一、实验目的及实验内容 （本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析） **实验内容：** 编写一个可以自行启动计算机，不需要在现有操作系统环境中运行的程序。 该程序的功能如下。 （1）列出功能选项，让用户通过键盘进行选择，界面如下。 1）reset pc；重新启动计算机 2）start system；引导现有的操作系统 3）clock；进入时钟程序 4）set clock；设置时间 （2）用户输入“1”后重新启动计算机（提示：考虑：0 单元）。 （3）用户输入“2”后引导现有的操作系统（提示:考虑硬盘 C 的 0 道 0 面 1 扇区）。 （4）用户输入“3”后，执行动态显示当前日期、时间的程序。 显示格式如下：年/月/日时：分：秒 进入此项功能后，一直动态显示当前的时间，在屏幕上将出现时间按秒变化的效果（提示：循环读取 CMOS）。 当按下 Fl 键后，改变显示颜色；按下 Esc 键后，返回到主选单（提示：利用键盘中断）。 （5）用户输入“4”后可更改当前的日期、时间，更改后返回到主选单（提示：输入字符串）。 **实验原理：** 开机后，CPU 自动进入到 FFFF：0 单元处执行，此处有一条跳转指令。CPU 执行该指令后，转去执行 BIOS 中的硬件系统检测和初始化程序。 初始化程序将建立 BIOS 所支持的中断向量，即将 BIOS 提供的中断例程的入口地址登记在中断向量表中。 硬件系统检测和初始化完成后，调用 int19h 进行操作系统的引导。如果设为从软盘启动操作系统，则 int19h 将主要完成以下工作。 （1）控制 0 号软驱，读取软盘 0 道 0 面 1 扇区的内容到 0：7c00； （2）将 CS:IP 指向 0：7c00。 软盘的 0 道 0 面 1 扇区中装有操作系统引导程序。int 19h 将其装到 0：7c00 处后，设置 CPU 从 0：7c00 开始执行此处的引导程序，操作系统被激活，控制计算机。 如果在 0 号软驱中没有软盘，或发生软盘 I/O 错误，int19h 将主要完成以下工作。 （1）读取硬盘 C 的 0 道 0 面 1 扇区的内容到 0：7c00； （2）将 CS:IP 指向 0：7c00。 **实验建议：** （1）在 DOS 下编写安装程序，在安装程序中包含任务程序； （2）运行安装程序，将任务程序写到软盘上；	小题分：

（3）若要任务程序可以在开机后自行执行，要将它写到软盘的 0 道 0 面 1 扇区上。如果程序长度大于 512 个字节，则需要用多个扇区存放，这种情况下，处于软盘 0 道 0 面 1 扇区中的程序就必须负责将其他扇区中的内容读入内存。	
二、实验环境及实验步骤 （本次实验所使用的器件、仪器设备等的情况；具体的实验步骤）	小题分：

实验环境：

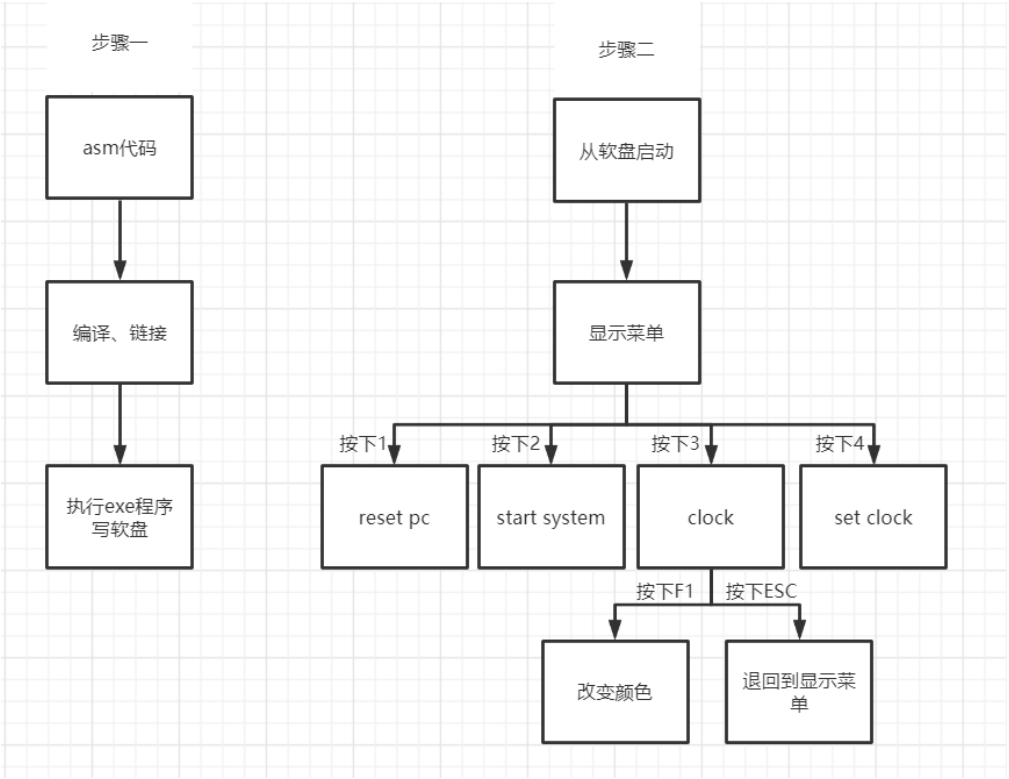
Dosbox0.74-3，win xp

实验步骤：

接下来将按照如下几个步骤详细分析：

1. 代码总架构
2. 引导程序
3. 显示菜单
4. 菜单选项 1　reset pc；重新启动计算机
5. 菜单选项 2　start system；引导现有的操作系统
6. 菜单选项 3　clock；进入时钟程序
7. 菜单选项 4　set clock；设置时间

