



中山大學
SUN YAT-SEN UNIVERSITY

《操作系统原理实验》

实验报告

(实验一)

接管裸机的控制权

学 院 名 称 : 数据科学与计算机学院

专业 (班级) : 18 级计算机科学与技术

学 生 姓 名 : 钟婕

学 号 : 18340225

时 间 : 2020 年 4 月 23 日

成绩：

实验一：接管裸机的控制权

一. 实验目的

1. 学会搭建实验和应用环境
2. 掌握接管裸机控制权的操作

二. 实验内容

1. 搭建和应用实验环境

虚拟机安装，生成一个基本配置的虚拟机XXXPC和多个1.44MB容量的虚拟软盘，将其中一个虚拟软盘用DOS格式化为DOS引导盘，用WinHex工具将其中一个虚拟软盘的首扇区填满你的个人信息。

2. 接管裸机的控制权

设计IBM_PC的一个引导扇区程序，程序功能是：用字符‘A’从屏幕左边某行位置45度角下斜射出，保持一个可观察的适当速度直线运动，碰到屏幕的边后产生反射，改变方向运动，如此类推，不断运动；在此基础上，增加你的个性扩展，如同时控制两个运动的轨迹，或炫酷动态变色，个性画面，如此等等，自由不限。还要在屏幕某个区域特别的方式显示你的学号姓名等个人信息。将这个程序的机器码放进放进第三张虚拟软盘的首扇区，并用此软盘引导你的XXXPC，直到成功。

三. 实验器材

PC机一台，VMware Workstation，NASM，WinHex。

四. 实验方案与过程

● 实验原理：

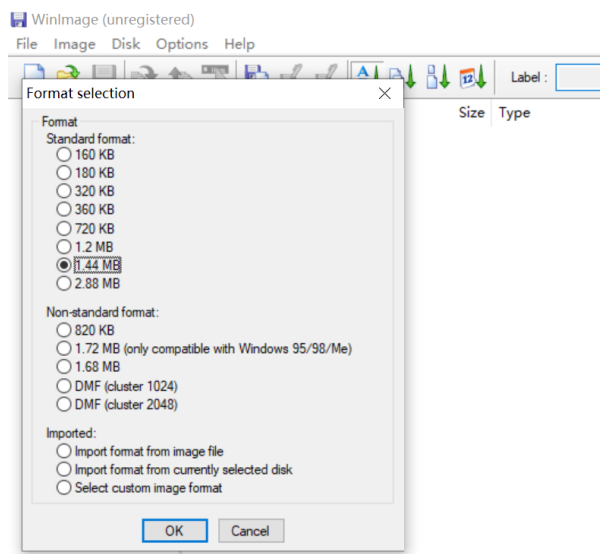
将本实验过程分成两部分：a、搭建和应用实验环境；b、接管裸机的控制权。

1、搭建和应用实验环境

在这部分的实验操作中，可使用Vmware Workstation生成一个基本配置的虚拟机，利用WinImage软盘镜像文件处理器生成多个1.44MB的虚拟软盘。可以使用WinHex工具以二进制文件形式将其中一个虚拟软盘的首扇区填满个人信息，也可以使用汇编语言先生成填满个人信息的ASM文件，用NASM编译器生成BIN文件，再用dd命令生成1.44MB的IMG文件。

操作过程如下所示:

打开WinImage, 创建多个1.44MB的虚拟软盘。



再用WinHex工具将其中一个虚拟软盘首扇区填满我的个人信息。

填充结果如下，以0xAA55结尾符合要求。

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
00000000	31	38	33	34	30	32	32	35	20	00	0A	5A	68	6F	6E	67	18340225	Zhong
00000010	20	4A	69	65	00	00	00	00	00	00	00	00	00	00	00	00	Jie	
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA		

