实验一实验报告

一、 【个人信息】

院系：数据科学与计算机学院
专业：计算机科学与技术（超算方向）
班级：教务 2 班
姓名：劳马东
学号：16337113
邮箱：laomd@mail2.sysu.edu.cn

二、 【实验题目】

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。

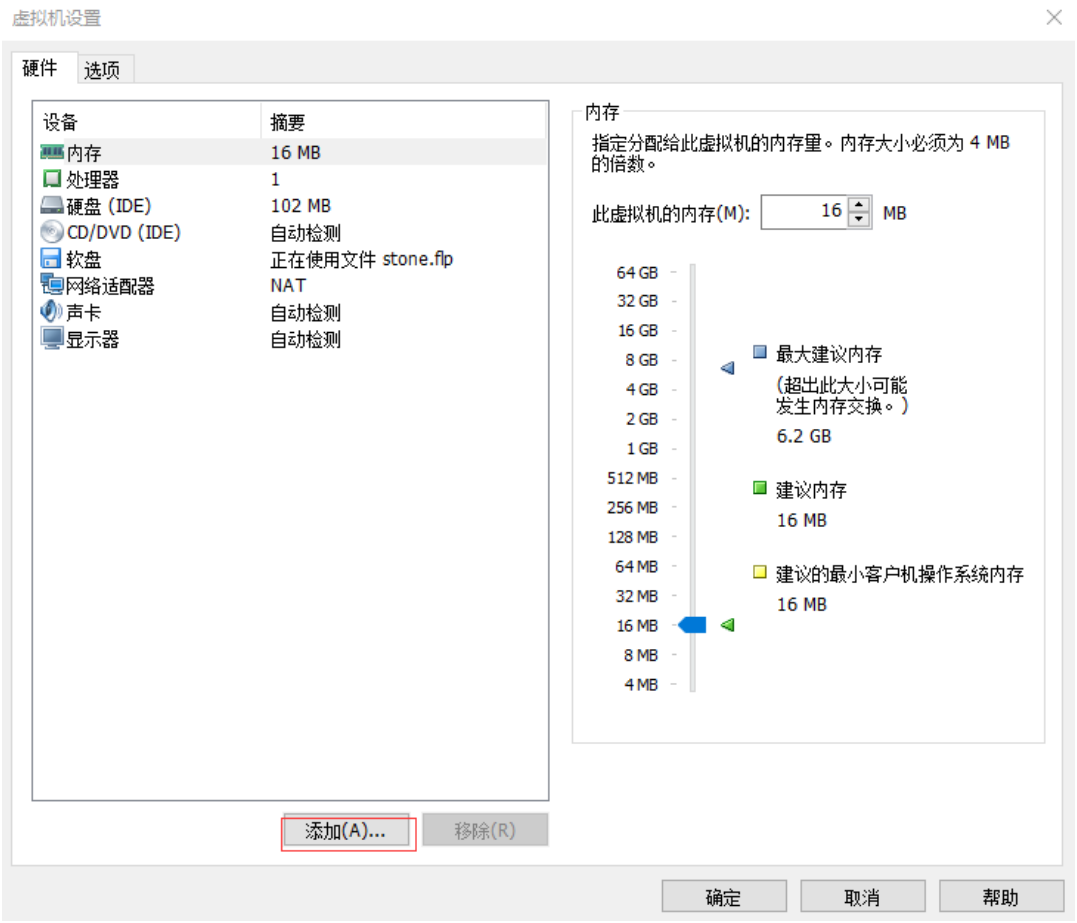
三、 【实验目的】

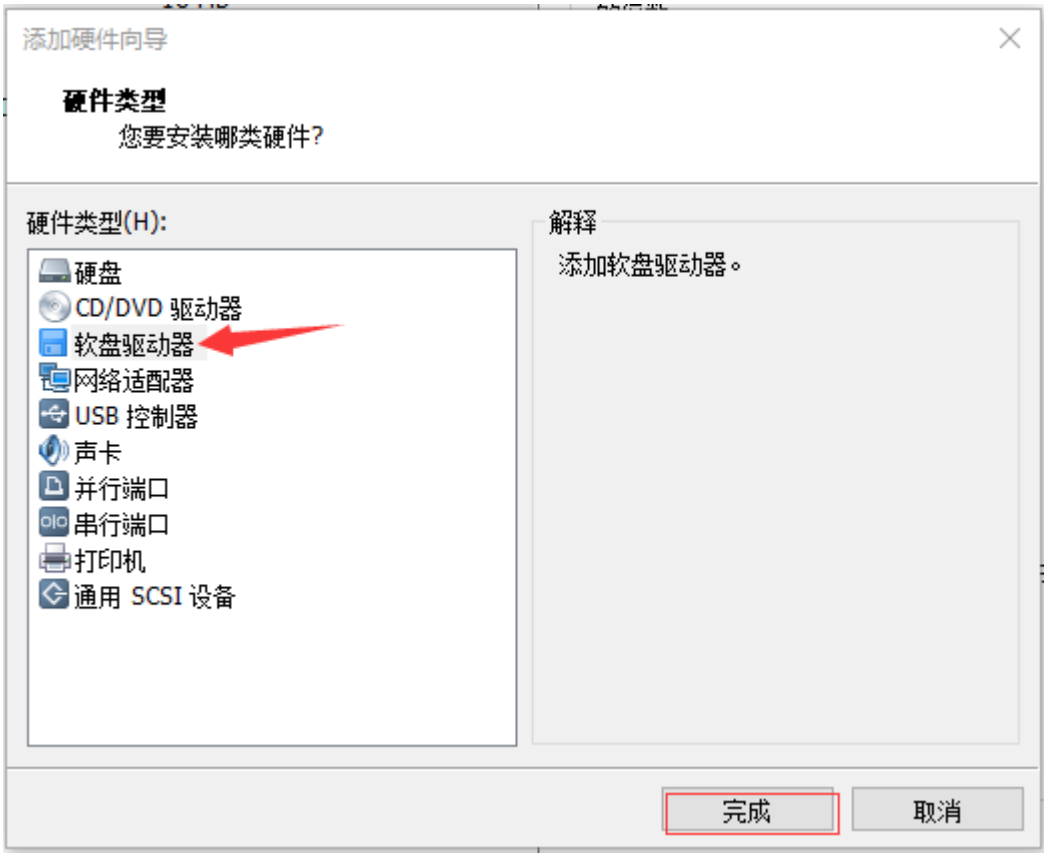
1. 熟悉 GCC+NASM 工具组合的使用；
2. 熟悉 x86 汇编语言语法；
3. 熟悉在裸机上的编程；