该计算机使用流程图如下：



1. 代码总架构

主程序如下所示：

```
assume cs:code

0 references
code segment
0 references
start:
        call copy_introduce
        call copy_boot_disk
        mov ax,4c00h
        int 21h
0 references
copy_introduce:…
0 references
copy_boot_disk:…
;=======================================引导程序开始=========================
0 references
introduce:…
0 references
introduce_end:nop
;=================================引导程序结束=====================================

;=================================主代码开始=======================================
0 references
Boot:…
0 references
Boot_end:nop
;=================================主代码结束=======================================
code ends
end start
```

主程序①
会调用两个函数 copy_introduce 和 copy_boot_disk ②
其中 copy_introduce 函数会把③的代码复制到软盘的 0 面 0 道 1 扇区
copy_boot_disk 函数会把④的代码复制到软盘的 0 面 0 道 2 和 3 扇区

②中两个函数如下：

```
copy_introduce:
        mov bx,cs
        mov es,bx
        mov bx,offset introduce

        mov al,1
        mov ch,0
        mov cl,1
        mov dl,0
        mov dh,0
        mov ah,3
        int 13h
        ret
```

```
copy_boot_disk:
        mov bx,cs
        mov es,bx
        mov bx,offset Boot

        mov al,2
        mov ch,0
        mov cl,2
        mov dl,0
        mov dh,0
        mov ah,3
        int 13h
        ret
```

两个函数都调用了 int13，代码基本相同，下面说一下入口参数

（ah）=int13h的功能号（3表示写扇区）

（al）=写入的扇区数

（ch）=磁道号（cl）=扇区号

（dh）=磁头号（面）

（dl）=驱动器号

软驱从 0 开始，0：软驱 A，1：软驱 B；

硬盘从 80h 开始，80h：硬盘 C，81h：硬盘 D

es:bx 指向将写入磁盘的数据

2. 引导程序

这里主要是上面说的第③部分。

引导程序代码如下：

```
introduce:
        call save_old_int9
        call copy_Boot_fromdisk

        mov bx,0
        push bx
        mov bx,7e00h
        push bx
        retf
;-----------------------------
0 references
copy_Boot_fromdisk:…
;-----------------------------------
 0 references
save_old_int9:…
;----------------------
        db 512 dup (0)
0 references
introduce_end:nop
```

会调用 save_old_int9 和 copy_Boot_fromdisk 两个函数，
最后使用 retf，把 cs：ip 修改为 0:7e00h 开始执行。
注意这里有个 db 512 dup (0)，意思是说在该引导程序复制到 0 面 0 道 1 扇区的时候，由于不够 512 个字节，所以用 0 填充。

引导程序的作用有二：
一是把原来的 int9 中断存储下来，存到 0：200h。后面需要用到 esc 和 f1 按键所以需要写新的 int9。
二是把软盘的 0 面 0 道 2 和 3 扇区复制到 0：7e00 中。
之所以复制到 0：7e00 是因为本身启动时会自动把 0 面 0 道 1 扇区复制到 0：7c00，1 个扇面 512 字节，也就是 200h，所以后面的扇区就是从 0：7e00 开始。

save_old_int9 代码如下：

```
save_old_int9:
        mov bx,0
        mov es,bx

        push es:[9*4]
        pop es:[200h]
        push es:[9*4+2]
        pop es:[202h]
        ret
```

意思是把第 9 号中断存储到 0：200h 里面。

copy_Boot_fromdisk 代码如下：

```
copy_Boot_fromdisk:
        mov bx,0
        mov es,bx
        mov bx,7e00h

        mov al,2
        mov ch,0
        mov cl,2
        mov dl,0
        mov dh,0
        mov ah,2
        int 13h

        ret
```

这里不同的时 ah 为 2 表示从磁盘读取数据，写入到 0：7e00 中。

3. 显示菜单

这里以及下面所有的都说的是第④部分。

```
;=================================主代码开始=================================
0 references
Boot:
    jmp Boot_start
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx定义数据xxxxxxxxxxxxxxxxxxxxxxxxxx
0 references
option0   db 'welcome to 2017301510052-zbw os',0
0 references
option1   db '    1) reset pc',0
0 references
option2   db '    2) start system',0
0 references
option3   db '    3) clock',0
0 references
option4   db '    4) set clock',0

0 references
address_option   dw offset option0 - offset Boot + 7e00h
                 dw offset option1 - offset Boot + 7e00h
                 dw offset option2 - offset Boot + 7e00h
                 dw offset option3 - offset Boot + 7e00h
                 dw offset option4 - offset Boot + 7e00h
0 references
timeinfo   db'Press (F1) to change color     Press (ESC) to return',0
0 references
timestyle   db '00/00/00 00:00:00',0
0 references
timeadress  db 9,8,7,4,2,0
0 references
string_stack   db 12 dup ('0'),0
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

这里定义了一些显示的数据。

然后直接跳转到 Boot_start。如下：

```
Boot_start:
    mov bx,0b800h
    mov es,bx           ;es指向显存
    mov bx,0
    mov ds,bx

    call clear_screen
    call show_option

    jmp short choose_option

    mov ax,4c00h
    int 21h
```

将 es 指向显存，ds 指向 0.
之后调用清屏函数 clear_screen 和显示选项函数 show_option，
最后跳转到 choose_option 等待用户的输入。

3.1

清屏函数 clear_screen 如下：

```
clear_screen:
        push bx
        push dx
        push cx

        mov bx,0
        mov dx,0700h    ;清屏中对字符属性设置应该为07h，而不是0
        mov cx,2000
0 references
clear_screen_loop:
        mov es:[bx],dx
        add bx,2
        loop clear_screen_loop

        pop cx
        pop dx
        pop bx
        ret
```

首先保护现场进行压栈，
bx 指向要输出的地方，由于一个字符占两个字节，所以 bx 每次加 2.
dx 是清屏中的字符，清屏中对字符属性设置应该为 07h，而不是 0
cx 是下面 clear_screen_loop 的循环次数，显示缓冲区 80*25=2000，所以要循环 2000 次。

3.2

显示选项函数 show_option 如下：

```
show_option:
        mov bx,offset address_option - offset Boot + 7e00h
        mov cx,5
        mov di,160*9 + 25*2
0 references
show_option_loop:
        mov si,ds:[bx]
        call showstr
        add di,160
        add bx,2
        loop show_option_loop

        ret
```

这里的意思就是把刚才定义的数据给循环显示出来，有 5 句话，所以 cx=5，循环 5 次。从第 9 行，第 25 列开始输出。

这里调用了 showstr 函数，如下：

