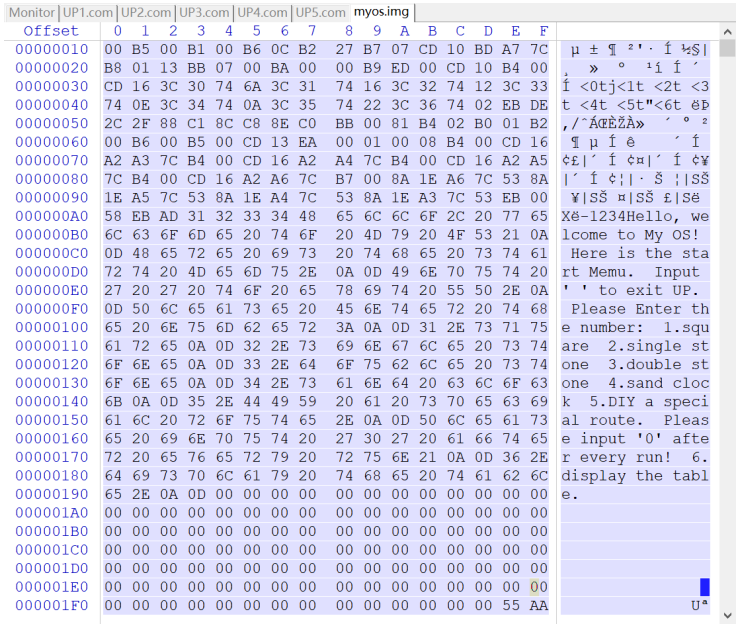
由于在上一次实验中，本人是在Linux环境使用dd命令进行1.44MB虚拟软盘映像的生成，因此在本次实验中中，想要更换一种方式，利用WinHex编辑软盘，帮助了解一下WinHex更多的功能与操作。

利用winHex工具打开上述生成的各个二进制文件，先将“Monitor”二进制文件内容拷贝到首扇区，然后按照第一个用户程序UP1存放在第二个扇区（0200H~0400H），第二个用户程序UP2存放在第三个扇区（0400H~0600H）的规则，以此类推将UP1~UP4的二进制文件拷贝到对应扇区中。