最后得到大小为1.44MB的PersonalInfo.img。

2、接管裸机的控制权

在这部分的实验中，首先需要了解BIOS的中断调用以及字符显示的相关原理。

A. BIOS的中断调用

BIOS的10H提供了显示字符串的调用，功能号为13H，其中该中断服务程序的特定参数为：

AL：放置光标的方式

BL：字符的显示属性

BH：字符串的显示页号

CX：字符串的字节数

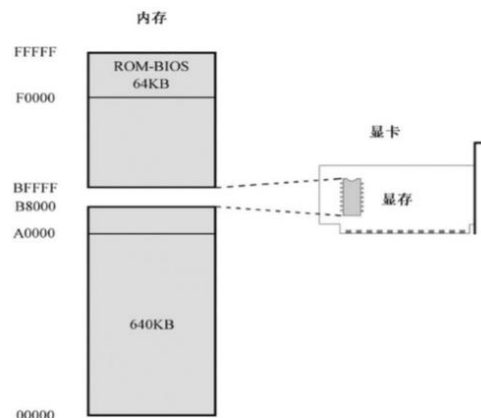
DH：行位置

DL：列位置

ES: BP：字符串的起始地址//ES为段基地址，BP为偏移地址

B. 显存和显卡

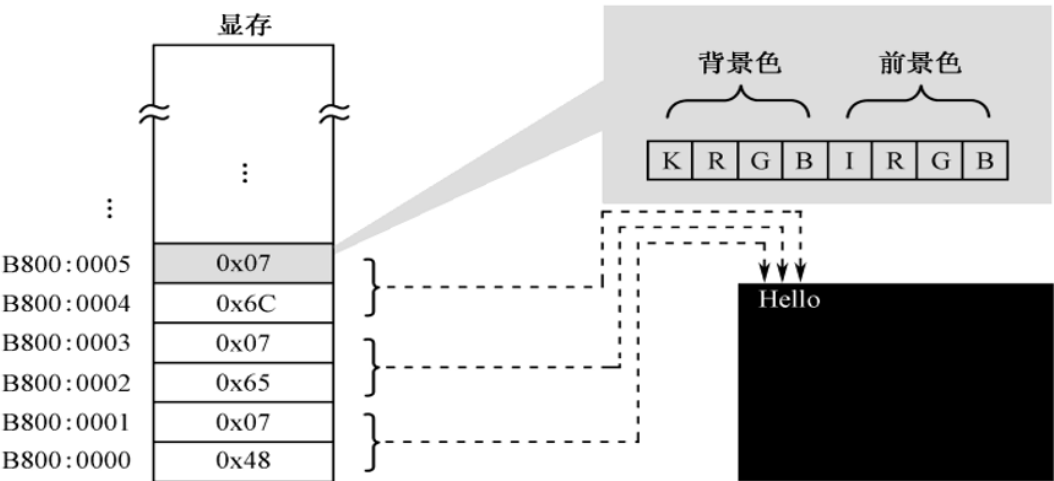
一直以来，在内存的地址结构中，B8000H~BFFFFH总共32KB的空间是留给显卡的，用来显示文本。而所有在PC机上使用的显卡，在加电自检以后都会把自己初始化到80×25的文本模式中。B8000H~BFFFFH是一个80×25的彩色字符模式的显示缓冲区，如下图所示。



当我们向这段显示缓冲区空间写入数据时，写入的内容将即刻出现在显示器上。在80×25模式下，显示器可以显示25行，每行80个字符，每个字符有256种显示属性。由ASCII码可知，我们需要一个字节来确定显示的字符，而对于每个字符的256种属性，我们也需要

一个字节来指定显示属性。因此，一个字符在显示缓冲区就要占两个字节即一个字的空间，分别存放字符的ASCII码和显示属性。在80×25模式下，一屏的内容在显示缓冲区就需要占4000个字节的空间大小。

为了访问显存，采用段基址：偏移地址的形式，段基址为B800H，偏移地址的范围为：0000H~FFFFH。显存与每个字符的对应关系如下图所示：



每个字符的显示属性有256种，具体如下所示：

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

● 汇编语言程序设计

根据实验要求将程序设计过程分成两部分：在屏幕某区域显示个人信息（学号和姓名）与编写引导扇区程序（显示字符运动轨迹）。下对这两部分分别叙述设计思路与过程。

1、显示个人信息

PersonnalInfo 段代码用来在屏幕的左上角显示个人信息。这段代码主要通过 ROM BIOS 的中断调用 10H 实现在屏幕显示字符串的功能。根据 BIOS 的中断可知，中断号

10H、功能号 13H 为显示字符串功能，因此 AH 赋值 13H，INT 10H 表示显示器调用。

;显示个人姓名 学号

PersonnalInf:

mov ax,myName	
mov bp,ax;	es:bp=串基地址:偏移地址=串地址
mov cx,9;	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,00h;	dh 行号=0h;dl 列号=0h
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,8;	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,0100h;	dh 行号=01h;dl 列号=00h
int 10h;	调用 10h 号中断

myName、myNumber 为声明与初始化字符串伪指令的标志，代表着相对于程序开始处的偏移地址，也是相对于段基地址的偏移地址，因此将它赋值给 BP 寄存器，CX 寄存器存储字符串字节数，AH 为 13H 代表功能号，AL 为光标位置，BH 为显示的页号、BL 为显示属性，此处为蓝底浅品红无闪烁，DX 为行号与列号位置。INT 10H 调用中断号为 10H 的中断服务程序。