```
showstr:
        push cx
        push di
0 references
showstr_loop:
        mov cl,ds:[si]
        cmp cl,0
        je showstr_end
        mov es:[di],cl
        add di,2
        inc si
        jmp short showstr_loop
0 references
showstr_end:
        pop di
        pop cx
        ret
```

首先保护现场，之后循环打印，并判断是否打印完毕，利用

cmp cl,0

je showstr_end

进行判断是否输出完毕，如果输出完毕，则还原现场，不然继续打印。

3.3

到现在显示成功了，开始等待用户输入进而执行哪一部分代码，choose_option 如下:

```
choose_option:
        call clear_buff

        mov ah,0
        int 16h

        cmp al,'1'
        je choose1
        cmp al,'2'
        je choose2
        cmp al,'3'
        je choose3
        cmp al,'4'
        je choose4

        jmp choose_option
```

首先调用清除键盘缓冲区函数 clear_buff；

之后的两行代码

mov ah,0

int 16h

从键盘缓冲区读取输入的字符。

然后判断 1，2，3，4 中的哪一个进而跳转到相应的代码。

最后如果不是 1234 中的任何一个则继续循环这个函数。

其中清除键盘缓冲区函数 clear_buff 如下：

```
clear_buff:
        ;下面两句调用BIOS的16号中断的01号功能，读取键盘状态。
        ;(1)若无按键，零标志位版ZF←1
        ;(2)若有按键，零标志位ZF←0，AH←键扫描码，AL←按权键字符ASCII码
        mov ah,1
        int 16h

        jz clearbuff_end ;JZ指令是在ZF=1时跳转，ZF=0时不跳转

        ;下面两句调用BIOS的16号中断的0号功能,从键盘缓冲区读取一个输入，并将其从缓冲区删除
        mov ah,0
        int 16h

        jmp clear_buff
0 references
clearbuff_end:
        ret
```

代码中的注释写的很清楚了。

4. 菜单选项 1　reset pc；重新启动计算机

```
choose1:
        mov di,160*3
        mov byte ptr es:[di],'1'

        mov bx,0ffffh
        push bx
        mov bx,0
        push bx
        retf
```

代码分两部分：

1. 先在屏幕上打印出“1”。
2. 然后使用 retf 控制 cs:ip 为 ffff:0 单元。
   这是因为开始之后 cpu 自动进入到 ffff:0 单元，所以效果和重新启动计算机效果一样。

5. 菜单选项 2　start system；引导现有的操作系统

```
choose2:
    mov di,160*3
    mov byte ptr es:[di],'2'

    mov bx,0
    mov es,bx
    mov bx,7c00h

    mov al,1
    mov ch,0
    mov cl,1
    mov dl,80h      ;80h代表C盘
    mov dh,0
    mov ah,2
    int 13h

    mov bx,0
    push bx
    mov bx,7c00h
    push bx
    retf
```

代码分三部分：

1. 先在屏幕上打印“2”
2. 把驱动器号设为 80h，表示 C 盘，把该部分的扇区复制到 0：7c00
3. 使用 retf 控制 cs:ip 为 0：7c00 单元。开始执行代码。

6. 菜单选项 3　clock；进入时钟程序

```
choose3:
    mov di,160*3
    mov byte ptr es:[di],'3'
    call show_clock
    jmp Boot_start
```

同样先显示 3，然后调用 show_clock 函数。

```
show_clock:
        call clear_screen
        call show_info
        call set_new_int9
        mov bx,offset timeadress - offset Boot + 7e00h
0 references
show_clock_time:
        mov si,bx
        mov di,160*12+30*2
        mov cx,6
0 references
show_clock_time_loop:
        mov al,ds:[si]
        out 70h,al
        in al,71h

        mov ah,al
        shr ah,1
        shr ah,1
        shr ah,1
        shr ah,1
        and al,00001111b
        add ah,30h
        add al,30h
        mov es:[di],ah
        mov es:[di+2],al
        add di,6
        inc si
        loop show_clock_time_loop

        jmp show_clock_time
```

首先调用 clear_screen 函数清屏。

然后调用 show_info 函数显示提示信息。

然后调用 set_new_int9 函数设置新的 int9 中断，从而接受 esc 和 f1 键盘输入。

最后循环从 CMOS RAM 中读取存储的时间信息。

6.1

clear_screen 函数清屏之前已经讲过了。

6.2

show_info 函数显示提示信息如下：

```
show_info:
        mov si,offset timeinfo - offset Boot + 7e00h
        mov di,160*10+15*2
        call showstr
        mov si,offset timestyle - offset Boot + 7e00h
        mov di,160*12+30*2
        call showstr
        ret
```

就是把 timeinfo 和 timestyle 给打印出来，这两个数据之前定义过了：

```
timeinfo    db'Press (F1) to change color     Press (ESC) to return',0
0 references
timestyle   db '00/00/00 00:00:00',0
```

6.3

然后设置一下新的 int9 中断，从而接受 esc 和 f1 键盘输入：

```
set_new_int9:
        push bx
        push es

        mov bx,0
        mov es,bx

        cli
        mov word ptr es:[9*4],offset newint9 - offset Boot + 7e00h
        mov word ptr es:[9*4+2],0
        sti

        pop es
        pop bx
        ret
```

首先保护现场，随后把该中断的处理程序开始位置填入到 es:[9*4]和 es:[9*4+2]。该程序如下：

```
newint9:
        push ax
        call clear_buff

        in al,60h
        pushf
        call dword ptr cs:[200h]

        cmp al,01h ;是不是esc
        je inesc
        cmp al,3bh ;是不是f1
        jne int9ret
        call change_time_color
```

首先保存现场，压入 ax。
之后清空缓冲区，这个函数上面讲过。
在之后使用 in al,60h，从 60h 端口处读取键盘输入。
判断是不是 esc，如果是，则跳转到下面：

