实验一

实验目的

1. 了解原型操作系统设计实验教学方法与要求
2. 了解计算机硬件系统开机引导方法与过程
3. 掌握操作系统的引导程序设计方法与开发工具
4. 学习PC字符显示方法、复习加强汇编语言程序设计能力

实验要求

1. 知道原型操作系统设计实验的两条线路和前6个实验项目的差别
2. 掌握PC电脑利用1.44MB软驱的开机引导方法与过程的步骤
3. 在自己的电脑上安装配置引导程序设计的开发工具与环境
4. 参考样版汇编程序，完成在PC虚拟机上设计一个1.44MB软驱的引导程序的完整工作。
5. 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，按时打包提交实验相关文档。

实验方案

实验环境

硬件：个人计算机

操作系统：Windows 10

虚拟机软件：VirtualBox

实验开发工具

语言工具：16位x86汇编语言

汇编器：nasm

磁盘映像文件浏览编辑工具：WinHex

代码编辑器：Visual Studio Code

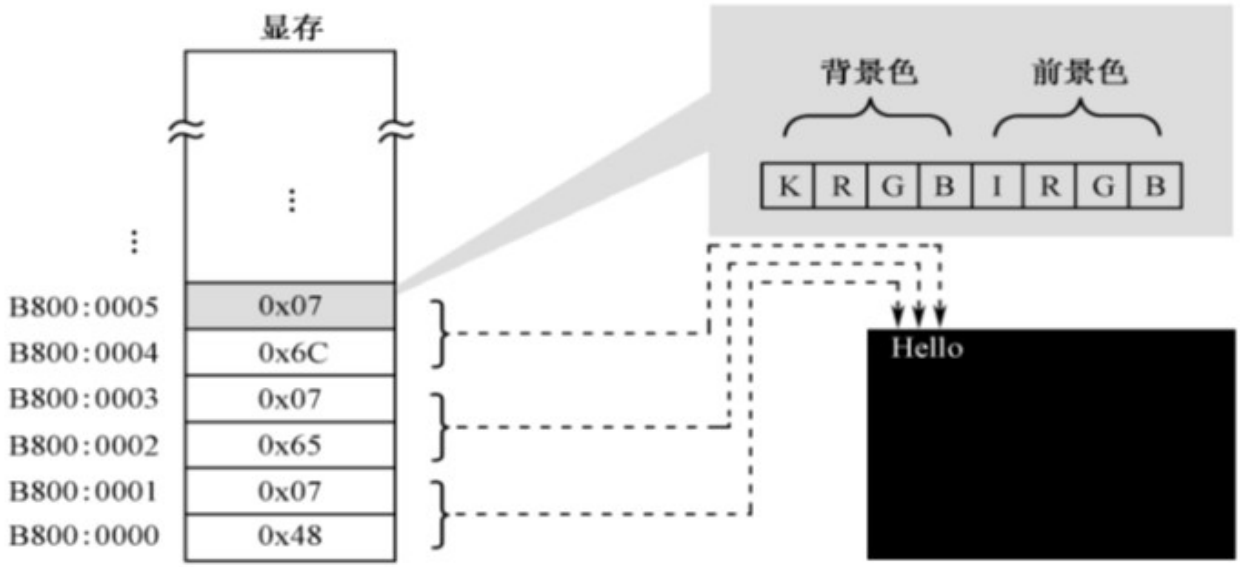
相关原理

在屏幕上显示文字

显示器将显存里的内容呈现在屏幕上。而显卡提供了一个文本模式，其最小的可控制单位为字符，VGA：25*80。这意味着，我们控制文字显示，实际上可以看成是编辑一个二维数组（大小为25*80）。

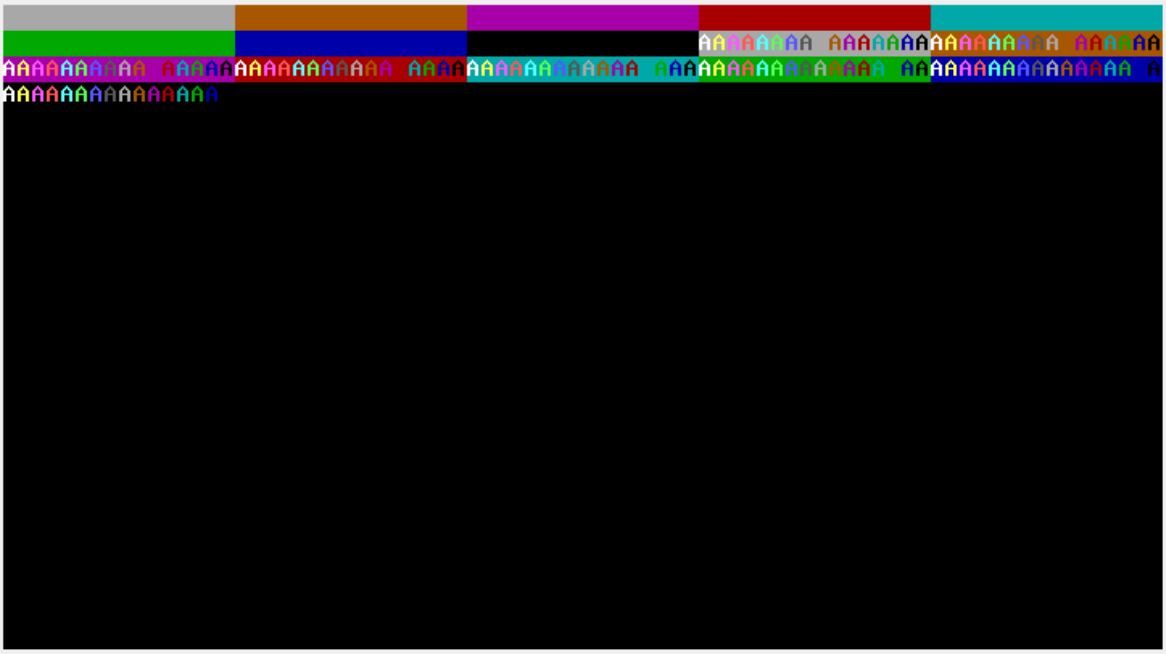
同时，由于显存在内存地址空间里占据了B8000-BFFFF这段区域（共32KB），所以，我们可以编辑这端内存区域，从而控制屏幕字符输出。

屏幕上字符的显示属性



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

由上图，显存里输出单个ascii字符需要两个字节。低8位是ascii码，高八位则是上图里显示的颜色控制属性。为了更清晰的看清上表的颜色，我写了一个小程序进行观察（没有A的地方实际上是在闪烁，静态图片看不出来）：



程序流程

初始化程序

首先是利用伪指令初始化一些常量。

```
      Dn_Rt equ 1                   ;D-Down,U-Up,R-right,L-Left
      Up_Rt equ 2                   ;
      Up_Lt equ 3                   ;
      Dn_Lt equ 4                   ;
      Still equ 5
      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
;      .386
      org 7c00h                   ; 程序加载到100h，可用于生成COM/7c00H引导扇区程序

start:
    ;xor ax,ax                    ; AX = 0    程序加载到0000：100h才能正确执行
    mov ax,cs
    mov es,ax                     ; ES = 0
    mov ds,ax                     ; DS = CS
    mov es,ax                     ; ES = CS
    mov ax,0B800h                 ; 文本窗口显存起始地址
    mov gs,ax                     ; GS = B800h
    mov byte[char],'A'            ; 设置程序初始化字符
```

设置窗口背景颜色

原理很简单，就是给整个大小为25* 80的显存，填充背景颜色为青色（按上面原理给出的图作为参考），字符为空格（即空白字符）。这样就可以形成一个空白的青色背景。

```
SetBackGround:
    mov cx, 0x7d0   ; 等于25 * 80 即显存的大小
    mov bx, 0x0     ; 用于显存偏移量
    mov ax, 0x3020  ; 颜色为青色的空格字符
ls:
    mov [gs:bx],ax  ; 给gs:bx地址填充颜色
    inc bx
    inc bx
    loop ls
```

显示自己的名字和学号

```
DisplayName:
    mov ah, 0x3f            ; 设置背景色
    mov bx, 0x0
    mov si, 0x0             ; 数组偏移量
ld:
    mov al, byte[id + si]   ; 名字起始地址是id，si寄存器相当于数组的索引
    cmp al, 0x0             ; 判断有没有到字符数组的结尾
    jz exit
    mov [gs:bx],ax          ; 显示字符的ASCII码值
    inc si                  ; 更新字符数组偏移量
    inc bx
    inc bx
    jmp ld
data:
    id db "18340133 Ouyang Haolan", 0x0
```

字符移动的决策

这个功能相当于是一个switch-case语句，利用一个rdul变量，控制字符的下一个位置应该相对于上个位置怎么移动。

```
    mov al,1
    cmp al,byte[rdul]
    jz  DnRt                ; rdul = 1 向右下运动
      mov al,2
      cmp al,byte[rdul]
    jz  UpRt                ; rdul = 2 向右上运动
      mov al,3
      cmp al,byte[rdul]
    jz  UpLt                ; rdul = 3 向左上运动
      mov al,4
      cmp al,byte[rdul]
    jz  DnLt                ; rdul = 4 向左下运动
      jmp $                 ; rdul = 其它，程序挂起
```

具体的移动实现代码

由于左上左下右上右下的代码逻辑都是一样的，我这里只截取一段进行代码逻辑分析。

```
; 右下程序
; 变量说明：x是字符在显示器的横坐标，y是纵坐标,以左上角为(0,0)坐标往下和右延申
DnRt:
    ; 右下就是x+1, y+1
    inc word[x]
    inc word[y]
    ; 判断是否超出下边界
    mov bx,word[x]
    mov ax,25
    sub ax,bx
      jz  dr2ur
    ; 判断是否超出右边界
    mov bx,word[y]
    mov ax,80
    sub ax,bx
      jz  dr2dl
    jmp show
; 反弹程序
dr2ur:
      mov word[x],23
      mov byte[rdul],Up_Rt  ;右下->右上
      jmp show
dr2dl:
      mov word[y],78
      mov byte[rdul],Dn_Lt  ; 右下->左下
      jmp show
```