2、显示字符运动轨迹

为了在屏幕上显示出字符运动的效果，每次运动需要有一定的时间间隔，因此定义计时器数据常量，控制画框运动速度。

delay equ 50000	;计时器延迟计数;用于控制画框的速度
ddelay equ 580	;计时器延迟计数;用于控制画框的速度

由于字符是不断在运动的，因此每次时间延迟的计算可以用一个循环实现，间隔指定的计数时间 (50000*580)，字符朝着一定方向运动一个单位。

Loop 段即为运动间隔的计算过程，间隔之后，让字符在相应运动方向运动。

Loop:

```
dec word[num]          ; 递减计数变量, 产生延迟一共50000*280
jnz Loop              ; 大于0时, 跳转
mov word[num],delay
dec word[dnum]
jnz Loop              ; 大于0时, 跳转
mov word[num],delay
mov word[dnum],ddelay
mov al,1
cmp al,byte[initial]   ; 类似于switch语句, 朝相应方向运动
    jz DnRt
mov al,2
cmp al,byte[initial]
    jz UpRt
mov al,3
cmp al,byte[initial]
    jz UpLt
mov al,4
cmp al,byte[initial]
    jz DnLt
jmp $
```

通过递减计数变量, 间隔延迟 50000*280, 利用类似于 switch 语句来判断字符的运动方向, 跳转到相应方向代码段执行字符运动。initial 为声明初始化运动方向的标志, 初始声明为 Dn_Rt(右下)方向, 每次运动之后判断是否更新 initial 内容 (即是否发生反射, 改变运动方向)。

下选取四个运动方向中的其中一个 DnRt(右下方向)说明运动过程的程序设计, 包括遇到边界反弹的情况。

不妨假设字符 A 从 (x,y) 开始向右下方运动, 那么下一时刻的位置确定, 将分为三种情况: 字符未碰触边界、字符碰触行边界、字符碰触列边界。这里, 行号的范围为 0~24, 列号的范围为 0~79。显而易见, 当字符未碰触边界时, 下一时刻的字符位置应该为 (x+1, y+1)。如果字符碰触到行边界, 即 x+1 后与 25 相等, 那么将发生反弹, 字符的运动方向将变为右上方向, 即程序跳转至 dr2ur 段执行。如果如果字符碰触到列边界, 即 y+1 后与 80 相等, 那么将发生反弹, 字符的运动方向将变为左下方向, 即程序跳转至 dr2dl 段执行。

而在发生反弹变换方向的代码中, 只需要将到达边界的相应参数重置, 同时更改 initial

中的内容即可。例如，在 dr2ur 段，字符运动到行边界，因此将 x 行坐标置为 24-1=23 即可，y 列坐标不变，而将 initial 的内容重置为右上方向。而在 dr2dl 段，字符运动到列边界，因此将 y 行坐标置为 79-1=78 即可，x 行坐标不变，而将 initial 的内容重置为左下方向。

具体代码如下所示：

```
DnRt:
    inc word[x]           ;坐标 x 行加一
    inc word[y]           ;坐标 y 列加一
    mov bx,word[x]
    mov ax,25
    sub ax,bx             ;比较行数是否到达边界
    jz dr2ur              ;到达则由右下方向变为右上
    mov bx,word[y]
    mov ax,80
    sub ax,bx             ;比较列数是否到达边界
    jz dr2dl              ;到达则由右下方向变为左下
    jmp display           ;未到达边界则显示字符
```

```
dr2ur:
    mov word[x],23        ;将x行坐标置为23
    mov byte[initial],Up_Rt ;将运动方向更改为右上
    jmp display           ;显示字符
```

```
dr2dl:
    mov word[y],78        ;将y列坐标置为79
    mov byte[initial],Dn_Lt ;将运动方向更改为左下
    jmp display           ;显示字符
```

确定了字符下一时刻的位置以及运动方向之后，则需要显示字符，在本次实验中即为 display 段中的内容。display 段代码向内存中某个地址空间写入需要显示的字符，则写入的内容将立即以给定的显示属性出现在屏幕中。我们需要根据当前字符的行号和列号计算出预期写入的显存地址。由于行号的范围为 0~24、列号的范围为 0~79，我们可以通过公式： $(\text{行号} \times 80 + \text{列号}) \times 2$ 得到预期的显存偏移地址。在确定了字符的显示属性之后，我们便可向计算得出的显存偏移地址与段基地址 B800H 组合的串地址空间写入相应内容。

在此处，为了凸显个性化的设计，我设计字符运动时字符为 26 个大写字母依次循环显

示，而字符的显示属性也是从 01H 到 07H 依次改变。color 为声明初始化字符显示属性的标志，初始化为 01H；char 为声明初始化字符的标志，初始化为 ‘A’，两者均为字节大小。

在每次显示时，写入的串地址为字大小的[gs:bp]的内存空间，ax 寄存器高位 ah 存放显示属性[color]，低位 al 存放显示的字符[char]。每次显示完成之后，将 color、char 的内容递增，递增后判断 color 的内容是否与 08H 相等，如果相等则跳转到 circle1 重置 color 的内容为 01H；同样，判断 char 的内容时候与 90 相等，如果相等则跳转到 circle2 重置 char 的内容为 65。这样即可实现个性化的设计。