```
inesc:
        pop ax
        add sp,4
        popf
        call set_old_int9
```

即还原现场，并设置回原来的 int9 中断：

```
set_old_int9:
        push bx
        push es

        mov bx,0
        mov es,bx
        cli
        push es:[200h]
        pop es:[9*4]
        push es:[202h]
        pop es:[9*4+2]
        sti

        pop es
        pop bx
        ret
```

我们在程序刚开始就把原来的 int9 中断存在了 0：200h 处，现在反向取回即可。

接着上面判断是不是 F1,
如果不是，则中断停止：

```
int9ret:
        pop ax
        iret
```

否则调用 change_time_color 函数：

```
change_time_color:
        push bx
        push cx
        push es

        mov bx,0b800h
        mov es,bx
        mov cx,17
        mov bx,160*12+30*2+1
0 references
change_time_colors:
        inc byte ptr es:[bx]
        add bx,2
        loop change_time_colors

        pop es
        pop cx
        pop bx
```

首先保护现场把寄存器压栈，
接着循环改变字体属性即可。

6.4

现在设置新 int9 中断设置好了。
接下来开始循环读取 CMOS RAM 中存储的时间信息，在 CMOSRAM 中，存放着当前的时间：年、月、日、时、分、秒。这 6 个信息的长度都为 1 个字节，存放单元为：
秒：0 分：2 时：4 日：7 月：8 年：9
我们在 timeaddress 变量中已经存储了：

```
timeadress  db 9,8,7,4,2,0
```

然后主要做以下两部分工作：

```
        mov bx,offset timeadress - offset Boot + 7e00h
0 references
show_clock_time:
        mov si,bx
        mov di,160*12+30*2
        mov cx,6
0 references
show_clock_time_loop:
        mov al,ds:[si]
        out 70h,al
        in al,71h

        mov ah,al
        shr ah,1
        shr ah,1
        shr ah,1
        shr ah,1
        and al,00001111b
        add ah,30h
        add al,30h
        mov es:[di],ah
        mov es:[di+2],al
        add di,6
        inc si
        loop show_clock_time_loop

        jmp show_clock_time
```

（1）从 CMOS RAM 的 x 号单元读出当前月份的 BCD 码。

要读取 CMOS RAM 的信息，首先用 out 指令要向地址端口 70h 写入要访问的单元的地址，然后用 in 指令从数据端口 71h 中取得指定单元中的数据。

（2）将用 BCD 码表示的月份以十进制的形式显示到屏幕上。

可以看出，BCD 码值=十进制数码值，则 BCD 码值+30h=十进制数对应的 ASCII 码。

从 CMOS RAM 的 8 号单元读出的一个字节中，包含了用两个 BCD 码表示的两位十进制数，高 4 位的 BCD 码表示十位，低 4 位的 BCD 码表示个位。

7. 菜单选项 4　set clock；设置时间

```
choose4:
        mov di,160*3
        mov byte ptr es:[di],'4'
        call set_clock
        jmp Boot_start
```

同样先显示“4”，

随后调用 set_clock 函数：

```
set_clock:
        call clear_screen
        call clear_string_stack
        call show_string_stack
        call get_string
        call set_time
        ret
```

首先调用清屏函数 clear_screen，前面介绍过了。
之后调用 clear_string_stack 函数把栈清空成 30.（这是 ascii 码，转为十进制就是 0）
之后调用 show_string_stack 函数显示输入信息
之后调用 get_string 函数获取键盘输入
最后调用 set_time 函数往 CMOS RAM 中的时间写进去。

7.1
清屏函数 clear_screen，前面介绍过了。

7.2
clear_string_stack 函数如下：

```
clear_string_stack:
        push bx
        push cx
        push es
        push si
        push di

        mov si,offset string_stack - offset Boot + 7e00h
        mov dx,3030h

        mov cx,6
0 references
clear_string_stack_loop:
        mov ds:[si],dx
        add si,2
        loop clear_string_stack_loop

        pop di
        pop si
        pop es
        pop cx
        pop bx
        ret
```

首先保护现场，
之后循环往 string_stack 里面写 3030（这是 ascii 码，转为十进制就是 0），
最后还原现场。

7.3

show_string_stack 函数如下：

```
show_string_stack:
        push si
        push di
        mov si,offset settimeinfo - offset Boot + 7e00h
        mov di,160*10+25*2
        call showstr
        mov si,offset string_stack - offset Boot + 7e00h
        mov di,160*12+30*2
        call showstr
        pop di
        pop si
        ret
```

即调用 showstr 函数显示信息。

7.4

get_string 函数：

```
getstring:
        call clear_buff
        mov ah,0
        int 16h
        cmp al,'0'
        jb notnumber
        cmp al,'9'
        ja notnumber
        call char_push
        call show_string_stack

        jmp getstring
0 references
getstring_end:
        ret
```

首先清空键盘缓冲区，

然后调用 16h 中断读取字符给 ah，

如果 a<0 或者 a>9 那么执行 notnumber 函数：

```
notnumber:
        cmp ah,0eh
        je isbackspace ;判断是否是删除键
        cmp ah,1ch
        je getstring_end ;判断是否是回车键
        jmp getstring
0 references
isbackspace:
        call char_pop
        call show_string_stack
        jmp getstring
;--------------------------
0 references
char_pop:
        cmp bx,0
        je charpopret
        dec bx
        mov byte ptr ds:[si+bx],'0'
0 references
charpopret:
        ret
;--------------------------
0 references
char_push:
        cmp bx,11
        ja charpushret
        mov ds:[si+bx],al
        inc bx
0 references
charpushret:
        ret
```

现在可以简单地确定程序的处理过程如下。

①调用 int16h 读取键盘输入；

②如果是数字，进入字符栈，显示字符栈中的所有字符；继续执行①；

③如果是退格键，从字符栈中弹出一个字符，显示字符栈中的所有字符；继续执行①；

④如果是 Enter 键，向字符栈中压入 0，返回。

7.5

set_time 函数如下：