显示字符

运动中，字符和字符的颜色都会动态的改变。fcolor和acount是记录当前字符和字符颜色号的变量。

```
show:
    ; 计算显存地址
      xor ax,ax
      mov ax,word[x]
    mov bx,80
    mul bx
    add ax,word[y]
    mov bx,2
    mul bx
    mov bx,ax

    ; 设置字体颜色 背景色与之前设置的相同，前景色随移动进行变化
color:
    inc byte[fcolor]
    mov ah, [fcolor]
    cmp ah, 0x10
    jnz continue
```

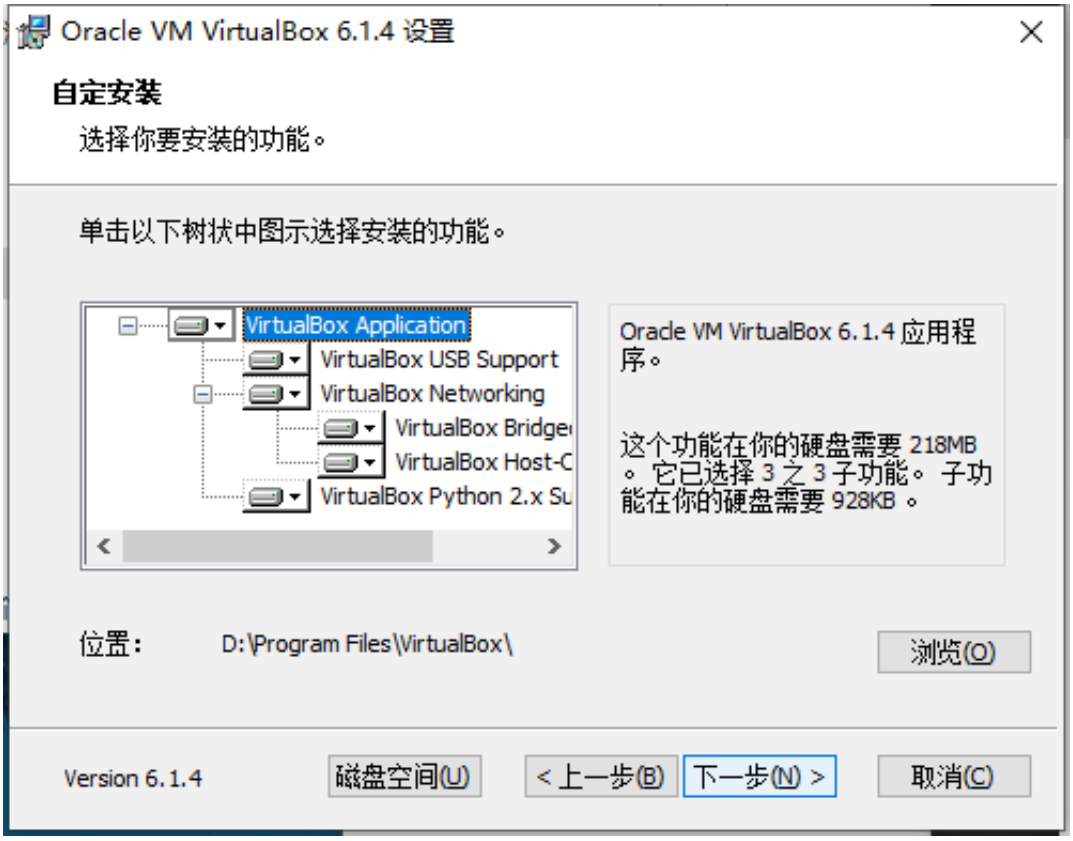
```
    mov ah, 0x0
continue:
    mov [fcolor], ah
    add ah, [bcolor]

    ; 设置显示的字符，随移动进行变化（按A-z的顺序）
character:
    inc byte[acount]
    mov cl, [acount]
    cmp cl, 0x1a
    jnz show1
    mov cl,0x0

    ; 完成设置后进行显示
show1:
    mov [acount],cl
    mov al,byte[char]            ;  AL = 显示字符值（默认值为20h=空格符）
    add al,cl
    mov [gs:bx],ax          ;  显示字符的ASCII码值
```