具体代码如下：

```
display:
    mov ax,word[x]
    mov bx,80
    mul bx
    add ax,word[y]
    mov bx,2
    mul bx          ;计算显存偏移地址
    mov bp,ax
    mov ah,[color] ;显示属性
    mov al,byte[char] ;显示字符
    mov word[gs:bp],ax ;向相应显存地址写入
    add byte[color],1
    add byte[char],1 ;递增
    mov bl,8
    cmp byte[color],bl ;判断是否相等
    jz circle1         ;相等则跳转到circle1
    mov bl,90
    cmp byte[char],bl ;判断是否相等
    jz circle2         ;相等则跳转到circle2
    jmp Loop
```

```
circle1:
    mov byte[color],1 ;重置color的内容
    mov bl,90
    cmp byte[char],bl
    jz circle2
```

```
circle2:
    mov byte[char],65 ;重置char的内容
    jmp Loop
```

- 实验个性化改进

在完成上述单条字符运动轨迹之后，本人希望在单条字符运动的基础上做出一些改进。因此萌发出设计两条字符运动轨迹。

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。

下面列出一些代码的改动部分：

```
datadef:
    num dw delay
    dnum dw ddelay
    initial db Dn_Rt
    x dw 2
    y dw 3
    arr dw 3,8; 用一个arr数组来不断交替两条线路的y坐标
    standard dw 0; 用一个变量来记录此时运行的是哪一条轨迹，数组访问的下标
    color db 1
    char db 'A'
```

下面是在每相隔 (50000*580) 时间后的字符 y 坐标的交替选择部分的代码：

```
Loop:
    dec word[num]
    jnz Loop
    mov word[num],delay
    dec word[dnum]
    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，为2第二条
    jz ch1
    jmp ch0
```

```

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
    jmp Swi
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

```

基于上述程序的改动，就可以实现两条轨迹的运动。

除此之外，由于改动后的程序生成的 bin 文件超过了一个扇区大小，因此在将此文件划分成两个扇区加载，首扇区用以显示个人信息，第二个扇区显示字符运动轨迹。

如下是具体代码：

```

Message:
    db '18340225', 0DH, 0AH
    db 'Zhong Jie'
    MessageLength equ ($-Message)
    times 510-($-$$) db 0    ;填充剩下的空间;使生成的二进制代码恰好为512字节
    dw 0xaa55                ;结束标志

```

\$为当前指令的地址，\$\$为程序起始地址，因此\$-\$\$即为当前指令之前的所有字节数。那么（510-（\$-\$\$））就是第一个扇区剩余的除了最后两个字节结束标志的字节数，用 times 语句填充 0.最后定义首扇区的结束标志即可。

因此，如上首扇区即用于显示个人信息。而要想显示字符的运动，程序需要加载到第二个扇区执行，因此 Load 段为读第二个扇区并跳转至相应内存地址执行的操作。

```

Load:
    ;读取扇区内容至ES:BX处
    mov ax, cs                ;段地址
    mov es, ax                ;ES=段地址
    mov bx, 7E00H             ;BX=偏移地址，第一个扇区偏移地址为7C00H
    mov ah, 02H               ;功能号02H;读扇区
    mov al, 01H               ;AL=扇区数
    mov ch, 00H               ;CH=柱面号;起始编号为0
    mov cl, 02H               ;CL=起始扇区号;起始编号为1
    mov dh, 00H               ;DH=磁头号;起始编号为0
    mov dl, 00H               ;DL=驱动器号;这里是软盘
    int 13H                   ;调用读磁盘BIOS的13h功能
    ;引导扇区程序已加载到指定内存区域中
    jmp 7E00H                 ;跳转至相应地址执行