拷贝操作的过程如下所示：



上图为“Monitor"监控程序二进制文件内容，利用操作：选中需要复制内容，右键——编辑——复制选块——正常，完成复制操作。

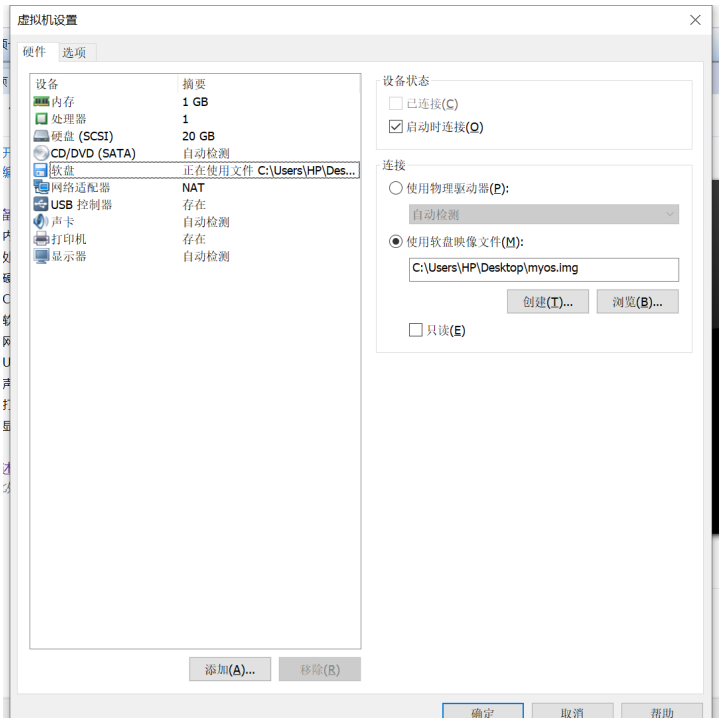


上图为1.44MB软盘，将上述”Monitor“的内容复制到此软盘的首扇区，操作为：右键——剪贴板数据——粘贴。

其余用户程序的拷贝操作类似。

3. 软盘启动裸机

将上述生成的监控程序二进制文件、COM格式文件依次按照扇区分配复制到1.44MB软盘的对应位置并保存后，将1.44MB软盘作为引导盘启动虚拟机。

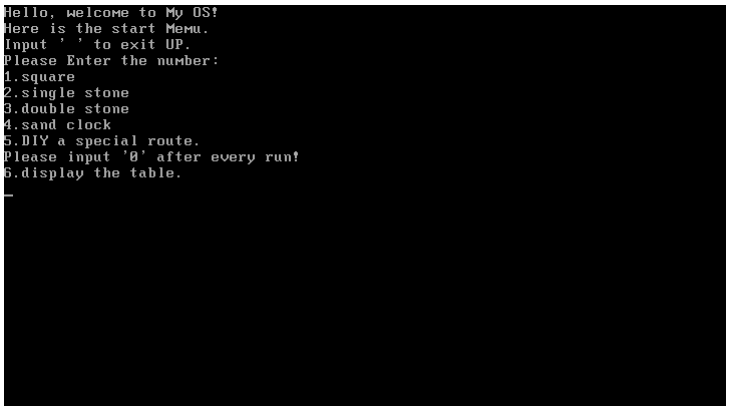


- ## 实验运行结果

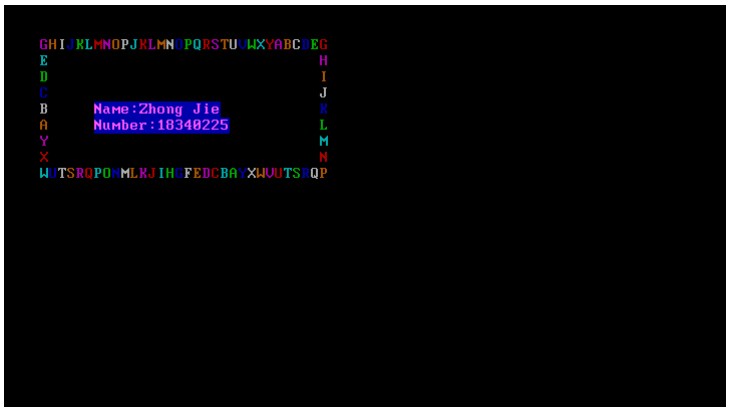
成功启动虚拟机之后，虚拟机就会将程序加载到虚拟机的内存空间中运行，可以观察到监控程序的界面，接着依次按照菜单，输入选择运行不同的用户程序，并输入空格返回监控程序，另外测试按照某种指定顺序执行用户程序、显示用户程序相关信息等操作。