实验过程和结果

(1) 安装虚拟机VirtualBox

首先打开下载好的安装包

设置一下安装位置，安装默认的功能点击下一步

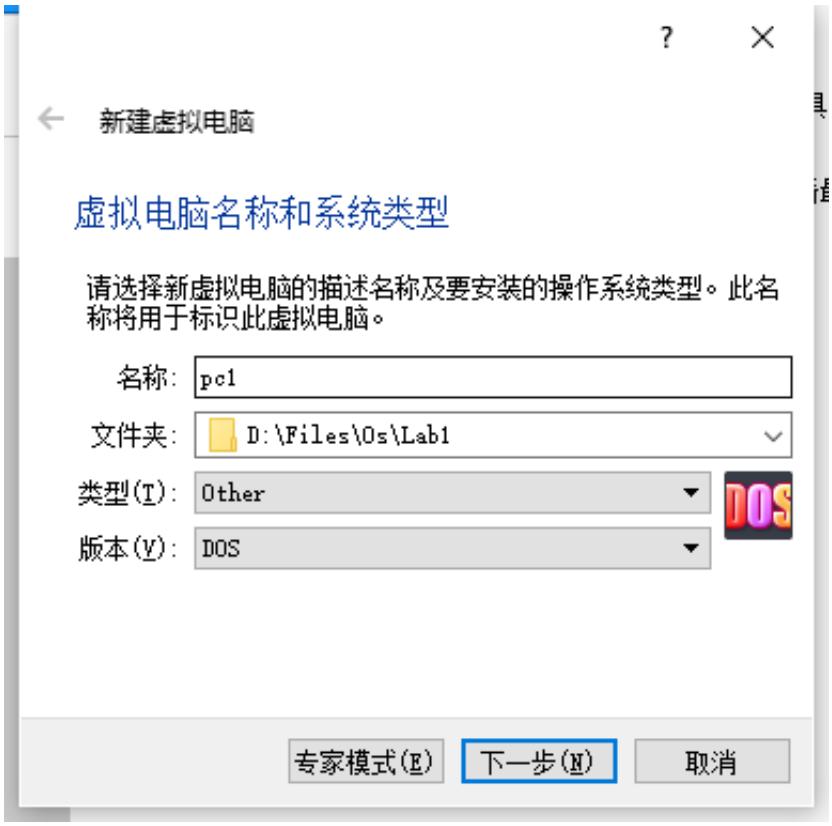


按照引导完成安装工作

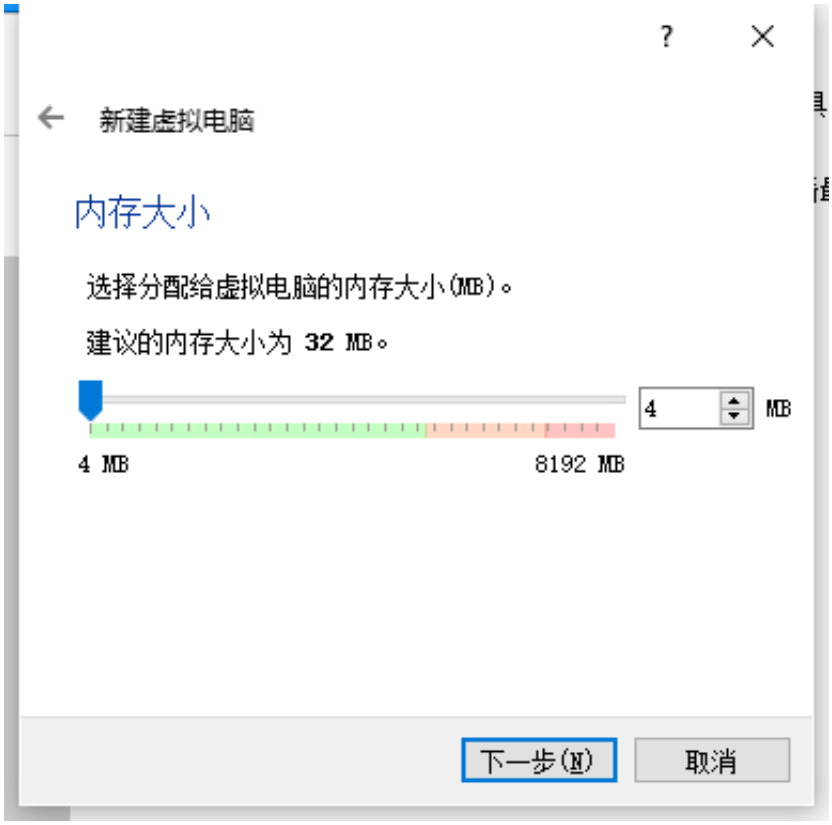
（2）生成一个PC虚拟机及三个空的软盘映像文件

按照向导，点击新建

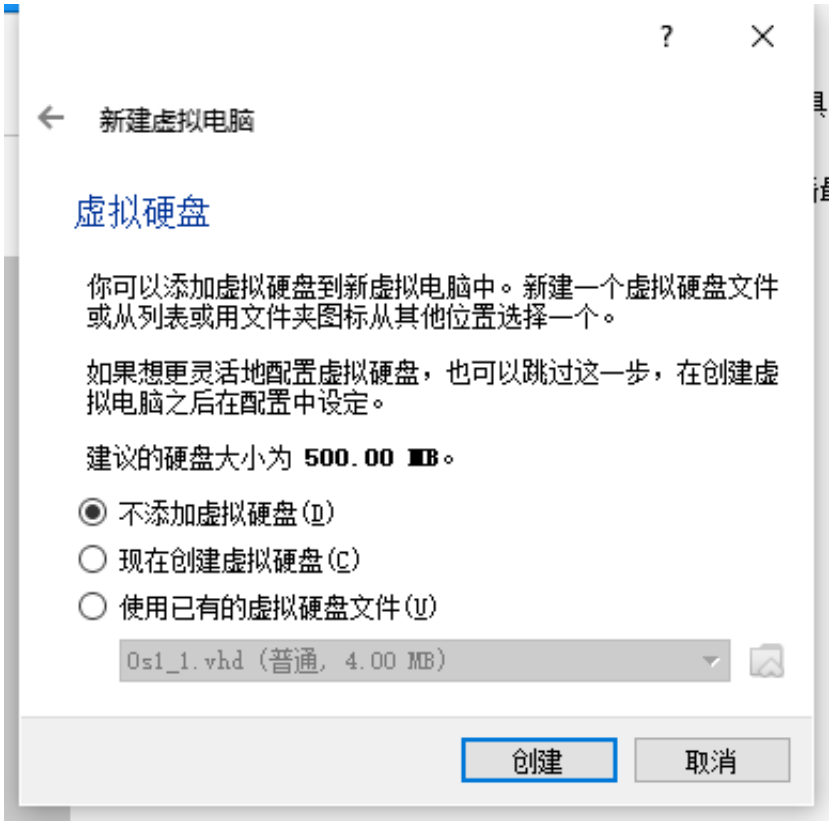
选择Other类型，DOS版本，命名这个操作系统为pc1。之后，点击下一步



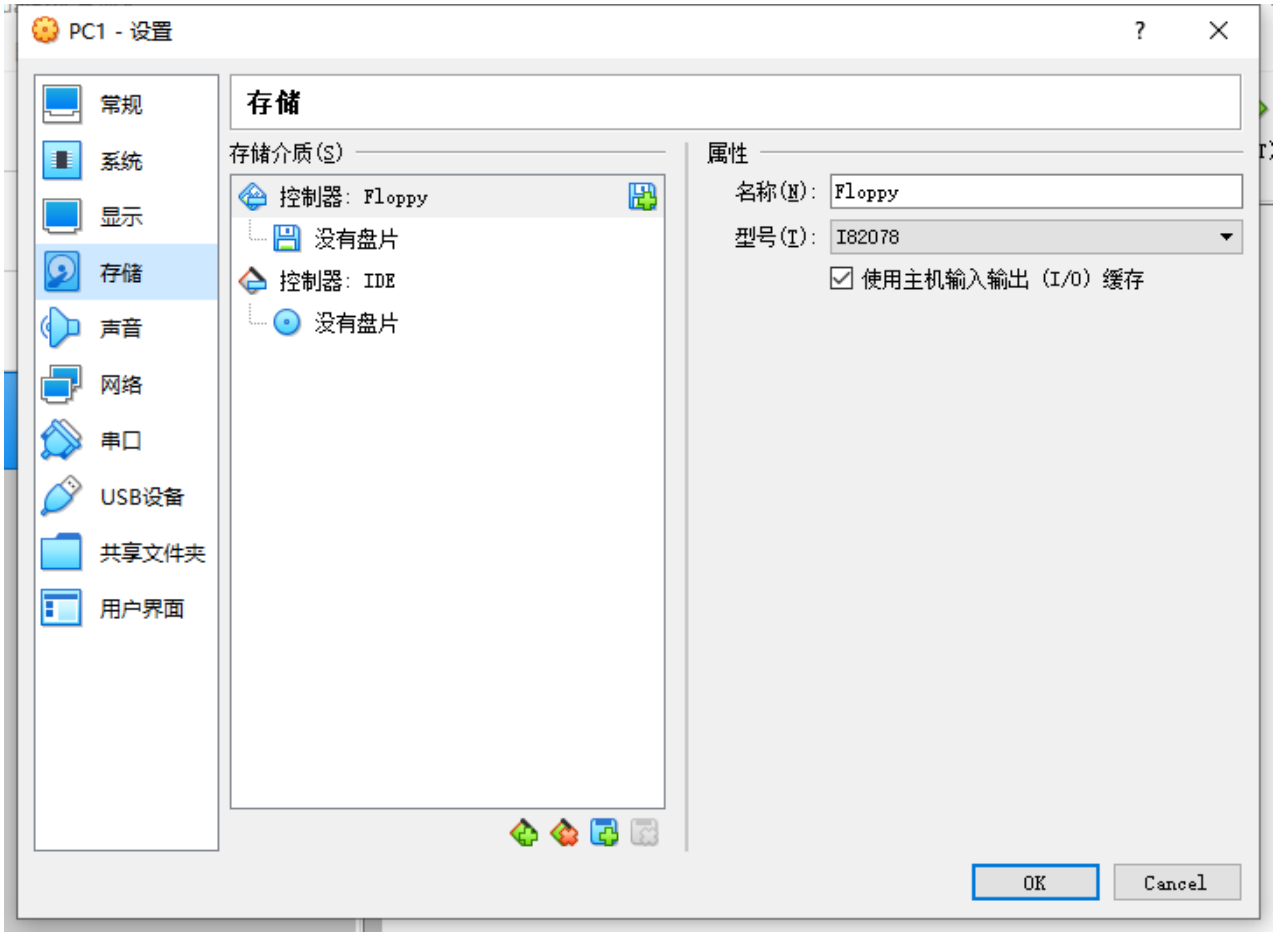
选择最小内存4MB（实验要求只需要1MB，但因为VirtualBOX默认最小是4MB）