```
set_time:
        mov bx,offset timeadress - offset Boot + 7e00h
        mov si,offset string_stack - offset Boot +7e00h
        mov cx,6
0 references
set_time_loop:
        mov dx,ds:[si]
        sub dh,30h
        sub dl,30h
        shl dl,1
        shl dl,1
        shl dl,1
        shl dl,1
        and dh,00001111b
        or dl,dh
        mov al,ds:[bx]
        out 70h,al
        mov al,dl
        out 71h,al
        add si,2
        inc bx
        loop set_time_loop
        ret
;-------------------------
```

这一步操作是和从 CMOS RAM 读取时间相反的操作，即写回去。上面读取的时候已经详细分析了，这里就不再分析了。

至此，整个程序就介绍完了。

三、实验过程分析 （详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格、绘制曲线、波形等）	小题分：

1. 刚开始不知道如何利用 asm 文件和系统挂钩，后来看了博客才理解全过程。
2. 在定义字符串时，经常忘记最后以“0”结尾：

```
settimeinfo  db'year/month/day/h/m/s',0
```

3.主程序代码应该写到 0：7e00 开始处，因为 7c00 开始处写的是第一扇区的引导程序。

四、实验结果总结 （对实验结果进行分析，完成思考题目，总结实验的新的体会，并提出实验的改进意见）	小题分：

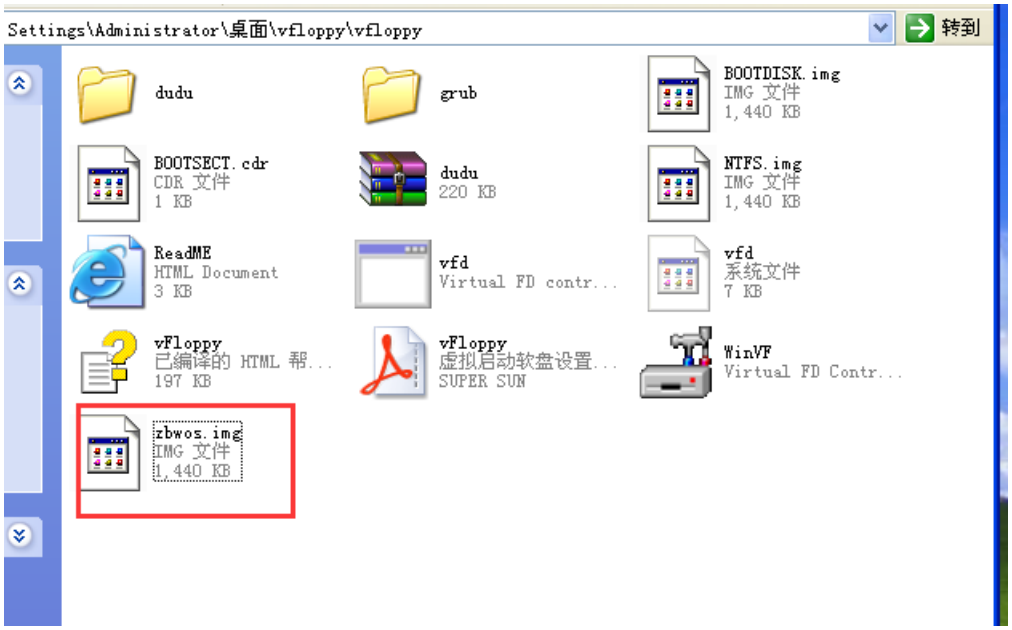
实验结果：

一：写软盘和从软盘启动

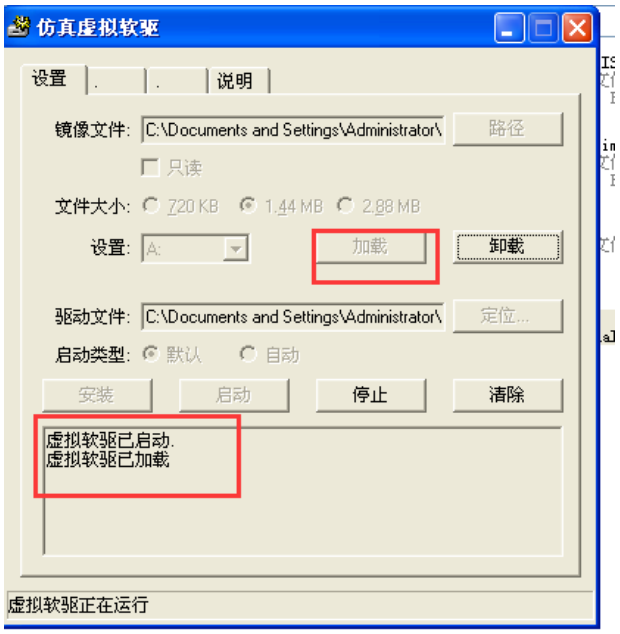
1. 在 dosbox 进行编译链接：

```
C:\>MASM.EXE PROJECT2.ASM
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Object filename [PROJECT2.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  51390 + 448690 Bytes symbol space free

       0 Warning Errors
       0 Severe  Errors

C:\>LINK.EXE PROJECT2.OBJ

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [PROJECT2.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

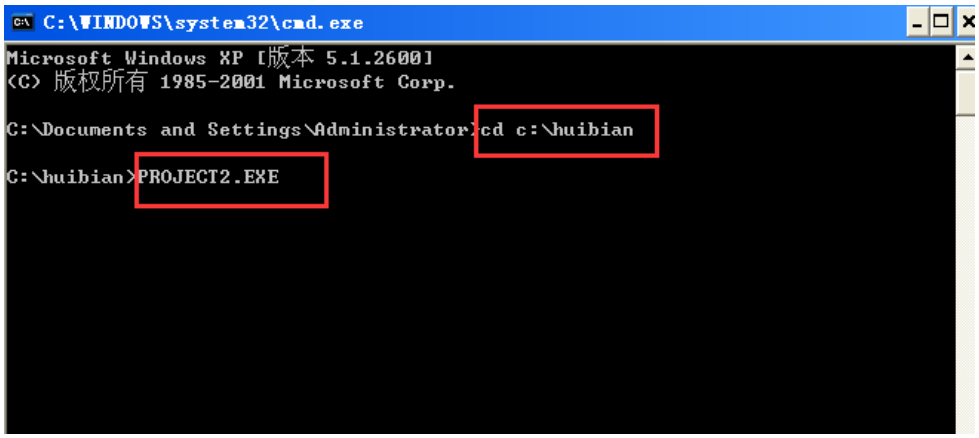
2. 把生成的 exe 文件放到 xp 系统里，打开 vfloppy 文件夹，把 NTFS.img 复制一份。改名为 zbwos:



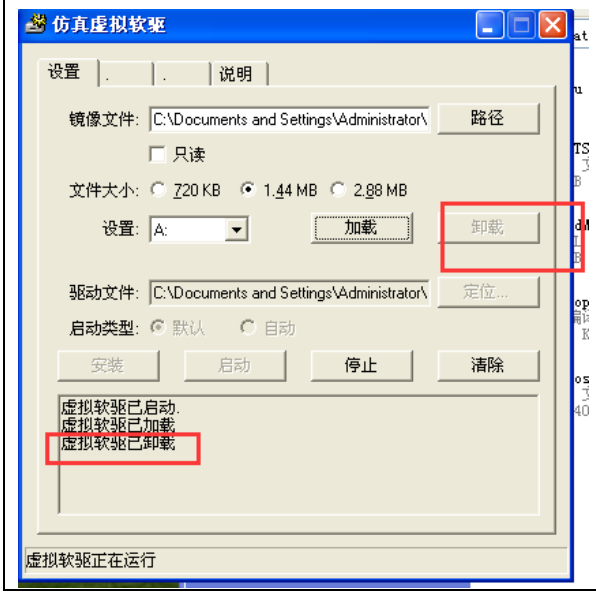
3. 打开 WinVF，选择 zbwos.img，点击加载：



4. 使用命令行切换目录，执行汇编生成的 exe 文件：

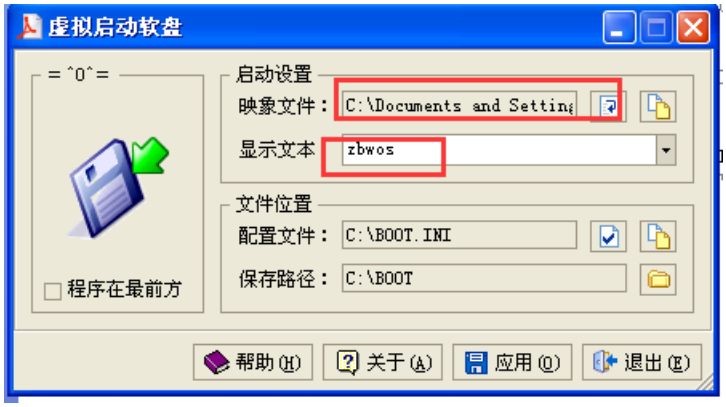


5. 执行之后就可以卸载了：

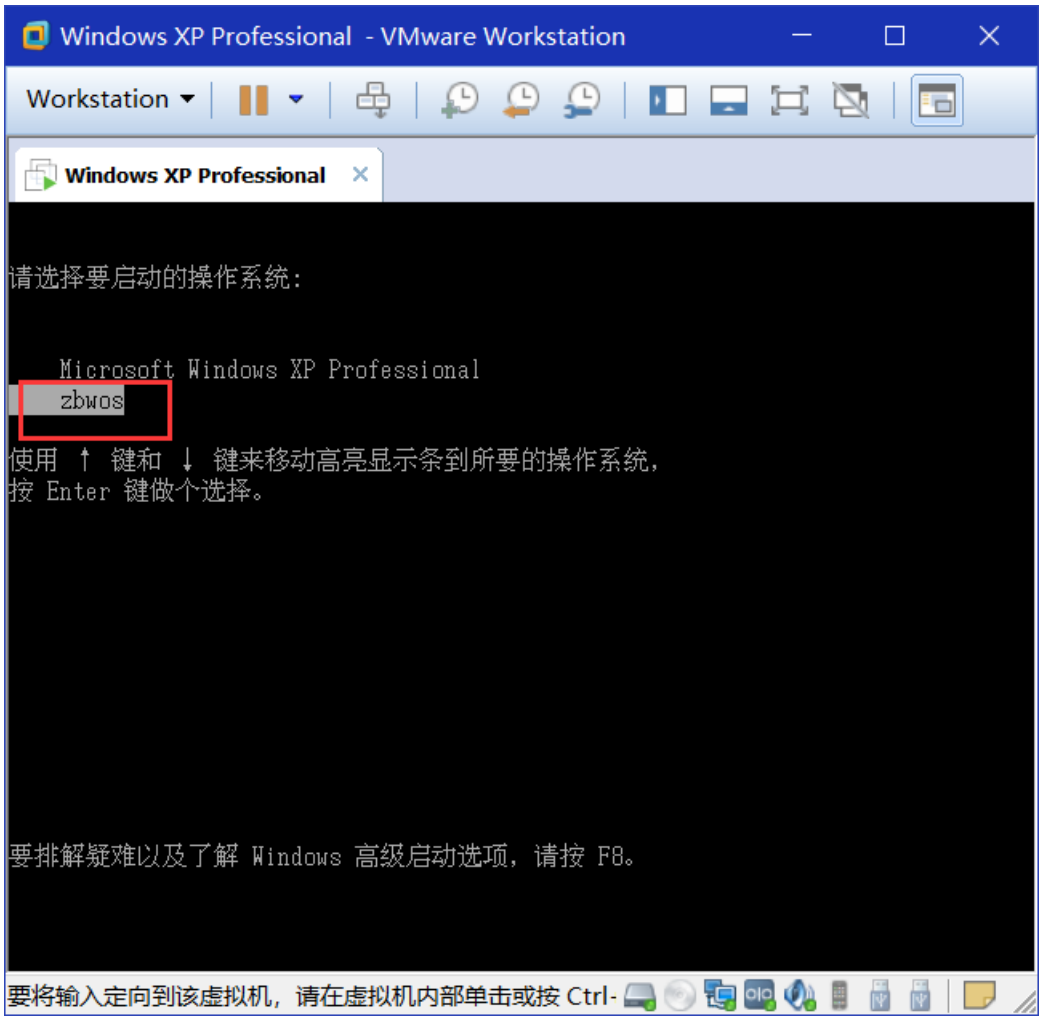


这时候代码已经成功写入了软盘里！

6. 之后打开虚拟启动软盘工具，印象文件选择 zbwos.img，名字改为 zbwos:



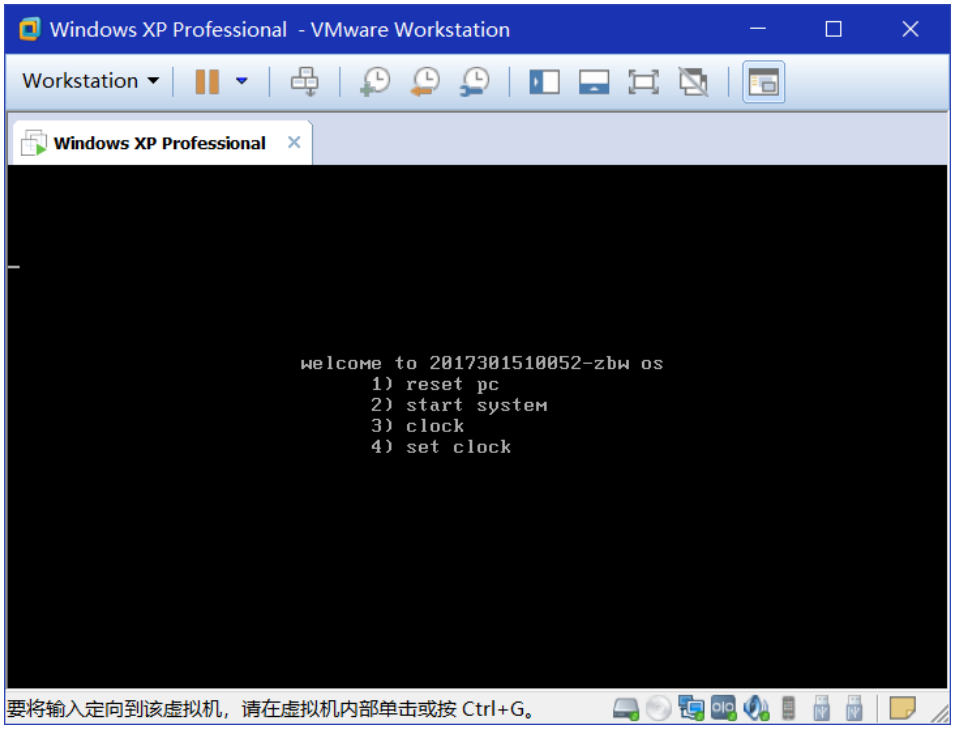
7. 点击应用，随后重启：



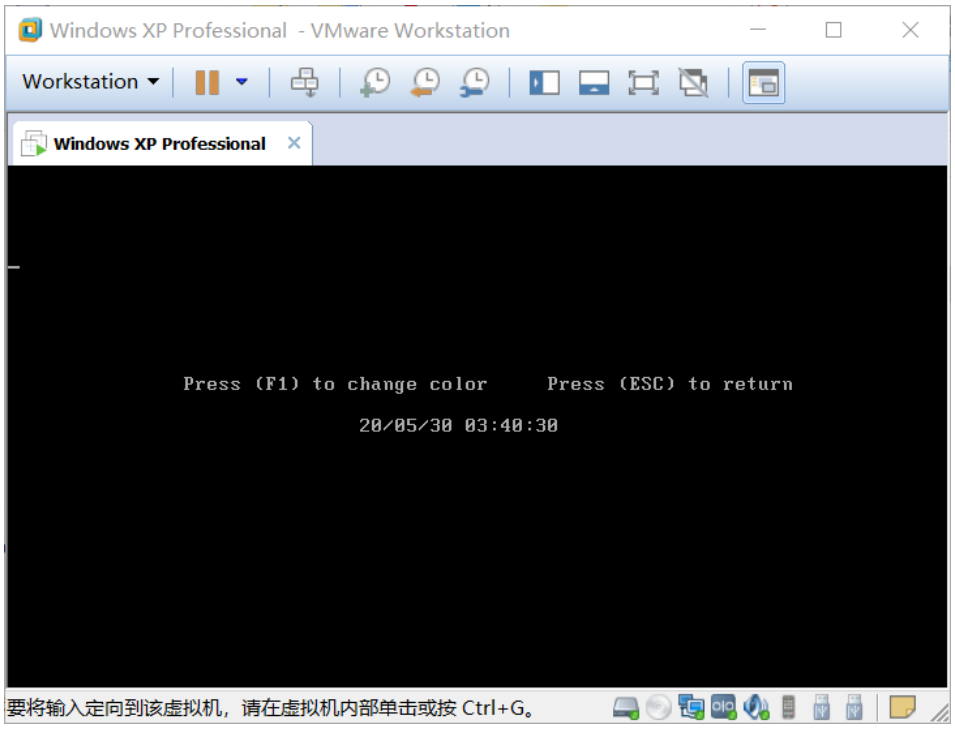
二：进入自己写的计算机系统

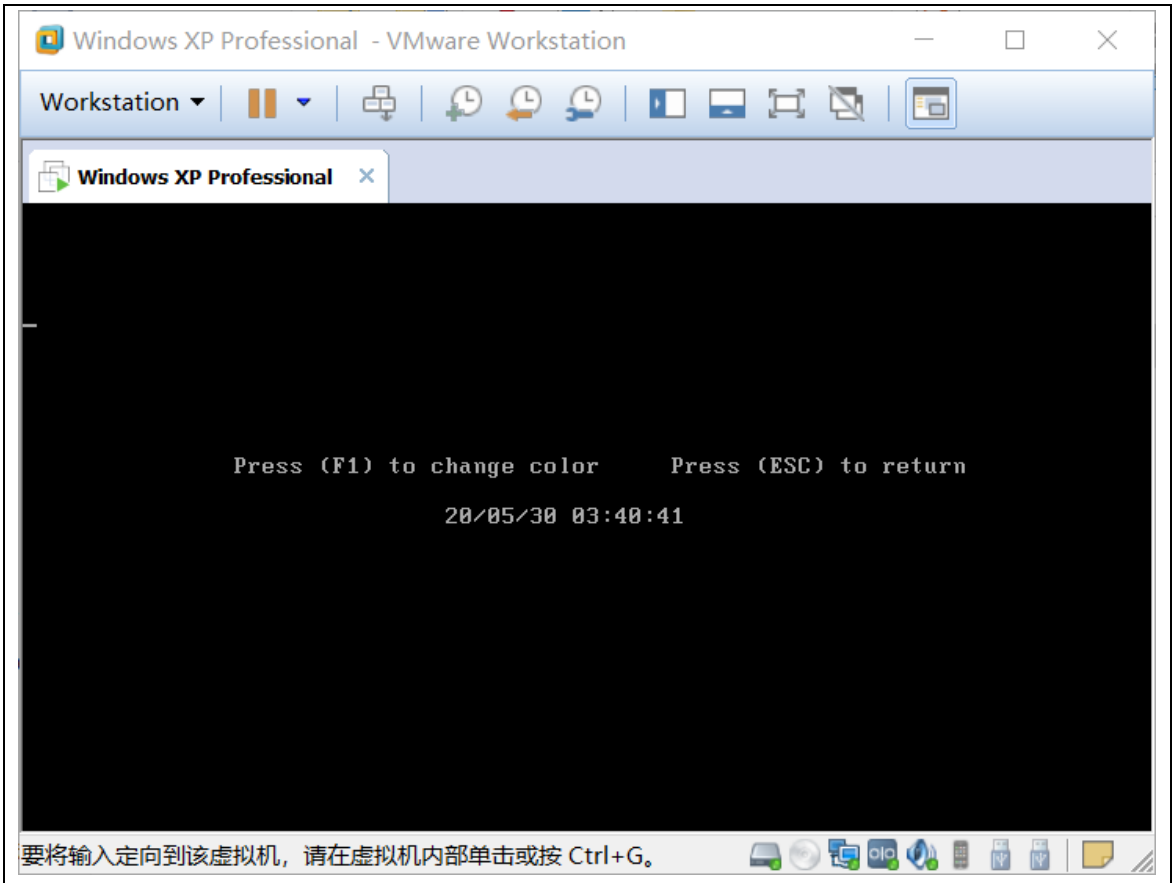