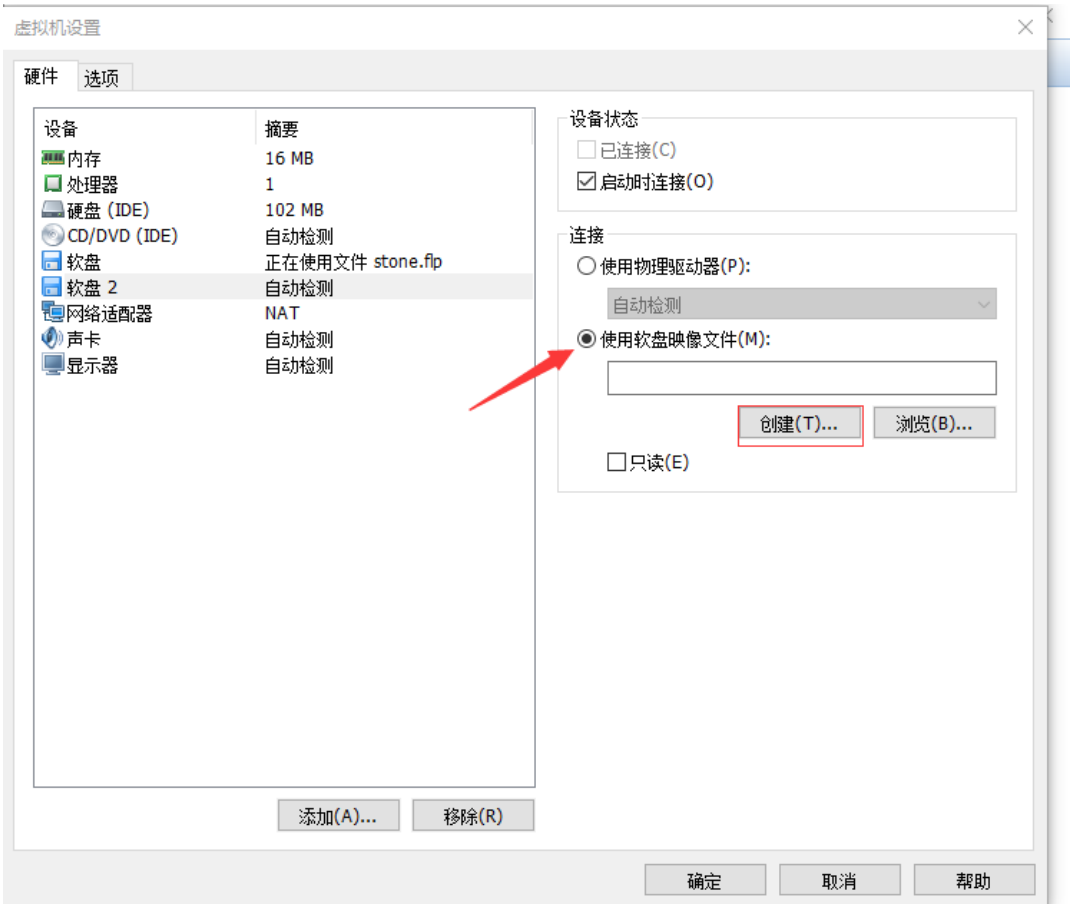
四、 【实验方案】

(一) 虚拟机配置方法

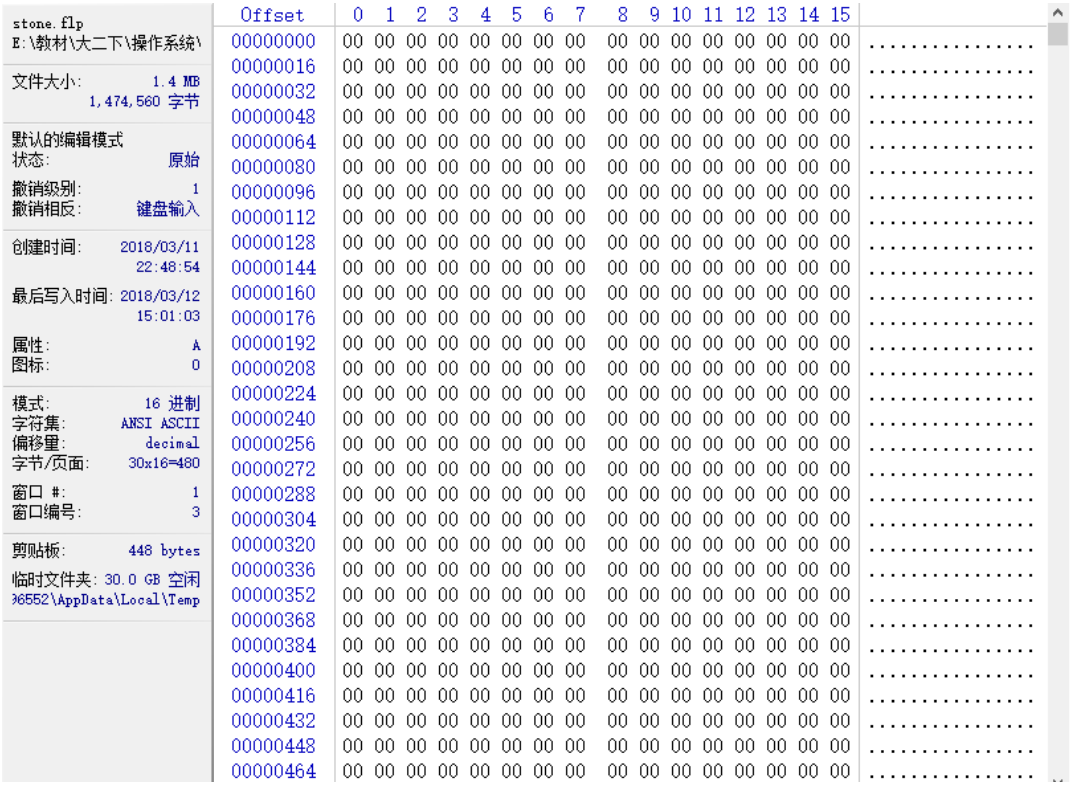
虚拟机的创建，在选择操作系统时候选择“其他-DOS”，最大内存大小改为 0.1GB，弄成单个文件，以下是创建软盘的方法：