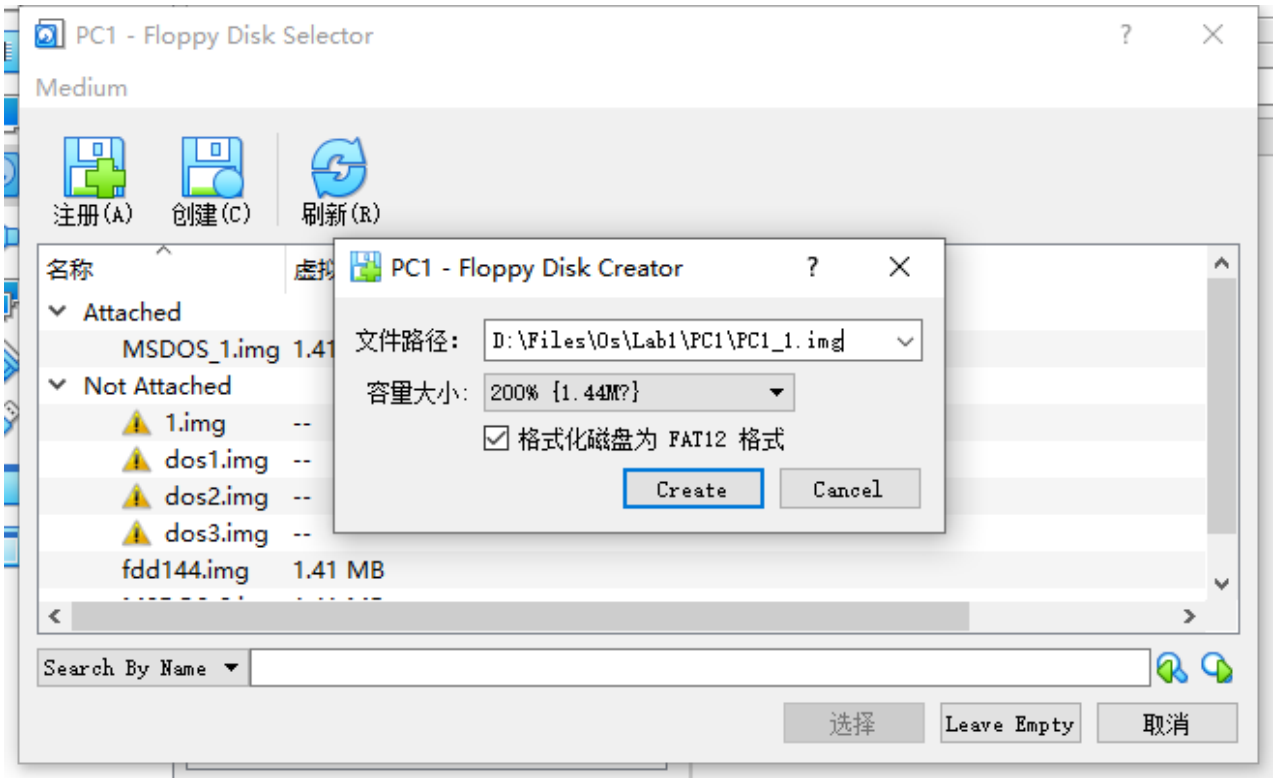
不创建虚拟硬盘，我会在进去后重新创建一个新的大小为1.44MB的软盘



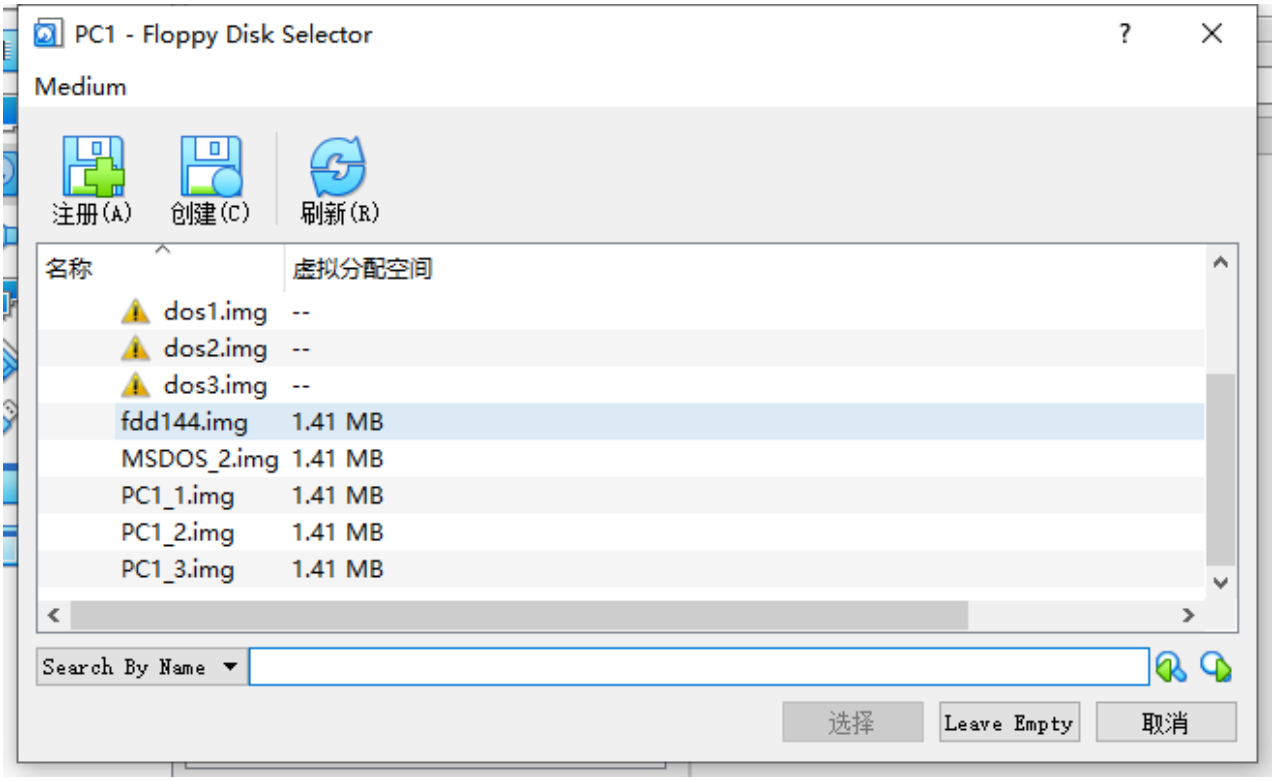
完成创建后，我打开这个虚拟机的设置，可以看到此时没有任何存储介质



点击右上角的创建符号弹出如下界面。

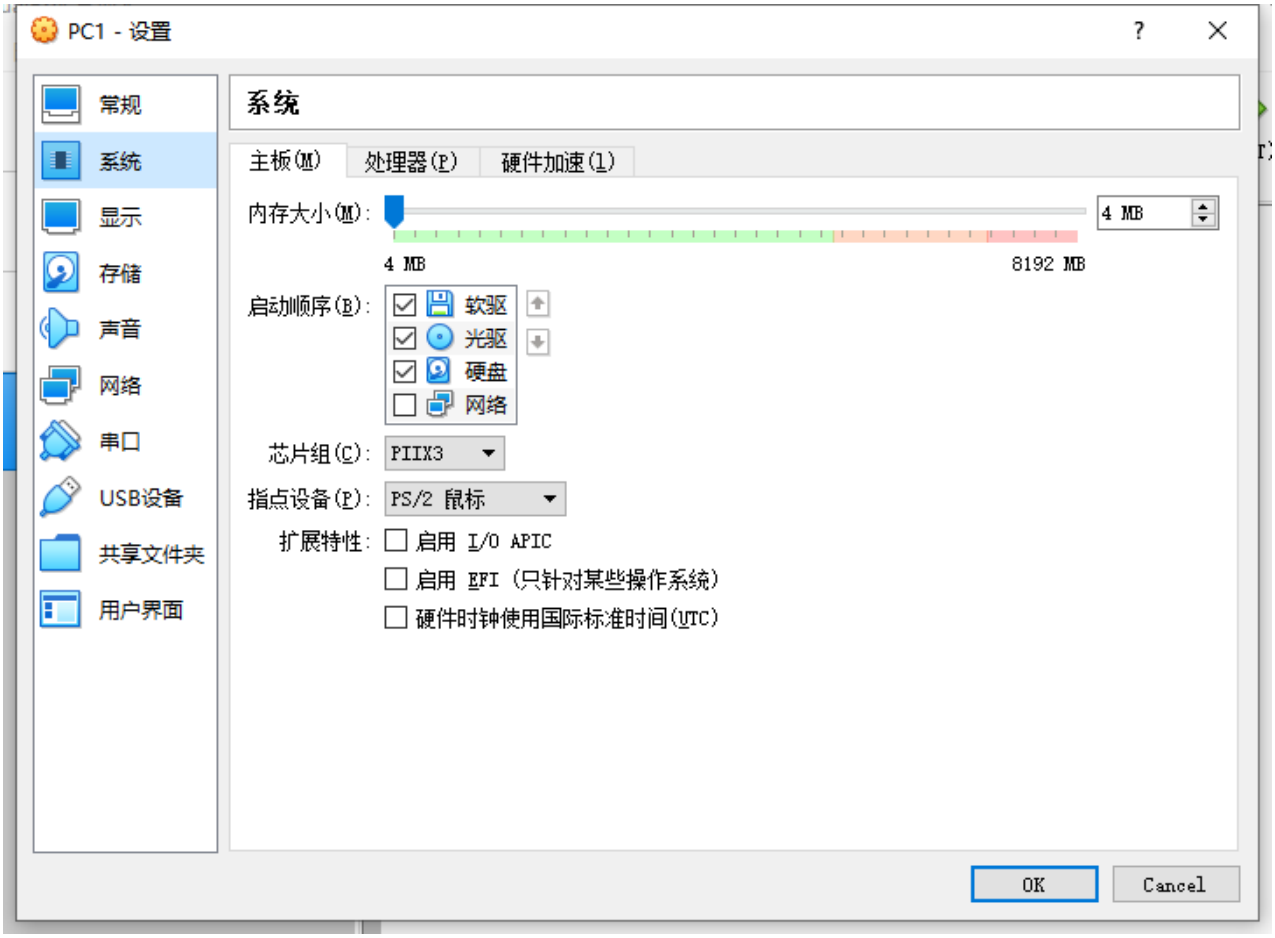


我按照这个方法创建了4个1.44MB大小的软盘映像文件：一个是PC1虚拟机的软盘映像文件，另外3个是实验要求的空的软盘映像文件。他们分别是PC1_1.img PC1_2.img PC1_3.img文件。