```

根据上述描述即可合理分配扇区。

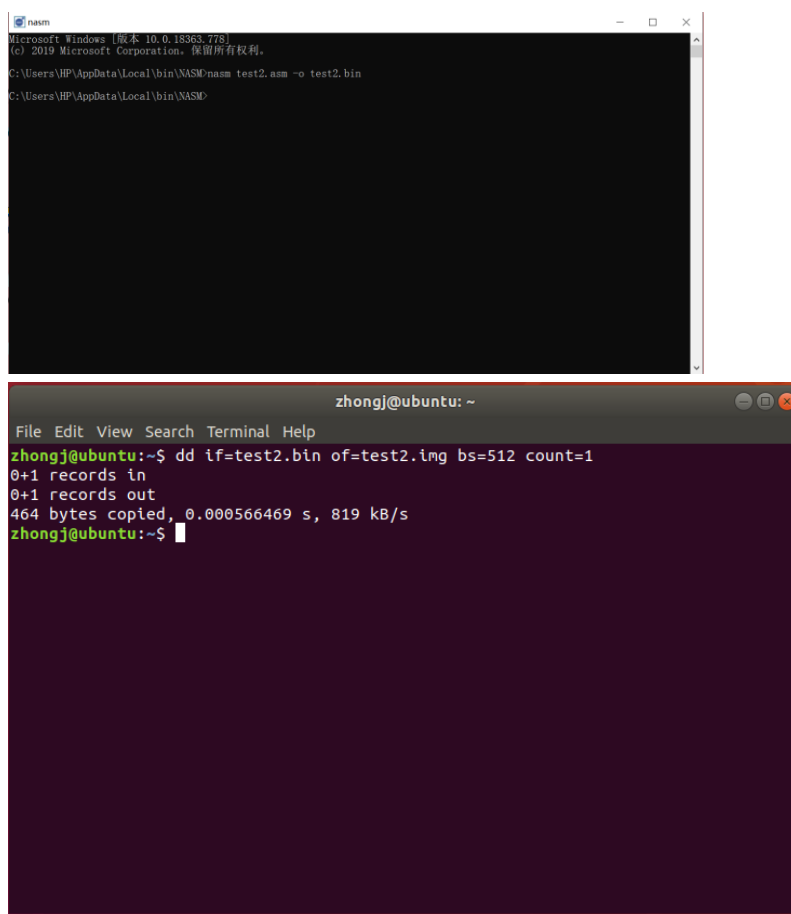
五、实验结果

注：test2.asm 为单条运动轨迹代码；test3.asm 为两条运动轨迹代码

● 单条运动轨迹

将上述包含上述代码的“test2.asm”文件利用 NASM 汇编生成“test2.bin”二进制文件，并在虚拟机中利用 dd 命令生成“test2.img”软盘映像文件。

操作过程如下所示：



```
nasm
Microsoft Windows [版本 10.0.18363.778]
(c) 2019 Microsoft Corporation. 保留所有权利。

C:\Users\HP\AppData\Local\bin\NASM>nasm test2.asm -o test2.bin
C:\Users\HP\AppData\Local\bin\NASM>
```

```
zhongj@ubuntu: ~
File Edit View Search Terminal Help
zhongj@ubuntu:~$ dd if=test2.bin of=test2.img bs=512 count=1
0+1 records in
0+1 records out
464 bytes copied, 0.000566469 s, 819 kB/s
zhongj@ubuntu:~$
```

这条 dd 命令能够创建一个大小为 512 字节，名字为 test2.img，写入了 test2.bin 的虚拟软盘映像，其中 test2.bin 为读取位置，test2.img 为写入位置，bs=512 是每次读写大小，count=1 指的是只读取一次。

基于上述操作就可以得到一个扇区（512B）大小的虚拟软盘映像，根据实验要求需将该程序机器码放入 1.44MB 的虚拟软盘中，即创建一个包含此引导扇区的 1.44MB 虚拟软盘映像。如下：

```
zhongj@ubuntu: ~  
File Edit View Search Terminal Help  
zhongj@ubuntu:~$ dd if=test2.bin of=test2.img bs=512 count=1  
0+1 records in  
0+1 records out  
464 bytes copied, 0.000566469 s, 819 kB/s  
zhongj@ubuntu:~$ dd if=/dev/zero of=empty.img bs=512 count=2880  
2880+0 records in  
2880+0 records out  
1474560 bytes (1.5 MB, 1.4 MiB) copied, 0.0070038 s, 211 MB/s  
zhongj@ubuntu:~$ dd if=empty.img of=test2.img skip=1 seek=1 bs=512 count=2879  
2879+0 records in  
2879+0 records out  
1474048 bytes (1.5 MB, 1.4 MiB) copied, 0.00823106 s, 179 MB/s  
zhongj@ubuntu:~$
```