虚拟机设置
硬件
选项
设备
摘要
内存
16 MB
处理器
1
硬盘 (IDE)
102 MB
CD/DVD (IDE)
自动检测
软盘
正在使用文件 stone.flp
网络适配器
NAT
声卡
自动检测
显示器
自动检测
内存
指定分配给此虚拟机的内存里。内存大小必须为 4 MB 的倍数。
此虚拟机的内存(M):
16
MB
64 GB
32 GB
16 GB
8 GB
4 GB
2 GB
1 GB
512 MB
256 MB
128 MB
64 MB
32 MB
16 MB
8 MB
4 MB
最大建议内存
(超出此大小可能发生内存交换。)
6.2 GB
建议内存
16 MB
建议的最小客户机操作系统内存
16 MB
添加(A)...
移除(R)
确定
取消
帮助


添加硬件向导
硬件类型
您要安装哪类硬件?
硬件类型(H):
硬盘
CD/DVD 驱动器
软盘驱动器
网络适配器
USB 控制器
声卡
并行端口
串行端口
打印机
通用 SCSI 设备
解释
添加软盘驱动器。
完成
取消



创建好的软盘用 winHex 打开查看是全 0 的初始状态，大小在 1.4MB 左右。



(二) 软件工具与作用

1. **实验支撑环境**

硬件：个人计算机

主机操作系统：Windows

虚拟机软件：VMware，创建虚拟机和引导软盘，运行虚拟机。

i. DOS 虚拟机

2. **实验开发工具**

汇编语言工具：x86 汇编语言

高级语言工具：标准 c 语言

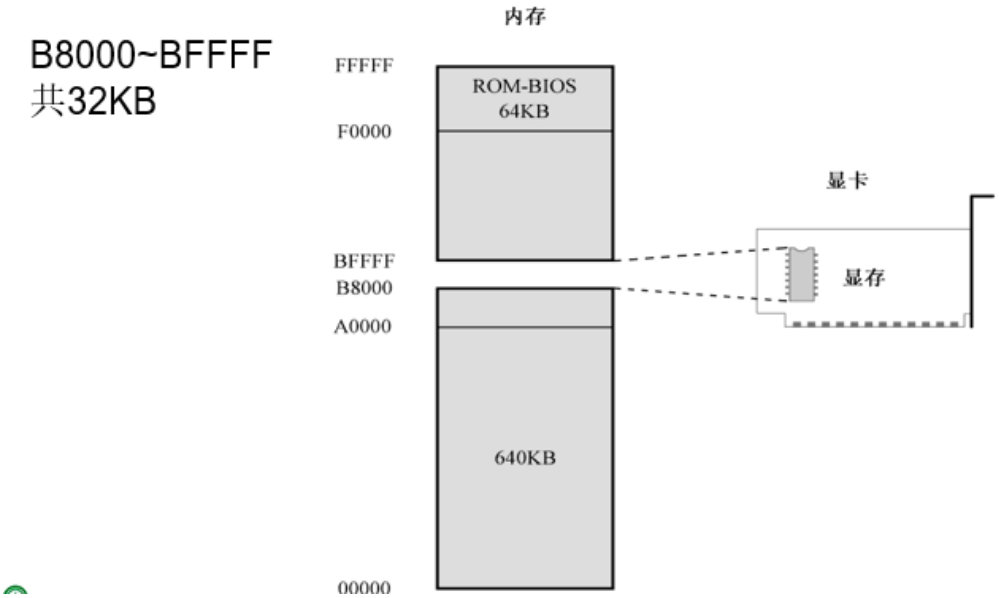
磁盘映像文件浏览编辑工具 WinHex：读写 com 可执行程序或软盘文件

调试工具：Bochs

(三) 实验原理

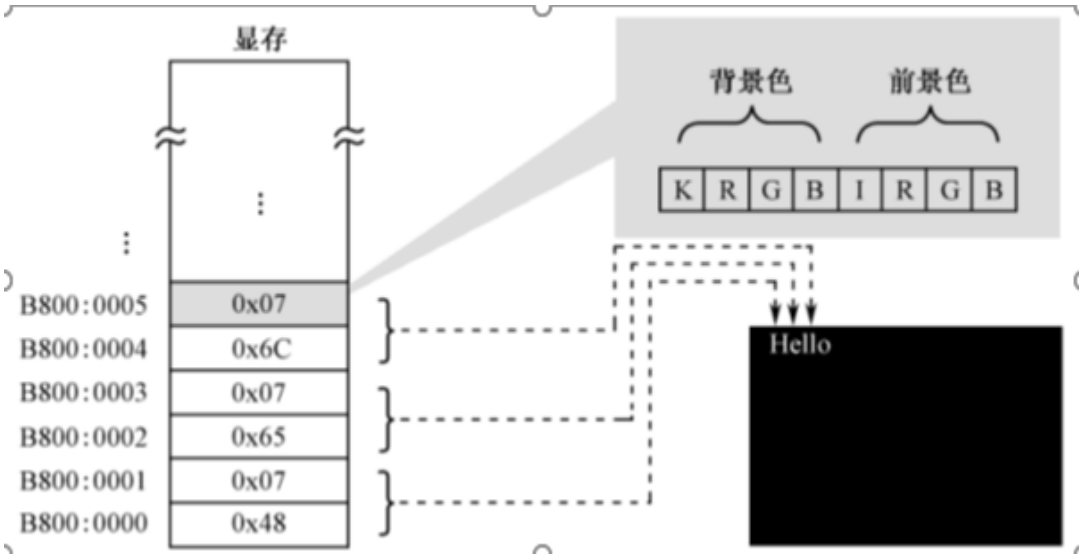
1. **引导程序原理**

当 BIOS 启动时，会将启动设备的第一个扇区（通常是 512B 大小）加载到内存的 0x07c0 处，并开始执行。另外，对于 8086 机器，内存的 0xb800 到 0xbffff 的地址被映射到显卡的显存中，因而，如果想要通过显卡来在屏幕上显示字符的话，必须将字符写到这块内存中。需要注意的是，BIOS 在启动首扇区时，会检查其代码格式，只有当最后两个字节为 0x55、0xAA 时，才是符合要求的引导程序。



2. **字符显示、前后景颜色和闪烁原理**

对于两个相邻的字节，第一个字节是字符本身，把这个字符的 AscaII 码存到显存的相应位置，就能显示字符；第二个字节是字符属性，控制字符的前后景颜色、是否闪烁。如果这个字节的八位假设为 KRGBIRGB，通过设置这八位的 01 值，就可以控制这些属性，具体 01 与属性的关系如下图：



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

3. 程序关键模块

a) 打印个人信息

```
initilize:
    mov ax,0x07c0
    mov ds,ax
    mov ax,0xb800                ;指向文本模式的显示缓冲区
    mov es,ax
    mov si,info
    xor di,di                    ;循环变量di=0
print_str:
    mov bx,infolen
    add bx,bx
    cmp bx,di                    ;while(di<infolen)  打印个人信息
    jb start
    mov al,[si]
    mov byte [es:di],al
    inc si
    add di,2                     ;+2是因为第一个byte是显示字符，第二个字节是颜色
    loop print_str
     info db '16337113 laomd'
     infolen equ $-info
```

首先初始化段寄存器 ds 和 cs(由于不能将立即数直接赋值给段寄存器，也出于安全考虑，需要先把立即数赋给 ax，然后通过 ax 传给段寄存器)，

把字符串首地址给 si，循环变量初始化为 0，然后进入循环。当循环变量小于字符串长度的时候，不断打印字符串中的字符。如上所说，一个字符的显示和属性需要两个字节，因此每次循环时，di 实际上应该加 2，这也是为了以后的可扩展性考虑，比如以上代码还没有设置字符的属性，假如以后要配置属性，只需要加上一行如“mov byte [es:di+1],7”。