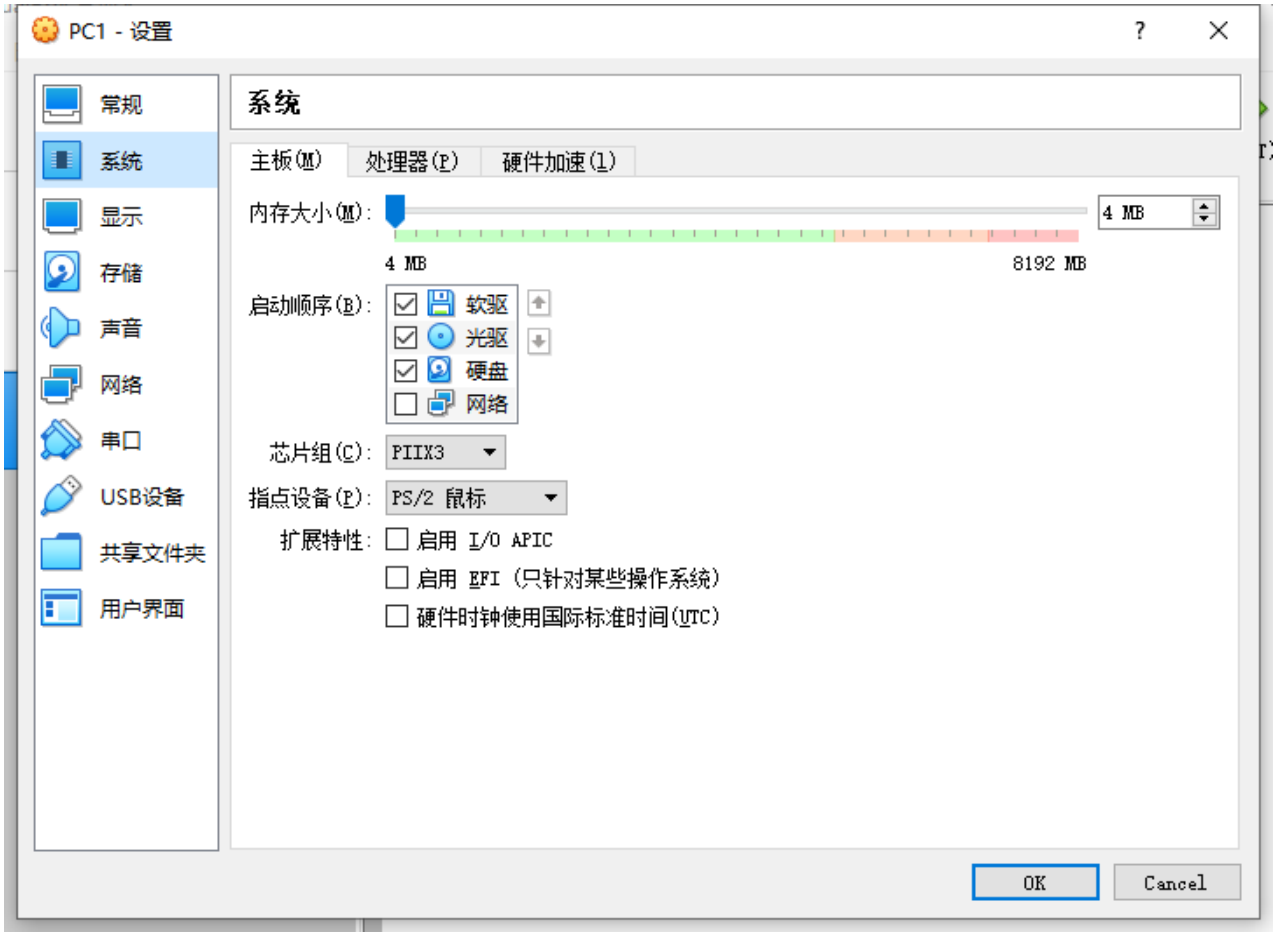


PC虚拟机配置

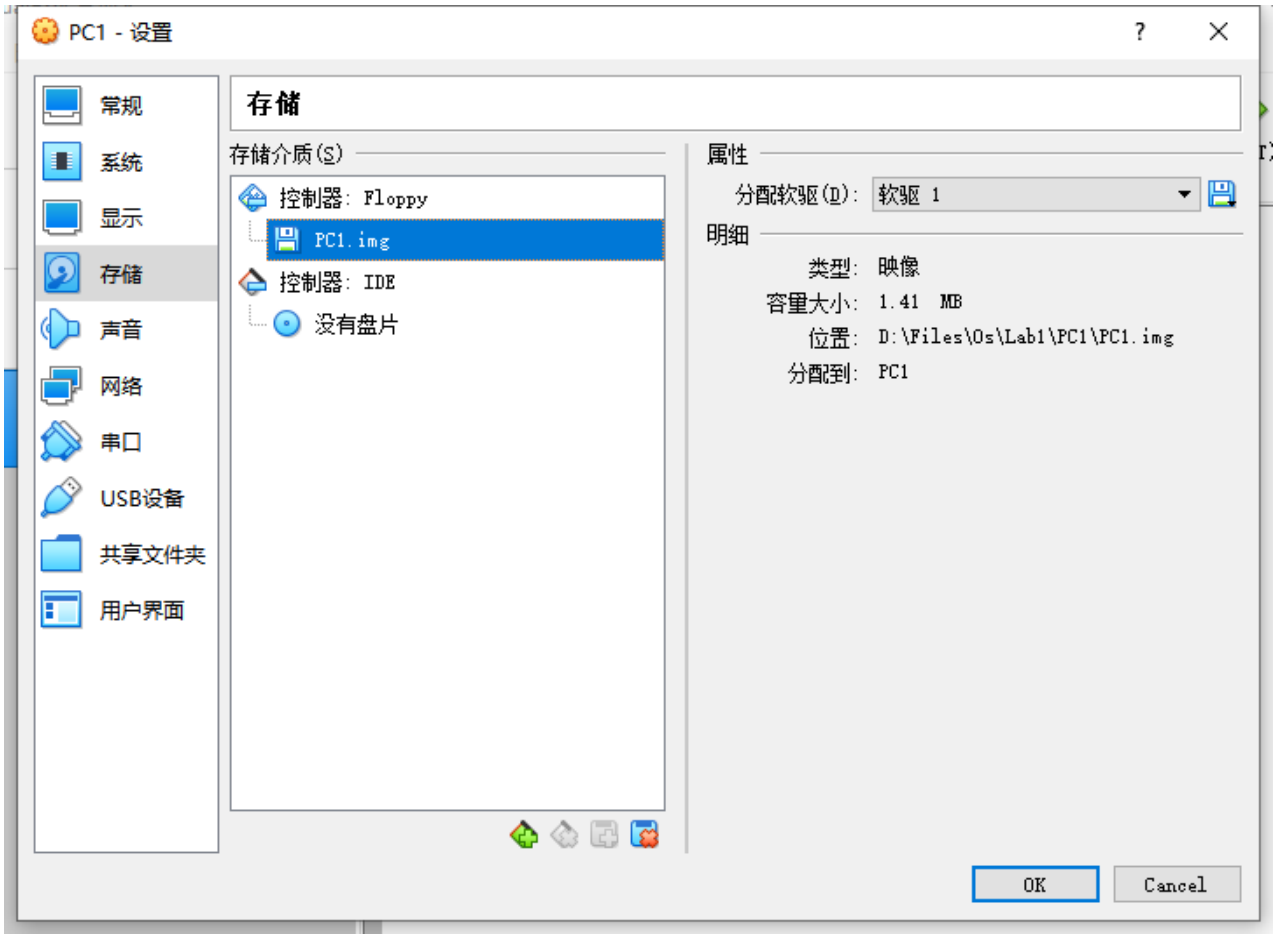
内存



显存

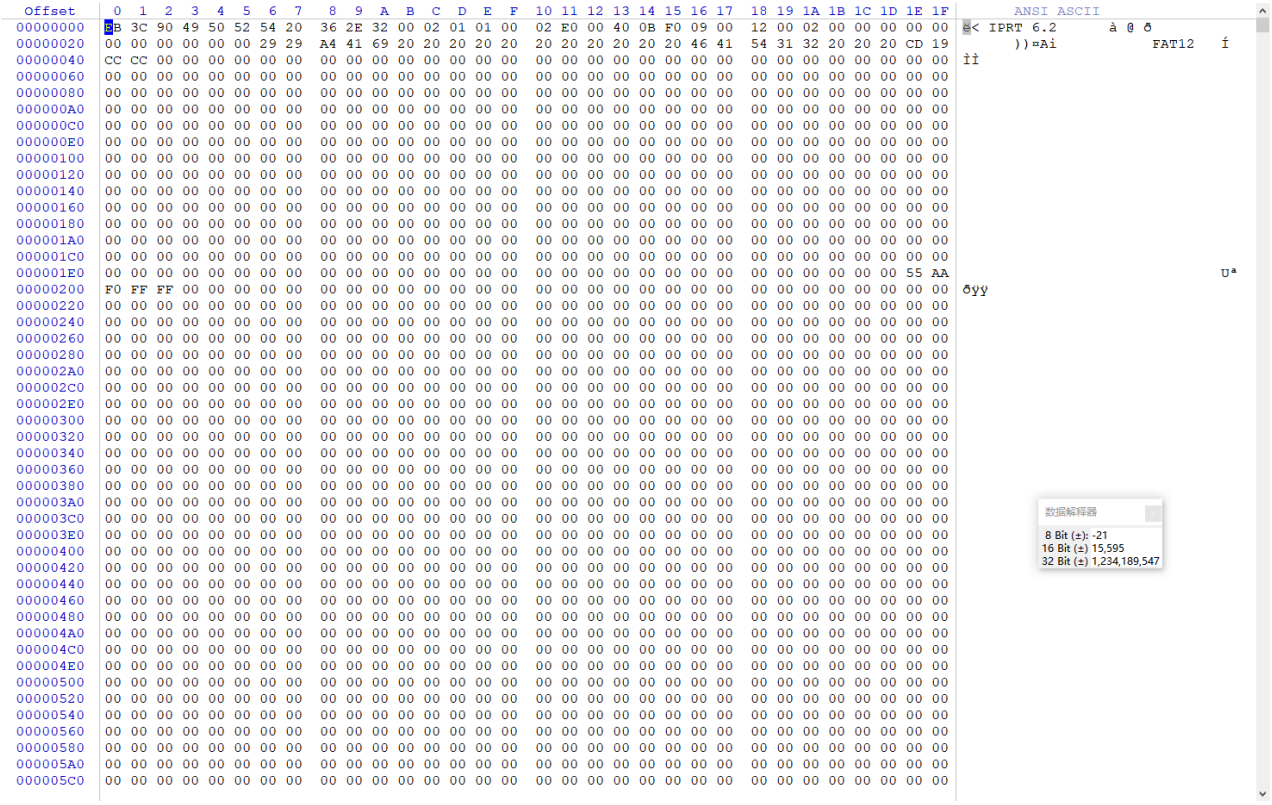

PC1 - 设置
常规
系统
显示
存储
声音
网络
串口
USB设备
共享文件夹
用户界面
系统
主板(M)
处理器(P)
硬件加速(L)
内存大小(M):
4 MB
4 MB
8192 MB
启动顺序(B):
软驱
光驱
硬盘
网络
芯片组(C):
PIIX3
指点设备(P):
PS/2 鼠标
扩展特性:
启用 I/O APIC
启用 EFI (只针对某些操作系统)
硬件时钟使用国际标准时间(UTC)
OK
Cancel

存储介质

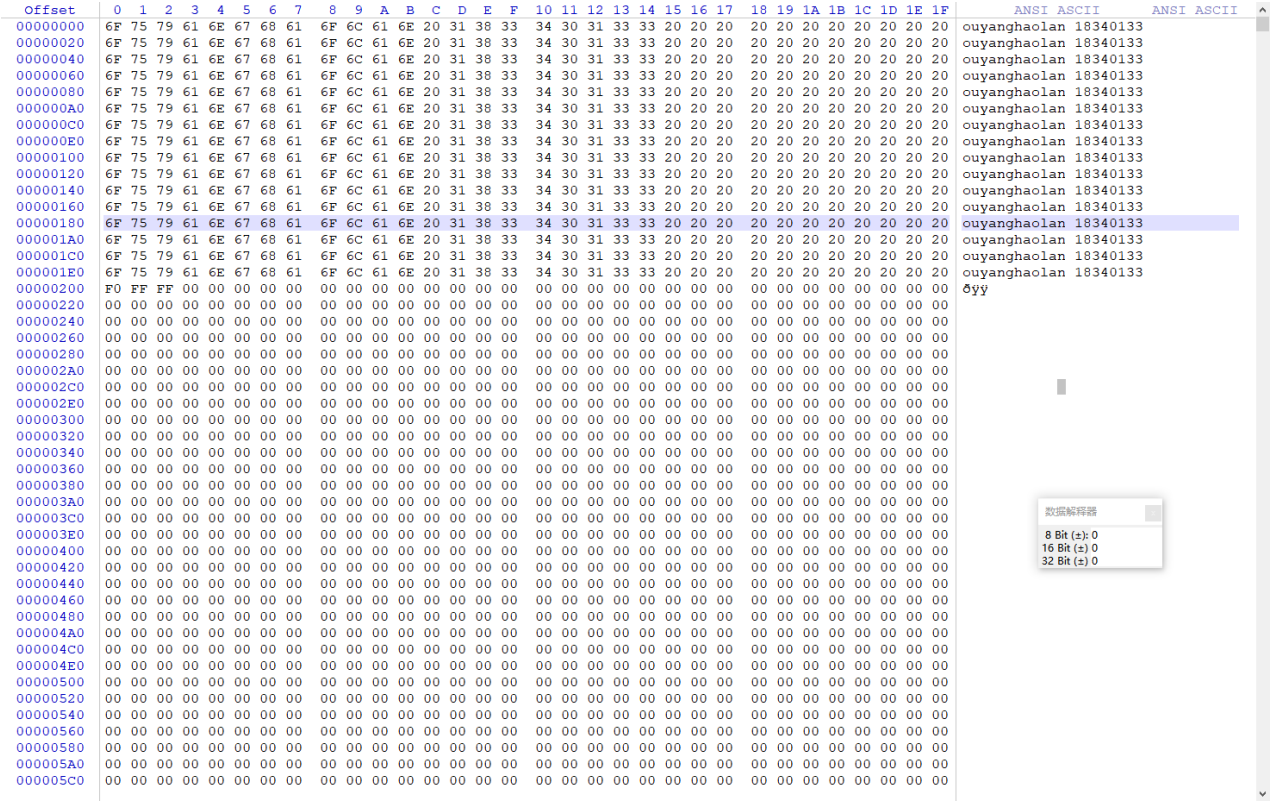

PC1 - 设置
常规
系统
显示
存储
声音
网络
串口
USB设备
共享文件夹
用户界面
存储
存储介质(S)
控制器: Floppy
PC1.img
控制器: IDE
没有盘片
属性
分配软驱(D):
软驱 1
明细
类型: 映像
容量大小: 1.41 MB
位置: D:\Files\Os\Lab1\PC1\PC1.img
分配到: PC1
OK
Cancel