首先我们先创建一个空的大小为 1.44MB 的虚拟软盘映像 empty.img。

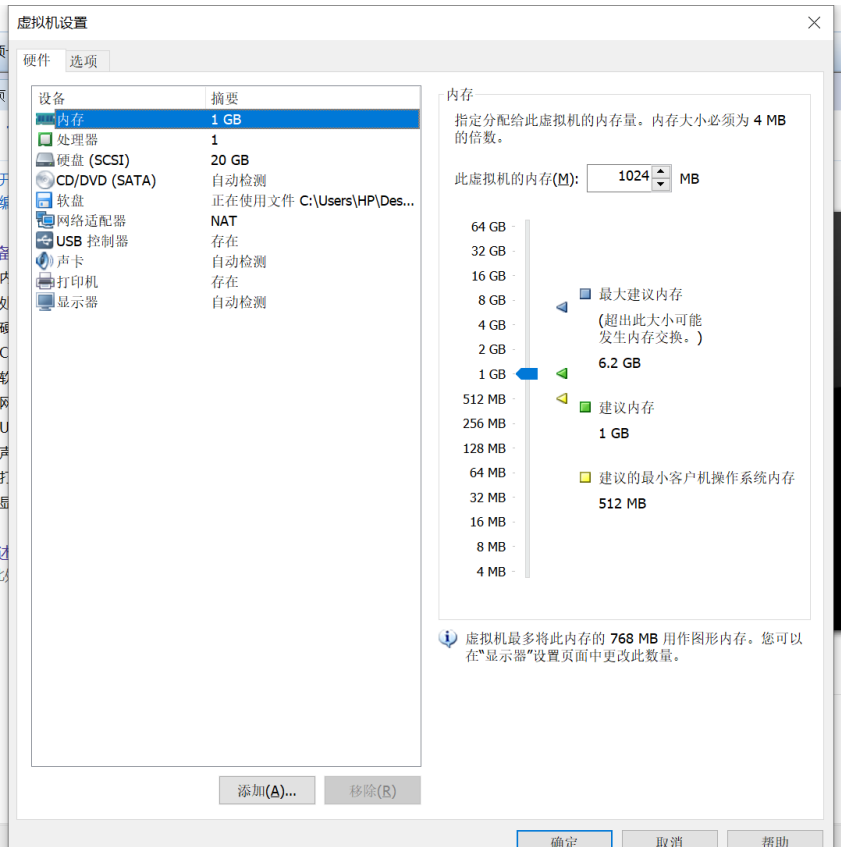
然后将 empty.img 第一个扇区之后的数据拷贝到我们之前制作的包含主引导记录的镜像文件 test2.img 后。那么 test2.img 即为大小为 1.44MB 的包含主引导记录的虚拟软盘镜像文件。

利用 WinHex 工具查看 test2.img(即为 src 文件夹下的 single_track.img)