b） 主循环

```
30  loop1:
31      dec word[count]                 ; 递减计数变量
32      jnz loop1                       ; >0: 跳转;
33      mov word[count],delay
34      dec word[dcount]                    ; 递减计数变量
35      jnz loop1
36      mov word[count],delay
37      mov word[dcount],ddelay
38
39      mov al,1
40      cmp al,byte[rdul]               ;相等时结果为0
41      jz  DnRt
42      mov al,2
43      cmp al,byte[rdul]
44      jz  UpRt
45      mov al,3
46      cmp al,byte[rdul]
47      jz  UpLt
48      mov al,4
49      cmp al,byte[rdul]
50      jz  DnLt
51      jmp $
```

```
4       Dn_Rt equ 1                     ;D-Down,U-Up,R-right,L-Left
5       Up_Rt equ 2                     ;
6       Up_Lt equ 3                     ;
7       Dn_Lt equ 4                     ;
```

将 rdul 逐个与四个方向（实际上定义为四个数字 1、2、3、4），如果相等（即结果为 0），就执行相应函数，往相应方向运动。

c） 运动和转向

```
53  DnRt:
54      inc word[x]                 ;取出当前行数和列数
55      inc word[y]
56      mov bx,word[x]
57      mov ax,25
58      sub ax,bx
59      jz  dr2ur                   ;如果行数是25，即已经下面出界，就转为右上方向
60      mov bx,word[y]
61      mov ax,80
62      sub ax,bx
63      jz  dr2dl                   ;如果列数是80，即已经右边出出界，就转右下方向
64      jmp show                    ;如果前面两种情况都不是，就正常显示
65  dr2ur:
66      mov word[x],23
67      mov byte[rdul],Up_Rt
68      jmp show
```

相关解释在注释中已有说明，其余方向的转向和运动类似。

五、 【实验过程】

（一） 改正编译错误

第一次用 nasm 编译器编译 stone.asm 时，发现有比较多的编译错误。如下图：

```
E:\教材\大二下\操作系统\实验\project1\OStoolsDOS>na stone
stone.asm:11: error: attempt to define a local label before any non-local labels
stone.asm:13: error: parser: instruction expected
stone.asm:14: error: parser: instruction expected
stone.asm:162: error: symbol `code' redefined
stone.asm:162: error: parser: instruction expected
stone.asm:163: error: parser: instruction expected
E:\教材\大二下\操作系统\实验\project1\OStoolsDOS>
```