（3）填充第一个软盘映像文件

利用winHex软件打开刚刚生成的空的软盘映像文件，显然第一个扇区基本是空的



经过编辑（即填充姓名和学号以后），利用右边的磁盘内容显示区域可以看到，编辑成功。



（4）安装x86汇编编辑工具

我安装的是Microsoft公司的Visual Studio Code

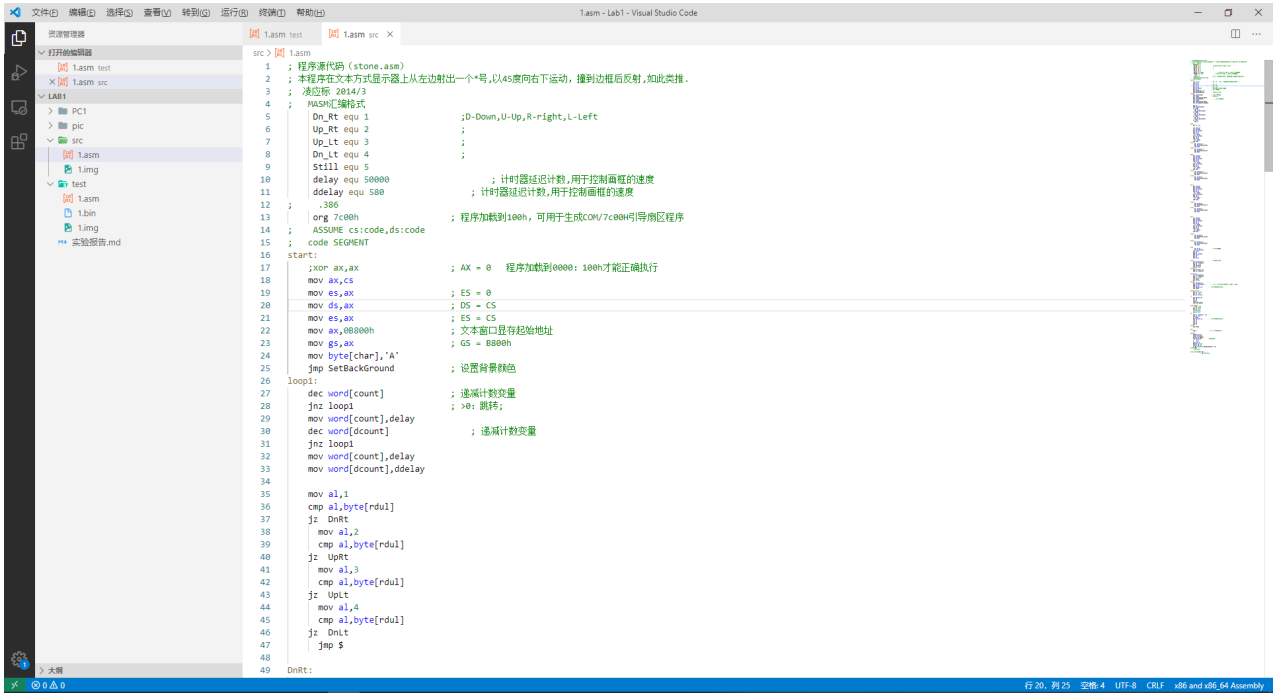
汇编编译器则是nasm（左上角可以看到nasm-shell）

（5）使用x86汇编语言编写程序

参考了老师给出的例程，我在此基础上增加了设置背景颜色、显示自己的名字和学号、字符移动中变色、字符移动中按照A-Z进行字符的变化这四个特色功能。

代码的编写

这里我采用的是Visual Studio Code



代码的编译

经过查资料可知，nasm的编译命令满足下列格式

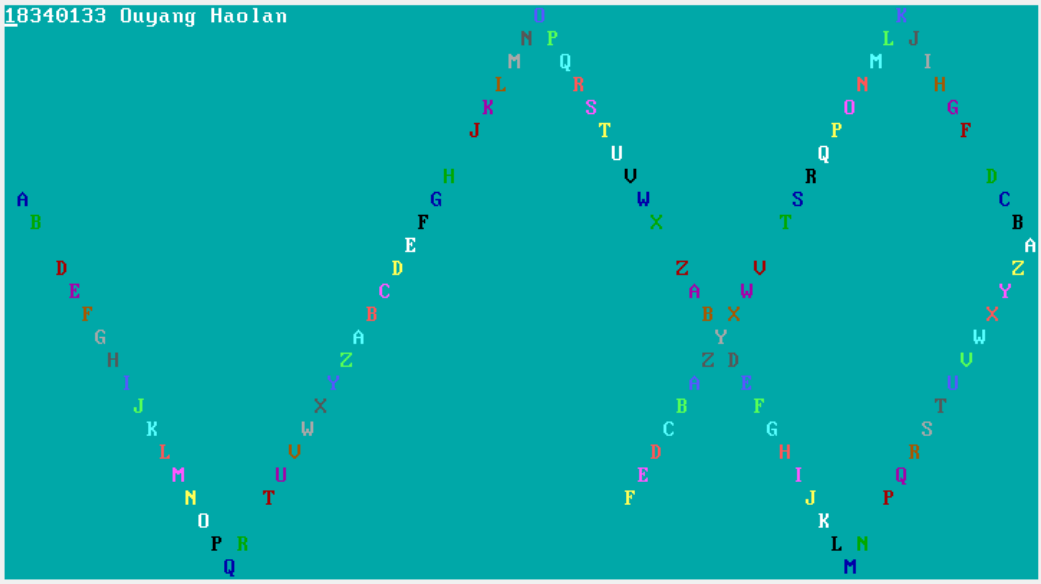
```
nasm –f <format> <filename> [-o output]
```