程序运行结果如下：

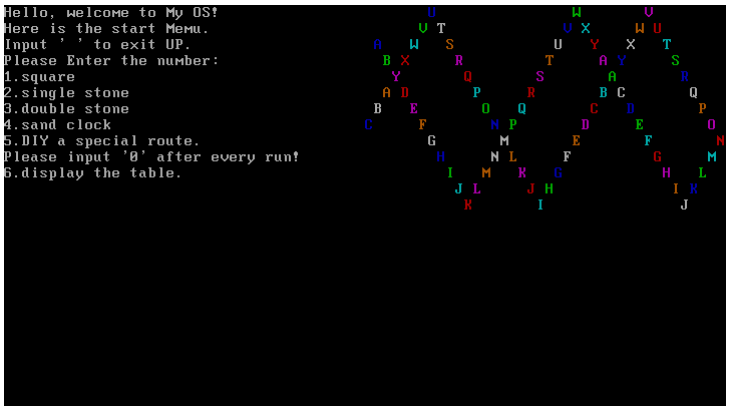
监控程序界面：



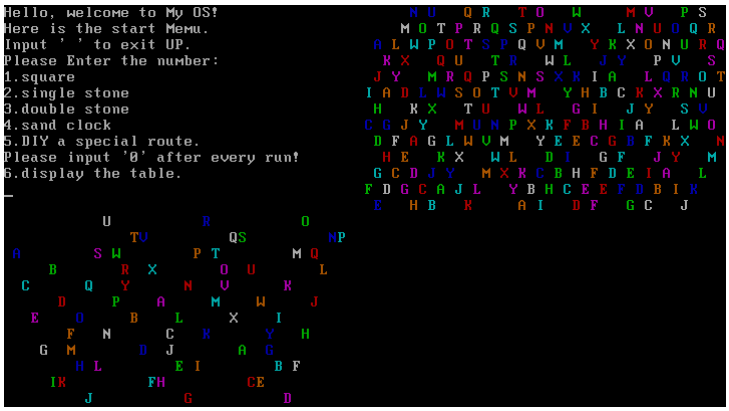
输入‘1’字符，显示第一个用户程序：



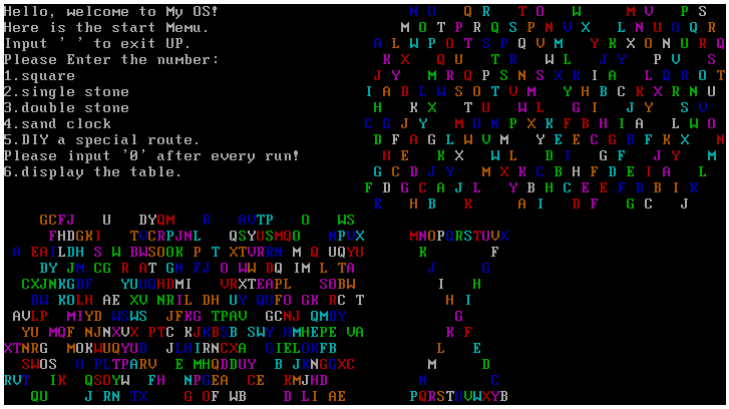
输入空格可以回到监控程序，接着输入‘2’字符，显示第二个用户程序：



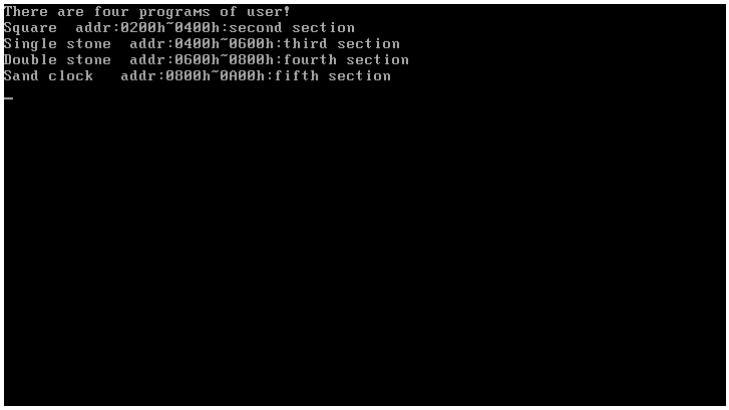
输入空格返回监控程序，接着输入‘3’字符，显示第三个用户程序：



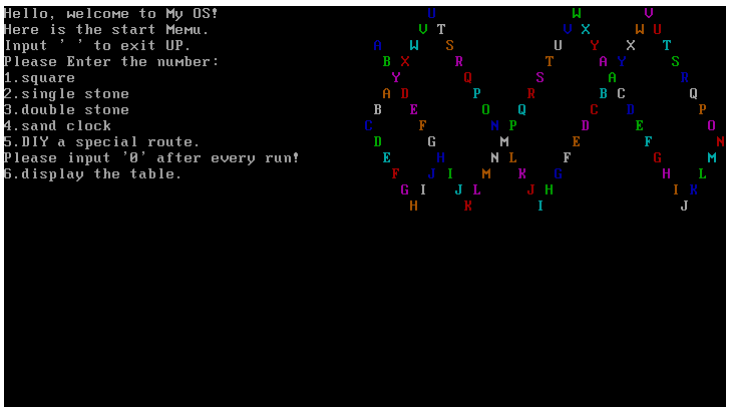
输入空格返回监控程序，接着输入‘4’字符，显示第四个用户程序：



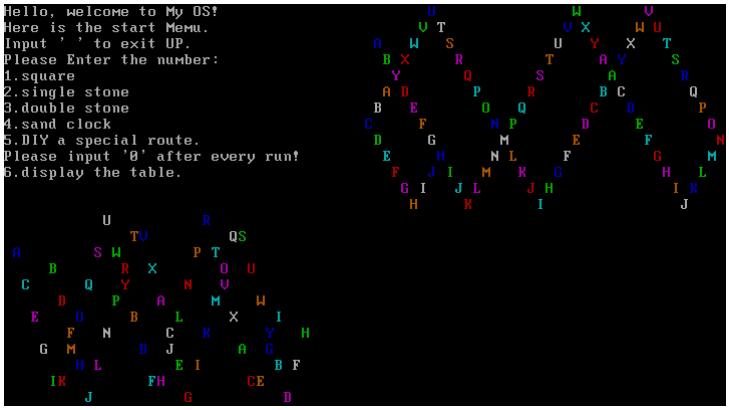
输入空格返回监控程序。接着输入‘6’字符查看各个用户程序的位置信息：



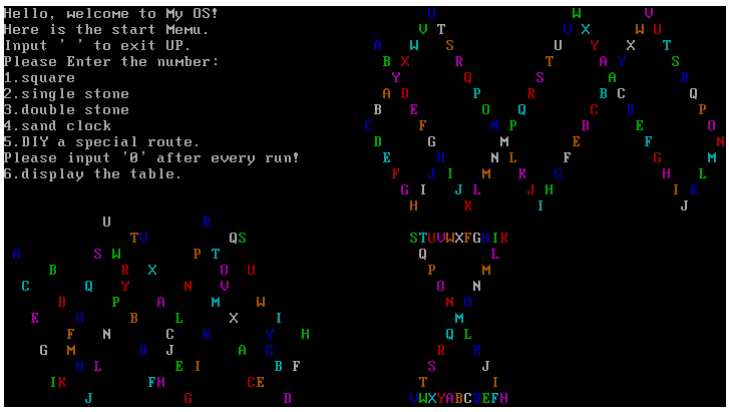
输入空格返回监控程序，接着测试按照输入指定顺序执行用户程序。首先输入‘5’字符，接着输入指定的顺序，在此输入‘2”3”4”1’作为测试。显示第二个用户程序：



输入空格结束第二个用户程序的执行，返回监控程序。再输入‘0’，表明仍处于自定义顺序执行中，显示第三个用户程序：



输入空格结束第三个用户程序的执行，返回监控程序，输入‘0’，表明继续按指定顺序执行，显示第四个用户程序：



输入空格结束第四个用户程序的执行，返回监控程序，输入‘0’，表明继续按指定顺序执行，显示第一个用户程序：

至此，完成所有功能测试，实验运行结果与预期相符合。

实验心得

本次实验主要是学会设计一个能够加载用户程序的监控程序，由于实验要求中的用户程序即是实验一中引导扇区程序的更改版本，因此对于用户程序的设计，在本次实验过程中相对轻松。而对于监控程序的设计，虽然有了老师提供的参考代码，整个编程思路已经更加明晰了，但是在具体编程的时候，仍然遇到了许多大大小小的问题，比如说在监控程序与用户程序切换时，显示区域的整洁美观问题。为了让用户更好了解本程序的使用，我在监控程序设置了菜单界面，但为了美观我选择将其在显示区域划分的第一个部分显示，然后在每个加载用户程序执行时，用户程序会执行自己所属范围内的清屏操作，返回监控程序时监控程序也会执行自己所在范围内的清屏操作。