查看源程序文件的 11、13、14 行，如下图：

```
    .386
    org 100h                    ; 程序加载到100h，可用于生成COM
    ASSUME cs:code,ds:code
code SEGMENT
```

由于初学 x86 汇编（上学期的计算机组成原理学习的是 MIPS 汇编语言），对这些语法不熟悉，于是请教了 15 级的一个师姐，她说这些代码都是用于生成 com 文件的，在引导程序中不需要，可以注释掉，自己也上百度查了查每句话的作用，发现确实这么回事，于是就去掉了这几句以后后面的 code ends、end start。去掉之后再编译，发现又出现了新的错误：

```
E:\教材\大二下\操作系统\实验\project1\OStoolsDOS>na stone
stone.asm:149: error: invalid combination of opcode and operands
E:\教材\大二下\操作系统\实验\project1\OStoolsDOS>
```

翻到 149 行，代码是这样的：

```
148     mov al,byte[char]           ;  AL = 显示字符值（默认值为20h=空格符）
149     mov es:[bx],ax          ;  显示字符的ASCII码值
```

似曾相识！这不是老师在实验课上给我们展示的打印@的代码吗？可是又有点不一样，老师的代码是这样的：

- 把字符的ASCII码和属性编码送到对应的显存中

```
mov byte [es:0x00],'L'
mov byte [es:0x01],0x07
```

- 或在屏幕中央显示一个“@”

```
mov byte [es:(12X80+39)X2],'@'；12行39列显示@
mov byte [es: (12X80+39)X2+1],0x07
```

由上面一行的“mov ah,0FH”的注释可以知道，这一行的作用是调颜色和闪烁，相当于老师代码的第二个 mov，因此问题必然出现在把字符送到显存的那一行！对比一看，原来有语法错误，访问显存应该使用“段地址：偏移地

址”的形式，而且这部分用中括号括起来，而编译错误那一行把段地址放在中括号外了，而且送的字符不应该是 ax，而是 al。改正之后，编译通过！编译之后生成的 com 文件用 winHex 查看，如下：

stone.com

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	8C	C8	8E	C0	8E	D8	8E	C0	B8	00	B8	8E	E8	C6	06	6E
00000016	01	41	FF	0E	65	01	75	FA	C7	06	65	01	50	C3	FF	0E
00000032	67	01	75	EE	C7	06	65	01	50	C3	C7	06	67	01	44	02
00000048	B0	01	3A	06	69	01	74	1E	B0	02	3A	06	69	01	74	53
00000064	B0	03	3A	06	69	01	0F	84	85	00	B0	04	3A	06	69	01
00000080	0F	84	B5	00	EB	FE	FF	06	6A	01	FF	06	6C	01	8B	1E
00000096	6A	01	B8	19	00	29	D8	74	0E	8B	1E	6C	01	B8	50	00
00000112	29	D8	74	11	E9	CC	00	C7	06	6A	01	17	00	C6	06	69
00000128	01	02	E9	BE	00	C7	06	6C	01	4E	00	C6	06	69	01	04
00000144	E9	B0	00	FF	0E	6A	01	FF	06	6C	01	8B	1E	6C	01	B8
00000160	50	00	29	D8	74	0E	8B	1E	6A	01	B8	FF	FF	29	D8	74
00000176	11	E9	8F	00	C7	06	6C	01	4E	00	C6	06	69	01	03	E9
00000192	81	00	C7	06	6A	01	01	00	C6	06	69	01	01	EB	74	FF
00000208	0E	6A	01	FF	0E	6C	01	8B	1E	6A	01	B8	FF	FF	29	D8
00000224	74	0D	8B	1E	6C	01	B8	FF	FF	29	D8	74	0F	EB	54	C7
00000240	06	6A	01	01	00	C6	06	69	01	04	EB	47	C7	06	6C	01
00000256	01	00	C6	06	69	01	02	EB	3A	FF	06	6A	01	FF	0E	6C
00000272	01	8B	1E	6C	01	B8	FF	FF	29	D8	74	0D	8B	1E	6A	01
00000288	B8	19	00	29	D8	74	0F	EB	1A	C7	06	6C	01	01	00	C6
00000304	06	69	01	01	EB	0D	C7	06	6A	01	17	00	C6	06	69	01
00000320	03	EB	00	31	C0	A1	6A	01	BB	50	00	F7	E3	03	06	6C
00000336	01	BB	02	00	F7	E3	89	C3	B4	0F	A0	6E	01	26	88	07
00000352	E9	AF	FE	EB	FE	50	C3	44	02	01	07	00	00	00	41	