所以，我打开powershell，输入以上的命令即可编译成功

代码运行

将刚刚生成的1.img作为磁盘软驱输入到虚拟机里进行运行

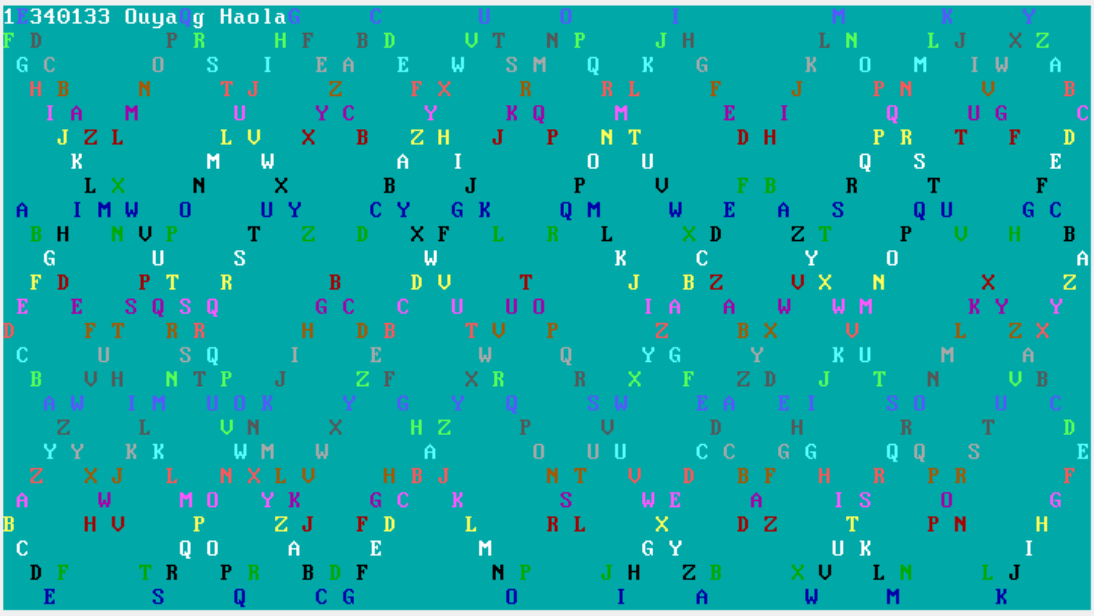
运行结果：



左上角即是我的学号和姓名的拼音。主界面为青色，正是我刚刚设置的。

可以看出，程序的字符从左边射出（刚开始是A），往右下方向运动。在移动过程中，字符按A-Z顺序进行变化。同时，他的颜色也在发生变化。

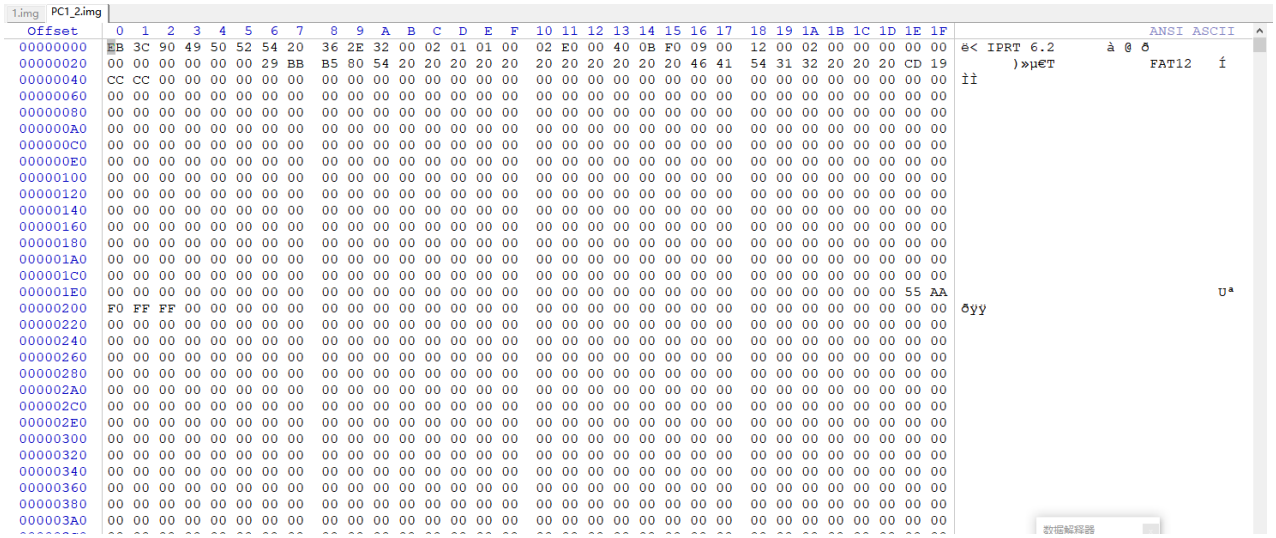
再碰到边界以后，将会反弹继续进行运动。

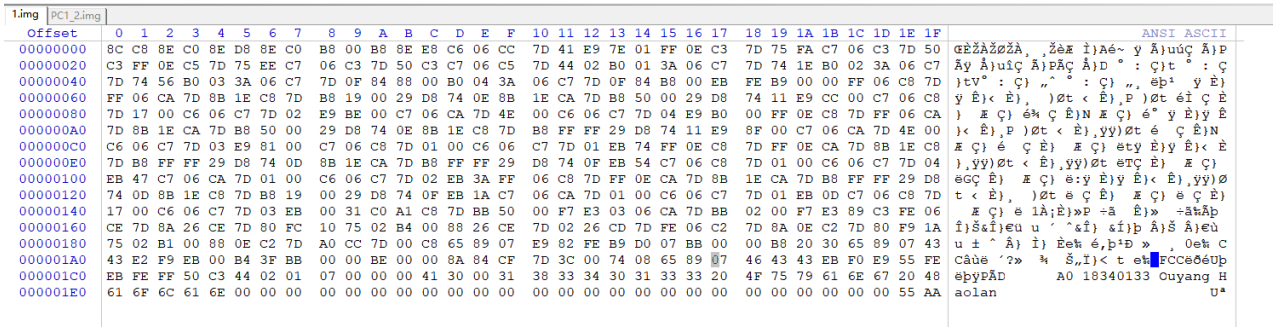


（6）将写好的汇编程序写入第二个软盘映像文件

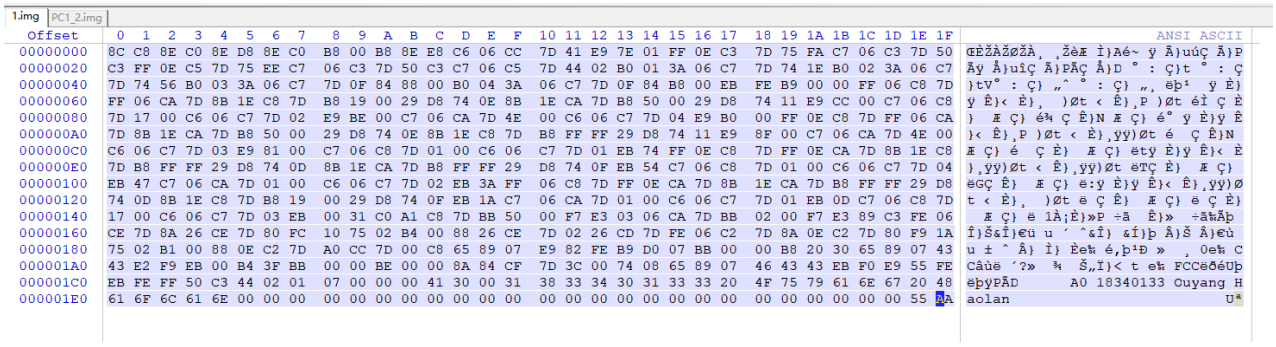
实验工具：WinHex

首先，用WinHex分别打开第二个软盘映像文件PC1_2.img和nasm汇编后的bin文件1.img

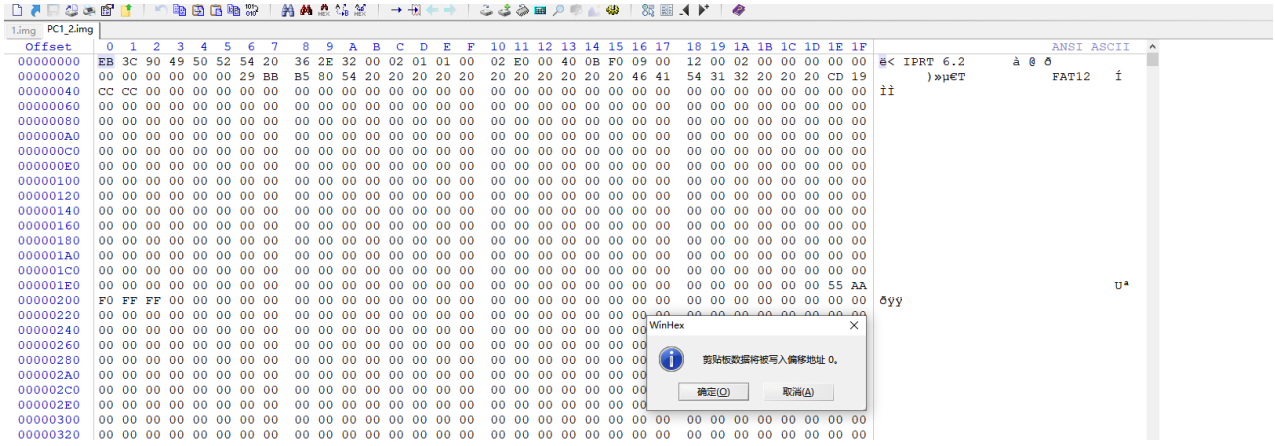




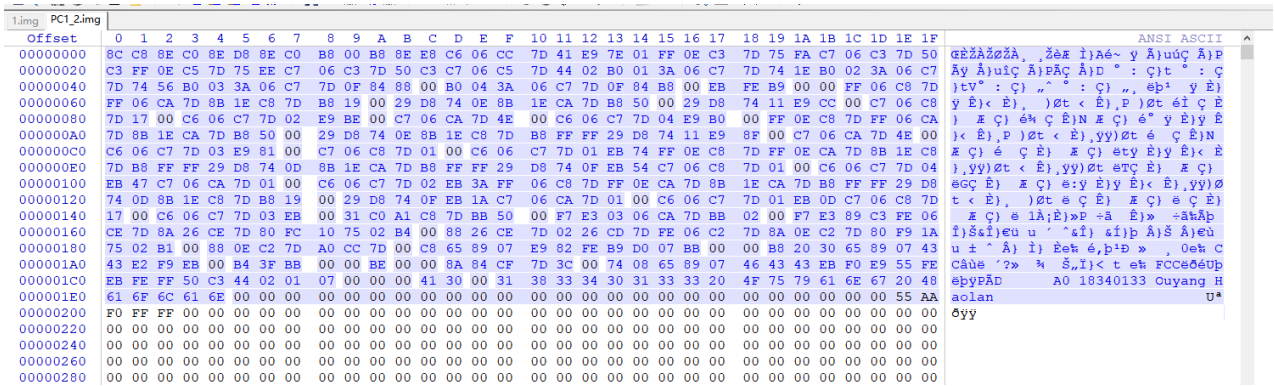
全选1.img的内容，敲击Ctrl-C进行复制



选择PC1_2.img的起始位置，使用Ctrl-V进行复制

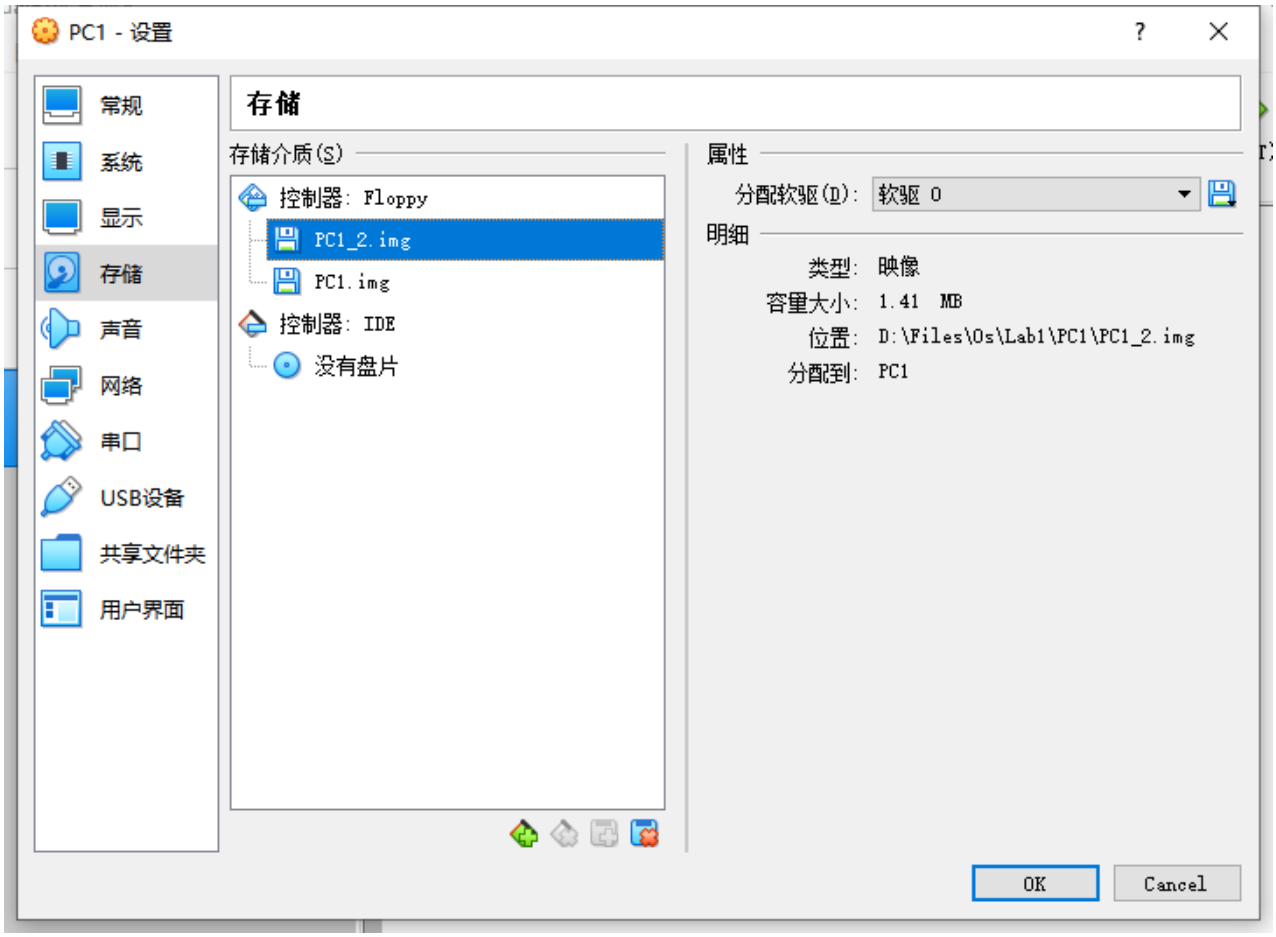


复制后结果：

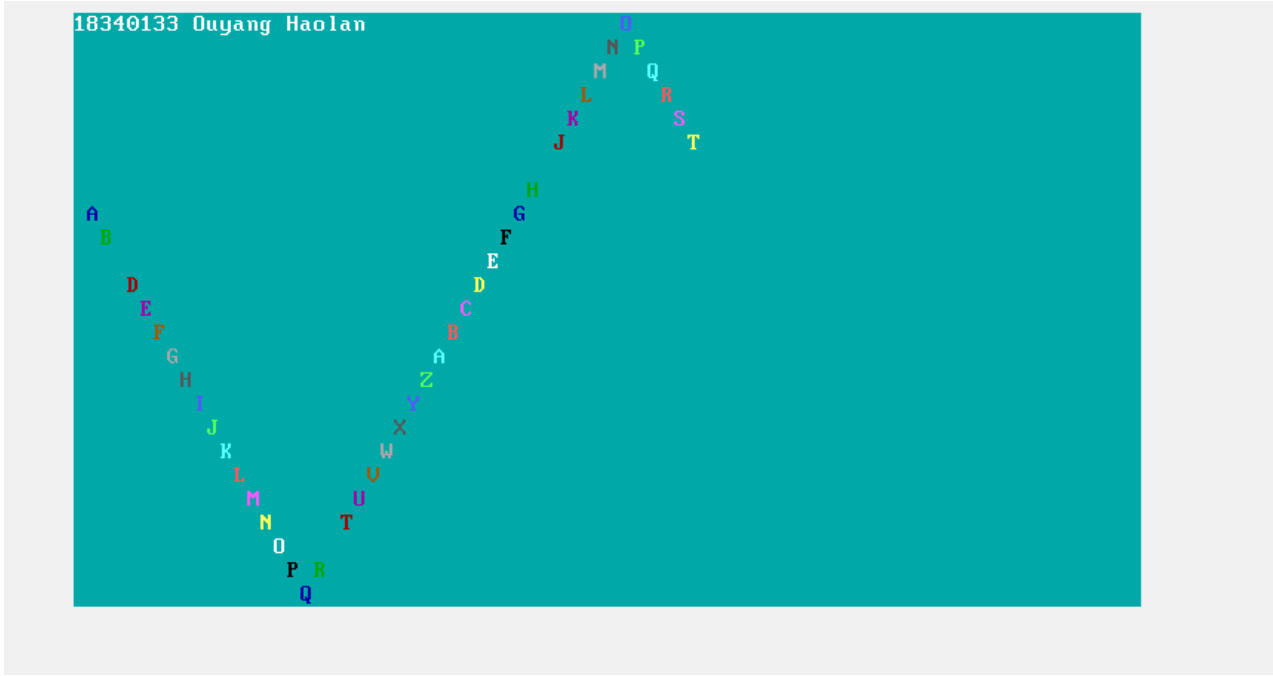


使用虚拟机进行验证

在设置里添加PC1_2.img作为存储器

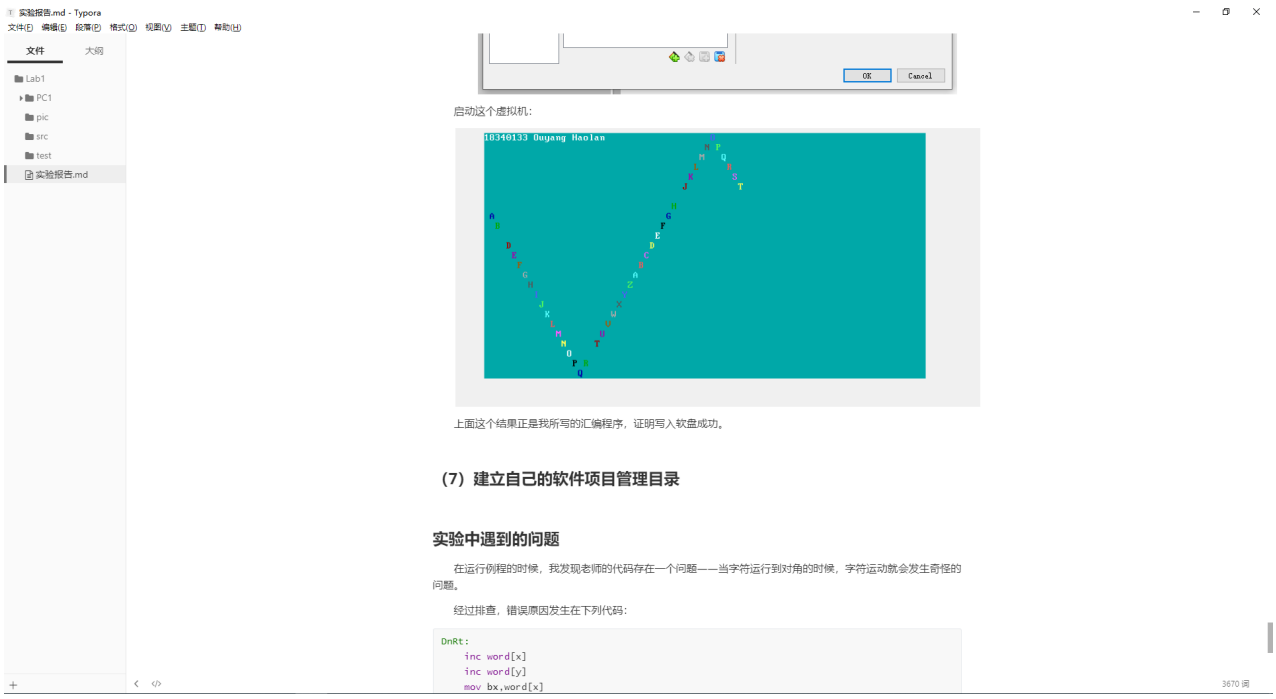


启动这个虚拟机：



上面这个结果正是我所写的汇编程序，证明写入软盘成功。

（7）建立自己的软件项目管理目录



如typora软件的左侧视图可见，我的管理目录树是：

```
-Lab1
     |
     |
     --PC1 虚拟机文件
     |
     |
```

```
--pic 实验截图文件
|
|
--src
|   |
|   |
|   --1.asm 实验汇编文件
--test
|
|
--实验报告.md
```

实验中遇到的问题

在运行例程的时候，我发现老师的代码存在一个问题——当字符运行到对角的时候，字符运动就会发生奇怪的问题。

经过排查，错误原因发生在下列代码：

```
DnRt:
    inc word[x]
    inc word[y]
    mov bx,word[x]
    mov ax,25
    sub ax,bx
      jz  dr2ur
    mov bx,word[y]
    mov ax,80
    sub ax,bx