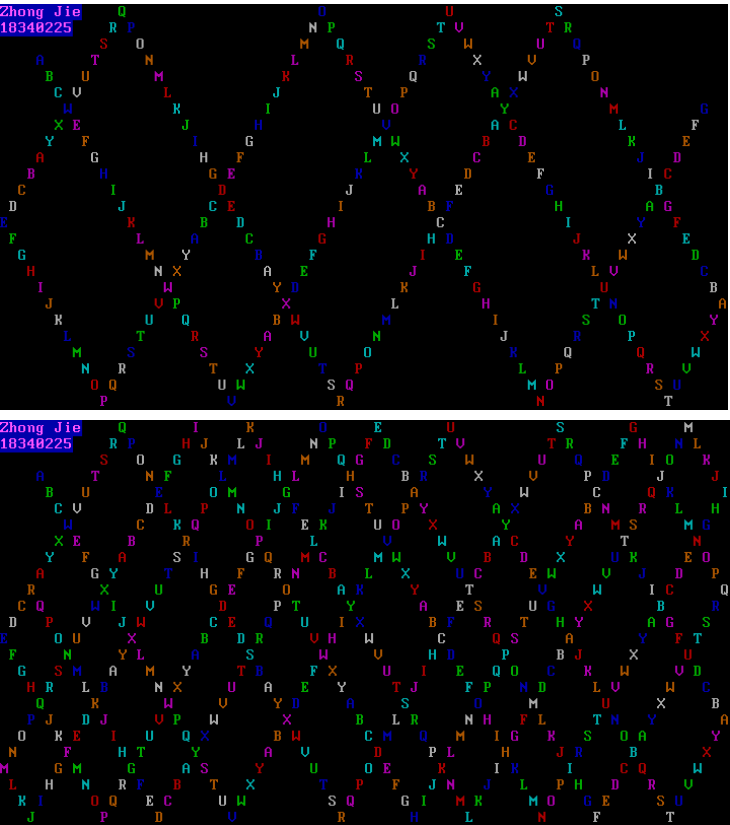
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000030	10	FF	0E	B2	7D	75	FA	C7	06	B2	7D	50	C3	FF	0E	B4	ÿ ²}uúÇ ²}PÃÿ ´
00000040	7D	75	EE	C7	06	B2	7D	50	C3	C7	06	B4	7D	44	02	B0	}uiÇ ²}PÃÇ ´}D °
00000050	01	3A	06	B6	7D	74	1C	B0	02	3A	06	B6	7D	74	51	B0	: ¶}t ° : ¶}tQ°
00000060	03	3A	06	B6	7D	0F	84	BC	00	B0	04	3A	06	B6	7D	74	: ¶} „¼ ° : ¶}t
00000070	7A	EB	FE	FF	06	B7	7D	FF	06	B9	7D	8B	1E	B7	7D	B8	zëþÿ ·}ÿ ¹}< ·},
00000080	19	00	29	D8	74	0E	8B	1E	B9	7D	B8	50	00	29	D8	74)Øt < ¹},P)Øt
00000090	11	E9	CB	00	C7	06	B7	7D	17	00	C6	06	B6	7D	02	E9	éÈ Ç ·} Æ ¶} é
000000A0	BD	00	C7	06	B9	7D	4E	00	C6	06	B6	7D	04	E9	AF	00	¼ Ç ¹}N Æ ¶} é
000000B0	FF	0E	B7	7D	FF	06	B9	7D	A1	B7	7D	BB	FF	FF	29	D8	ÿ ·}ÿ ¹}i ·}»ÿÿ)Ø
000000C0	74	0E	8B	1E	B9	7D	B8	50	00	29	D8	74	11	E9	8F	00	t < ¹},P)Øt é
000000D0	C7	06	B7	7D	01	00	C6	06	B6	7D	01	E9	81	00	C7	06	Ç ·} Æ ¶} é Ç
000000E0	B9	7D	4E	00	C6	06	B6	7D	03	EB	74	FF	06	B7	7D	FF	¹}N Æ ¶} étÿ ·}ÿ
000000F0	0E	B9	7D	8B	1E	B7	7D	B8	19	00	29	D8	74	0D	8B	1E	¹}< ·},)Øt <
00000100	B9	7D	B8	FF	FF	29	D8	74	0F	EB	54	C7	06	B7	7D	17	¹},ÿÿ)Øt étÇ ·}
00000110	00	C6	06	B6	7D	03	EB	47	C7	06	B9	7D	01	00	C6	06	Æ ¶} éÇÇ ¹} Æ
00000120	B6	7D	01	EB	3A	FF	0E	B7	7D	FF	0E	B9	7D	8B	1E	B7	¶} é:ÿ ·}ÿ ¹}< ·
00000130	7D	B8	FF	FF	29	D8	74	0D	8B	1E	B9	7D	B8	FF	FF	29	},ÿÿ)Øt < ¹},ÿÿ)
00000140	D8	74	0F	EB	1A	C7	06	B7	7D	01	00	C6	06	B6	7D	04	Øt é Ç ·} Æ ¶}
00000150	EB	0D	C7	06	B9	7D	01	00	C6	06	B6	7D	02	EB	00	A1	é Ç ¹} Æ ¶} é i
00000160	B7	7D	BB	50	00	F7	E3	03	06	B9	7D	BB	02	00	F7	E3	·}»P ÷ã ¹}» ÷ã
00000170	89	C5	8A	26	BB	7D	A0	BC	7D	65	89	46	00	80	06	BB	%ÃŠ&} ¼}e%F € »
00000180	7D	01	80	06	BC	7D	01	B3	08	38	1E	BB	7D	74	0B	B3	} € ¼} ³ 8 »}t ³
00000190	5A	38	1E	BC	7D	74	13	E9	97	FE	C6	06	BB	7D	01	B3	Z8 ¼}t é-pÆ »} ³
000001A0	5A	38	1E	BC	7D	74	03	E9	87	FE	C6	06	BC	7D	41	E9	Z8 ¼}t éþpÆ ¼}Aé
000001B0	7F	FE	50	C3	44	02	01	02	00	03	00	01	41	5A	68	6F	pPÃD Azho
000001C0	6E	67	20	4A	69	65	31	38	33	34	30	32	32	35	EB	FE	ng Jie18340225ëp
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA	Uª