复制到软盘文件中，在末尾加上 55、AA：

stone.flp | showinfo.com | stone.com

stone.flp
E:\教材\大二下\操作系统\

文件大小: 1.4 MB
1,474,560 字节

默认的编辑模式
状态: 原始
撤销级别: 2
撤销相反: 键盘输入

创建时间: 2018/03/11 22:48:54
最后写入时间: 2018/03/12 11:03:53

属性: A
图标: 0

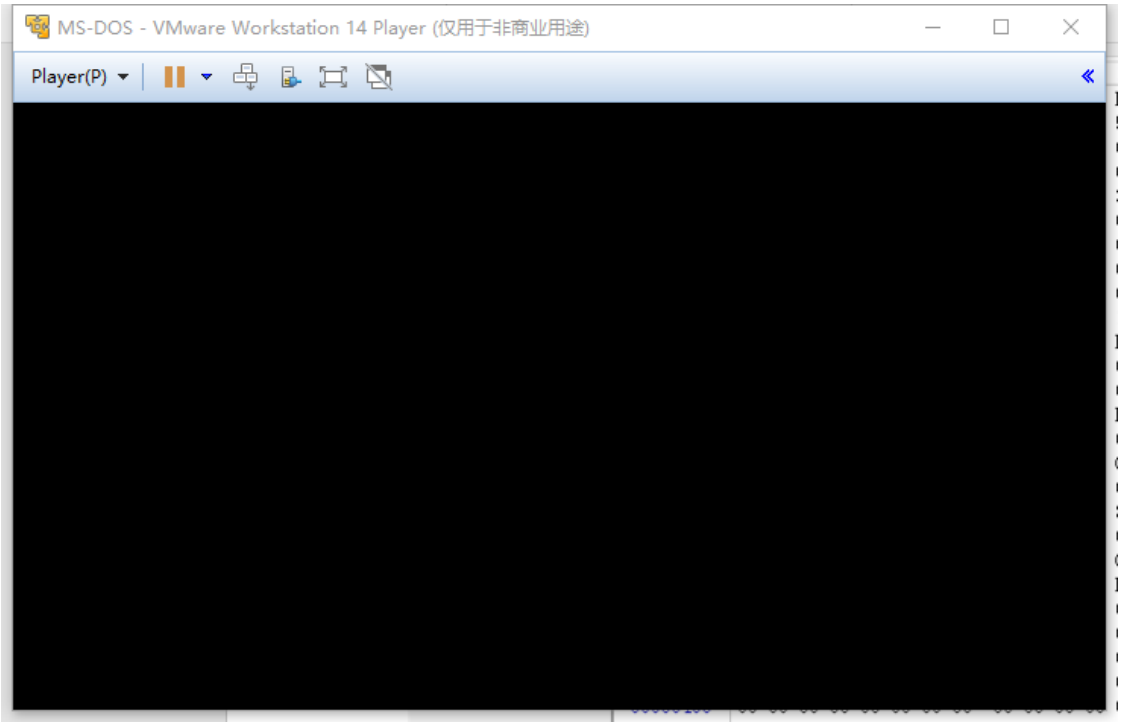
模式: 16 进制
字符集: ANSI ASCII
偏移量: decimal
字节/页面: 30x16=480

窗口 #: 1
窗口编号: 3

剪贴板: 367 bytes
临时文件夹: 29.9 GB 空闲
)6552\AppData\Local\Temp

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	8C	C8	8E	C0	8E	D8	8E	C0	B8	00	B8	8E	E8	C6	06	6E
00000016	01	41	FF	0E	65	01	75	FA	C7	06	65	01	50	C3	FF	0E
00000032	67	01	75	EE	C7	06	65	01	50	C3	C7	06	67	01	44	02
00000048	B0	01	3A	06	69	01	74	1E	B0	02	3A	06	69	01	74	53
00000064	B0	03	3A	06	69	01	0F	84	85	00	B0	04	3A	06	69	01
00000080	0F	84	B5	00	EB	FE	FF	06	6A	01	FF	06	6C	01	8B	1E
00000096	6A	01	B8	19	00	29	D8	74	0E	8B	1E	6C	01	B8	50	00
00000112	29	D8	74	11	E9	CC	00	C7	06	6A	01	17	00	C6	06	69
00000128	01	02	E9	BE	00	C7	06	6C	01	4E	00	C6	06	69	01	04
00000144	E9	B0	00	FF	0E	6A	01	FF	06	6C	01	8B	1E	6C	01	B8
00000160	50	00	29	D8	74	0E	8B	1E	6A	01	B8	FF	FF	29	D8	74
00000176	11	E9	8F	00	C7	06	6C	01	4E	00	C6	06	69	01	03	E9
00000192	81	00	C7	06	6A	01	01	00	C6	06	69	01	01	EB	74	FF
00000208	0E	6A	01	FF	0E	6C	01	8B	1E	6A	01	B8	FF	FF	29	D8
00000224	74	0D	8B	1E	6C	01	B8	FF	FF	29	D8	74	0F	EB	54	C7
00000240	06	6A	01	01	00	C6	06	69	01	04	EB	47	C7	06	6C	01
00000256	01	00	C6	06	69	01	02	EB	3A	FF	06	6A	01	FF	0E	6C
00000272	01	8B	1E	6C	01	B8	FF	FF	29	D8	74	0D	8B	1E	6A	01
00000288	B8	19	00	29	D8	74	0F	EB	1A	C7	06	6C	01	01	00	C6
00000304	06	69	01	01	EB	0D	C7	06	6A	01	17	00	C6	06	69	01
00000320	03	EB	00	31	C0	A1	6A	01	BB	50	00	F7	E3	03	06	6C
00000336	01	BB	02	00	F7	E3	89	C3	B4	0F	A0	6E	01	26	88	07
00000352	E9	AF	FE	EB	FE	50	C3	44	02	01	07	00	00	00	41	00
00000368	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000384	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA
00000400	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