      jz  dr2dl
    jmp show
dr2ur:
      mov word[x],23
      mov byte[rdul],Up_Rt
      jmp show
dr2dl:
      mov word[y],78
      mov byte[rdul],Dn_Lt
      jmp show
```

在判断越界的时候，没有判断同时出界的清况。当到达对角的时候，只修改了横坐标而忽视了纵坐标，导致代码逻辑出现错误。

但是，在修改代码的时候，我又出现了另一个问题：

```
PS D:\Files\Os\Lab1\src> nasm -f bin 1.asm -o 1.img
1.asm:235: error: TIMES value -16 is negative
PS D:\Files\Os\Lab1\src> nasm -f bin 1.asm -o 1.img
1.asm:230: error: TIMES value -8 is negative
PS D:\Files\Os\Lab1\src> nasm -f bin 1.asm -o 1.img
1.asm:230: error: TIMES value -2 is negative
```

由于引导区的代码长度是有限制的，我在例程原基础上修改的时候，增加了代码长度，导致其超过了510个字节的限制。

经过抉择，我决定通过修改点的起始位置而不修改源代码，使其不会出现进入对角的问题，避免了因为要修复一个bug，从而引出更多问题的窘境。

实验总结

这次是操作系统实验课的第一个实验，总体做起来还比较顺利。因为有老师的例程作为参考，同时在计算机组成原理课上也已经学过了x86汇编语言，在这方面比较有经验。比较不同的是，之前我们的实验环境是MASM，而这个实验里使用的是nasm，代码的部分格式上有区别，实验环境和编译命令也需要重新学，这方面是有点麻烦的，不过我也克服了。不仅如此，之前我们在pc机上显示字符用的是中断命令，而这次实验是直接写入显存。不过，我觉得这次实验更加有趣，因为通过修改代码，写出比较漂亮的界面效果（如实验结果的截图）。特别是能够显示颜色，因为不管是之前在计组课或者甚至是C语言课里交互界面都是黑框，看着很原始。而这次实验可以利用自己的想象力给电脑增添色彩，这是我觉得很好一点。同时，可以从零写一个自己的操作系统，这种操控一切，考虑所有的实现细节，我觉得是很有趣的。正所谓学以致用，通过理论课和网上的资料的学习，再通过实际实践，让我对这些知识理解得更加深入。

不过，在下一个实验里，我希望自己能够学会使用32位的x86汇编语言编写代码，因为现在主流的pc机都是ia32的机器或者是x64即32位和64位电脑，而16位电脑已经逐渐退出了历史的舞台，所以32位甚至64位的x86汇编语言才是未来会更多被使用的。