首扇区 512 个字节包含引导扇区程序，且以 0xAA55 结尾符合要求。

接着我们创建虚拟机，完成虚拟机的基本配置：



配置完成后，开启此虚拟机，虚拟机上程序运行结果如下：

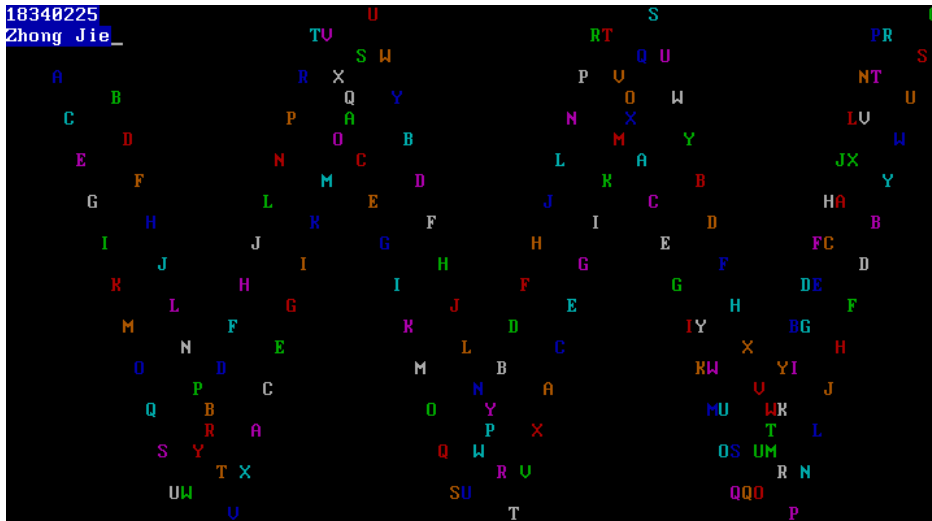


显而易见，程序运行结果与实验要求相符合。

● 两条运动轨迹

跟单条运动轨迹汇编程序文件“test2.asm”类似，两条运动轨迹程序“test3.asm”也利用 NASM 编译器生成 bin 文件，在利用 dd 命令生成 1.44MBIMG 虚拟软盘镜像文件即可。

虚拟机上程序运行结果如下：



程序运行结果与预期相符合。

六、实验总结：

本次实验是《操作系统原理实验》课程的第一个实验，对于刚刚接触这门课程的我来说，实验初期我感到十分迷茫与艰难。本次实验涉及虚拟软盘的创建、汇编语言的程序设计、nasm 汇编，虚拟软盘镜像文件的生成等问题，这对于刚刚入门的我来说，无疑是大片的空白与盲区，对于有些实验工具我闻所未闻，对于虚拟软盘、引导扇区等一些概念一无所知，而虽然上学期在《计算机组成原理实验》这门课程中接触过汇编语言，但 MIPS 汇编与 X86 汇编却大有不同，X86 的小端存储方式也让我感觉很不适应。

当然，万事开头难，我抱着“硬着头皮往前闯”的想法，先开始学习并了解这方面的相关知识。从一些网上资源中了解到在 PC 机开机时，操作系统是如何启动的，而基于此，我学习了引导扇区的相关知识，了解到在加电自检、复位操作之后，CPU 会取得一条位于 ROM 中的 jump 指令，执行完成之后，ROM-BIOS 会从硬盘中读取首扇区的内容将其加载到内存中 0x07c00 处，并用 jump 跳转至那里执行，进一步进行加载操作系统内核并初始化一些参数等操作，完成启动。

了解了这些知识，我对于实验题目的要求就有了更加清晰的理解，而在凌老师和一些同学的帮助下，我也认识了一些实验需要用到的工具，在不断的尝试与摸索过程后，我了解了这些实验工具的用途以及最基本的用法，因此在边查资料边自己摸索的状态下，我成功地创建了虚拟机、生成了多个 1.44MB 的虚拟软盘并且利用 WinHex 工具将其中一个虚拟软盘的首扇区填满了个人信息。

上述的学习过程、实验工具的准备以及实验环境的搭建，让我在本次实验的征程成功迈出了第一步，而接下来就是实验的关键部分——汇编语言程序设计。

X86 的汇编语言中有一些指令的含义与 MIPS 相似，但 X86 的一个字大小是 16 位 2 个字节的，因此寄存器的大小也是 16 位的，这是与 MIPS 的不同。除此之外，基于 NASM 这款汇编语言编译程序，NASM 汇编的数据定义与声明，数据常量的定义以及 I/O 显示的机制是与 MIPS 大大不同的。在网上查阅资料以及阅读相关书籍之后，我基本掌握了 DB、DW、DQ，EQU 等一些数据声明定义相关的指令，同时也对 BIOS 调用的功能号、中断号以及入口参数与各种特定寄存器的关系有了大致的认识。这对于我编写汇编语言程序可以说是大有帮助。在成功编写好程序之后，要将程序编译成二进制文件，再将二进制文件生成镜像文件，这过程中所用的工具、所需的命令也让我大开眼界。

总而言之，本次实验对于我而言是一次作业驱动的自主学习。这促使我通过查阅资料、阅读书籍了解了一些有关操作系统的知识，同时也锻炼、强化了个人的汇编程序设计能力。虽然，整个实验过程中，我感觉很艰难、难扛，但“良好的开端是成功的一半”，把入门实验做好，其实是充实我的知识储备的一种极佳的方式。

当然，即使做完本次实验，我对于操作系统、汇编语言的了解也只是零星半点。“路漫漫其修远兮，吾将上下而求索”，我还需要进一步的阅读、学习，来不断解决自己的一个又一个疑惑。希望之后，自己可以更深入的掌握 X86 汇编，更透彻理解操作系统相关原理，可以对本实验做出一些改进和更多的创新，也可以在之后的实验过程中更加得心应手。

七、参考文献：

[1] 《x86 汇编语言-从实模式到保护模式》

[2] [Intel 汇编-NASM]

https://blog.csdn.net/Lirx_Tech/article/details/42340619