运行虚拟机：



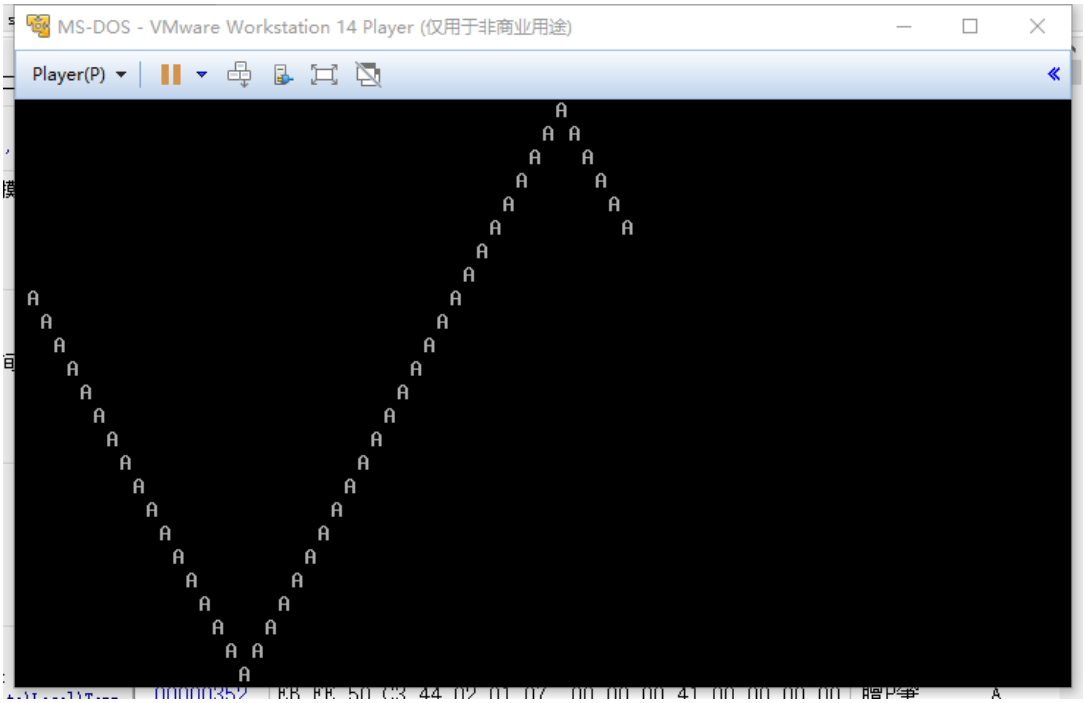
什么都没有！问题出在哪里？

```
21      mov ax,0B800h               ; 文本窗口显存起始地址
22      mov gs,ax                   ; GS = B800h
```

由于这里是把 0b800h 赋值给 gs 段寄存器，而下面用的是 es 寄存器，显存映射错误，而且没有把程序加载到内存的 0x07c0 处，当然无法显示，改成这样：

```
15  start:
16      ;xor ax,ax                  ; AX = 0    程序加载到0000：100h才能正确执行
17      mov ax,0x07c0
18      mov ds,ax                   ; DS = CS
19      mov ax,0B800h               ; 文本窗口显存起始地址
20      mov gs,ax                   ; GS = B800h
```

就可以正常运行了：



（二） 字符 A 按层闪烁和变色

查看老师提供的源程序可以发现，显示字符的代码逻辑大部分在 show 函数中，而且通过注释掉“mov ah”这一行发现，运行之后颜色改变，说明 ah 寄存器是控制颜色和闪烁的。于是就写了一个类似于 if-else 的逻辑：

```
151 show:
152     xor ax,ax                   ; 计算显存地址
153     mov ax,word[x]
154     mov bx,80
155     mul bx
156     add ax,word[y]
157     mov bx,2
158     mul bx
159     mov bx,ax
160     mov ah, byte[x]             ;把行数赋值给ah，随机背景和前景
161     add ah, 1                   ;加1是为了防止在第0行出现字的颜色是黑色而无法看到
162     mov cx, word[x]
163     and cx, 1                   ;和1相与相当于%2，即奇数行闪，偶数行不闪
164     jz bushan
165     jnz shan
166 bushan:
167     jmp show2
168 shan:
169     or ah, 10000000b            ;通过或操作把K置1，使字符闪烁
170     jmp show2
```

相关语句的作用在注释中已经说明得比较清楚。

（三） 同时控制两个字符

想法是通过在 A 字符的后一列再控制一个字符，因此在把字符 A 送到缓存之后，再次重复操作，就可以把字符 B“同时”送到显存，这样就在屏幕上显示了两个字符。

```
show2:
    mov al,byte[char1]          ;  AL = 显示字符值（默认值为20h=空格符）
    mov word [gs:bx],ax         ;  显示字符的ASCII码值
    mov al,byte[char2]
    mov word [gs:(bx+2)],ax         ;  显示字符的ASCII码值
    jmp loop1
end:
    jmp $                   ; 停止画框，无限循环
```

```
    char1 db 'A'
    char2 db 'B'
```