而对于显示表格信息的程序则为了美观执行了80×25范围内的清屏操作。关于如何显示美观这一点，我思考了很久，原本设计的是每个用户程序都对80×25的全屏进行清屏，但后来觉得这样就不能突出把屏幕分成四个部分的作用了，因此在之后选择让每个用户程序仅对自己所属部分进行清屏，监控程序也是。

除此之外，关于COM格式文件的问题也困扰我许久。在实验一中，我们在引导扇区程序程序开头加上了“org 7c00H”的语句，这是让计算机将程序加载到内存初始偏移量为7c00h的地址上，这时的段地址cs是0h。但由于本次实验限定了用户程序为COM格式程序，并将用户程序加载到0：8100h的地址上，而COM格式程序规定程序加载到内存的初始偏移量为100h，因此在用户程序中，段地址cs应该为800h，我需要实现监控程序与用户程序之间段地址的变换。在冥思苦想之后，我终于搞清楚这些关系，0：8100h与800h：100h其实是同一地址，那么在监控程序中我们以0：8100h的形式来描述读取用户程序对应扇区加载到内存的位置，但在读取扇区之后，监控程序跳转到用户程序执行时的jmp指令，需要用段前缀的形式指明800h：100h的地址，这样在跳转到用户程序时的cs值就是800h；同理，在用户程序中我们以800h：100h形式来描述，当用户程序跳转回到监控程序时，使用jmp指令也要用段前缀形式指明0：7c00h地址，这样在回到监控程序时的cs值为0h。

在想明白了上述的问题之后，我再结合网上查找的BIOS功能调用的相关资料，正确使用各种寄存器，那么本次实验就迎刃而解啦。当然，本次实验还有一些不足之处，如果能让指定顺序的命令更简洁一点就更加完美了。

总而言之，一次实验就是一次收获，一次充实自己知识的过程。希望自己可以在之后的实验中越做越好！

参考文献

1. 《BIOS中断大全》:https://blog.csdn.net/weixin_37656939/article/details/79684611
2. 《PSP程序段前缀》:https://blog.csdn.net/ruyanhai/article/details/7177904

3. 《寻址方式》:https://blog.csdn.net/at_the_beginning/article/details/85156577