（四） 打印个人信息

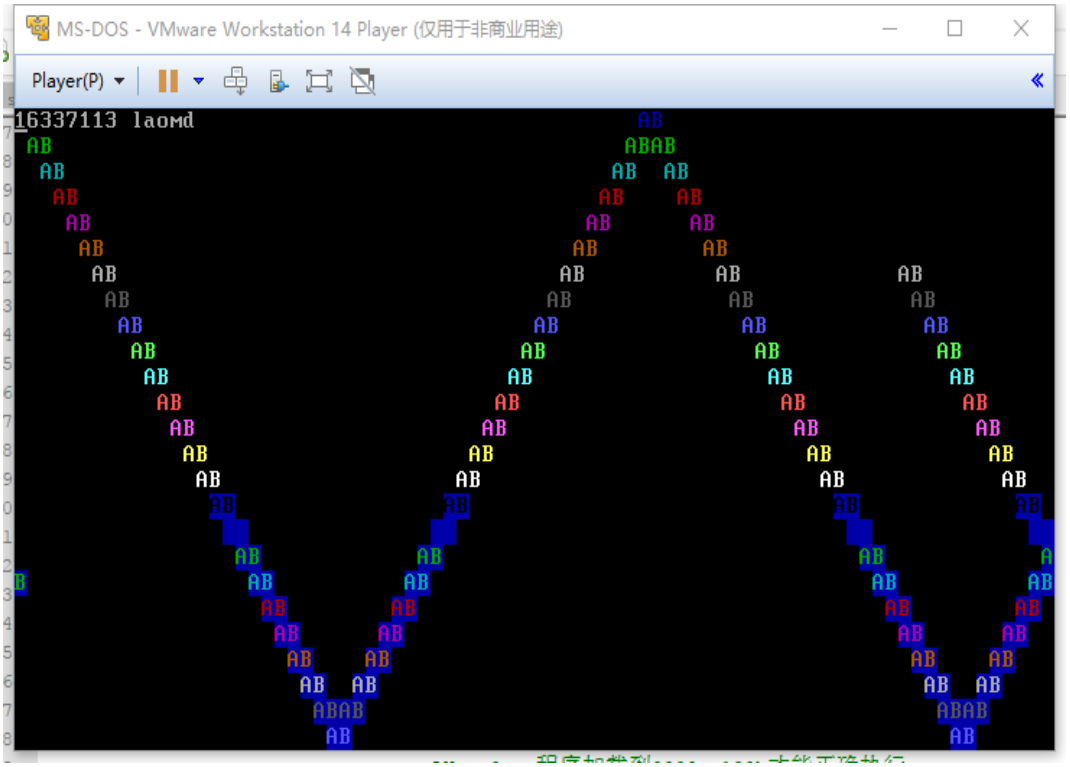
一开始是采用硬编码的形式，即用很多条类似“mov byte [es:0],’1’”、“mov byte [es:1],7”这样的语句来打印个人信息，但这样带来的问题：一是可扩展性极差，如果以后要显示其他字符串，又要重复一遍这个繁琐的操作；二是容易把源程序文件弄得很大（不能超过 512B），实际上在做了以上的功能之后，再采用这种方式，文件已经比 512B 大得多了，这样一来就没法添加其他代码。于是在百度上搜索 NASM 的 x86 汇编语言怎么输出字符串，CSDN 找到的示例程序如下：

```
BOOTSEG     equ 0x07c0
DISPLAYSEG  equ 0xb800

_start:

    ;初始化数据段，使其指向段基址0X7C0处，即Boot代码被加载的地方
    mov     ax, BOOTSEG
    mov     ds, ax

    ;将文本显示内存段基址 放在ES中，供后面显示字符使用
    mov     ax, DISPLAYSEG
    mov     es, ax

    mov     cx, msglen
    mov     si, message
    xor     di, di

print_str:

    mov     al, [si]
    mov     [es:di], al
    inc   si
    inc     di
    mov byte    [es:di], 0x07
    inc     di
    loop    print_str

    jmp near $      ;死循环，程序在此处终止


    message     db "Loading System...", 13, 10
    msglen      db $ - message

    times 510-($-$$) db 0
    dw  0xaa55
```

编译运行一遍发现没什么问题，于是模仿着写了一段。由于这个示例程序没有循环停止的逻辑，而我们的程序需要在打印个人信息之后转而执行字符 A 弹射的代码，因此就对这段代码自己做了一些修改，如下图：

```
initilize:
    mov ax,0x07c0
    mov ds,ax
    mov ax,0xb800                ;指向文本模式的显示缓冲区
    mov es,ax
    mov si,info
    xor di,di                    ;循环变量di=0
print_str:
    mov bx,infolen
    add bx,bx
    cmp bx,di                    ;while(di<infolen)  打印个人信息
    jb start
    mov al,[si]
    mov byte [es:di],al
    inc si
    add di,2                     ;+2是因为第一个byte是显示字符，第二个字节是颜色
    loop print_str
```

以上代码相当于一个 for 或者 while 循环，进入循环之前先判断循环变量是否超出范围，如果每有，就打印对应位置的字符，否则退出循环，转而执行 start。具体逻辑在注释以及上面的关键模块中已有说明。
程序最终的结构如下图：



可以看到，屏幕左上角显示了个人的学号姓名拼音，其余部分字符 AB 同时并排弹射，而且在不同的行上，字符的颜色、背景等不一样。

六、 【实验总结】

1. 一开始编译源代码文件的时候看到这么多的编译错误摸不着头脑，因为没怎么接触过 x86 汇编语言，对一些语法的报错信息没法很快明白，迅速找到解决方法。一开始是向高年级的同学请教，但是因为她也不是特别清楚程序的整体，就没法做到帮我解决遇到的所有问题。于是就把老师给的 PPT 看了一

遍，写了一些简单的程序，通过所谓的“控制变量法”一行行地注释，看会出现什么结果，来猜出每行代码的作用，并通过百度搜索相关知识验证。个人建议老师以后在给代码的时候，如果的确是让我们来改正一些编译错误可以明说，因为不说容易让人误以为是工具不行或者平台不支持，这就容易浪费时间在排错上。就像这次实验，一开始大家在群里讨论为什么会出现编译错误，很多人都说是 nasm 编译不行，要该用 masm，但是后来又有人反应 masm 编译也同样有语法错误。

2. 关于字体变色，一开始为了找到哪一行代码是调整颜色的，也是通过一行行注释，当然不是每一行都选，只是挑那些看起来像的，范围也大部分局限在 show 函数中。当注释掉“mov ah”那一行的时候，运行结果发生了惊人的变化！字符 A 在弹的过程中居然伴随着分层变色，可是自己还没写字体变色的逻辑呀！而且有一个很奇怪的现象，左半边的字符色层比右半边的低一层，左半边也没有运动到最顶端。于是认真分析了其余代码，发现并没有关于颜色变化的代码，只是一些撞到边缘后弹射的逻辑。后来也请教了其他同学，发现他们也有同样的现象，但是他们也没能回答上来为什么会这样，希望老师能在评语中给出解释，不胜感激！

七、 【参考文献】

1. Irvine K R. Assembly Language for x86 Processors[M]. 清华大学出版社, 2011.
2. 实验一 PPT
3. http://blog.csdn.net/sivolin/article/details/40859987
   开机输出字符串 - sivolin 的专栏 - CSDN 博客