选择 zbwos 进入：

1. 主界面如下：



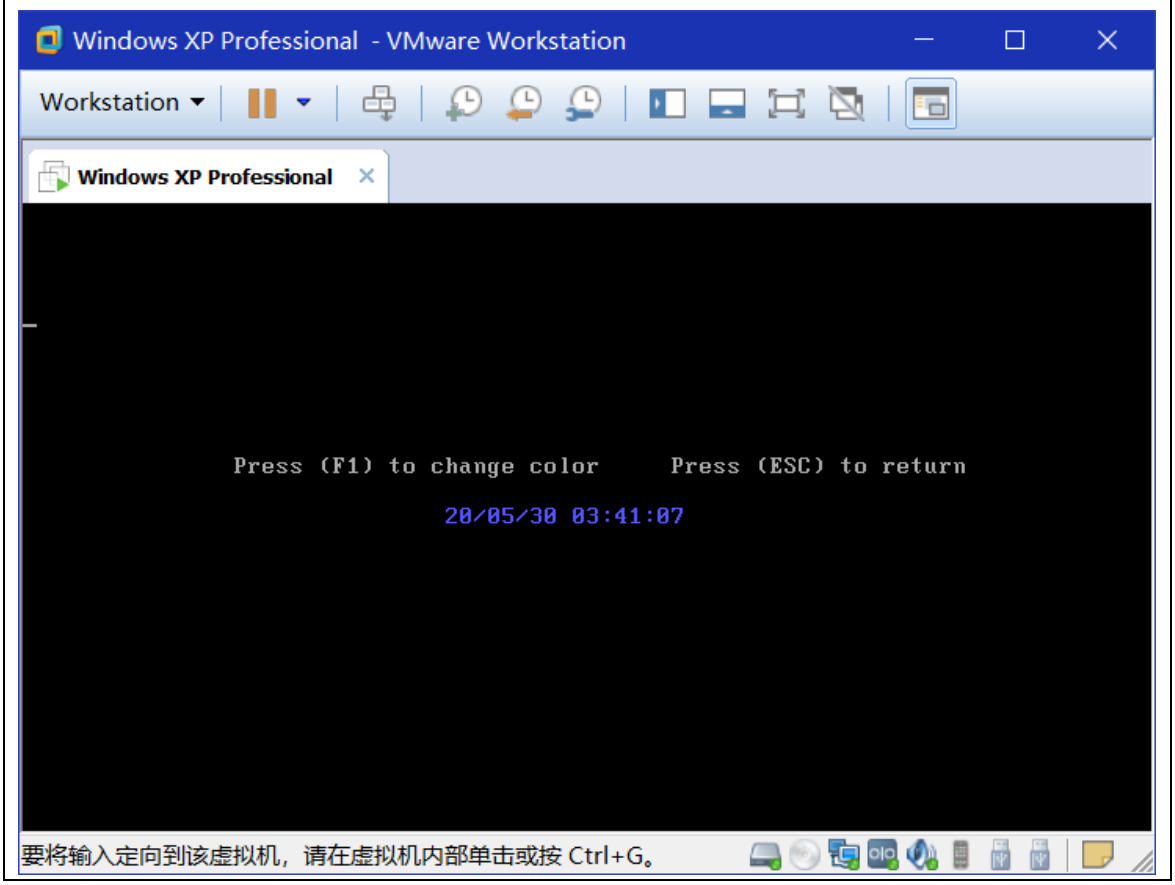
2. 按下 1 会重启；
3. 按下 2 会选择进入原来的 xp 系统；这两个没啥可演示的。
4. 按下 3：

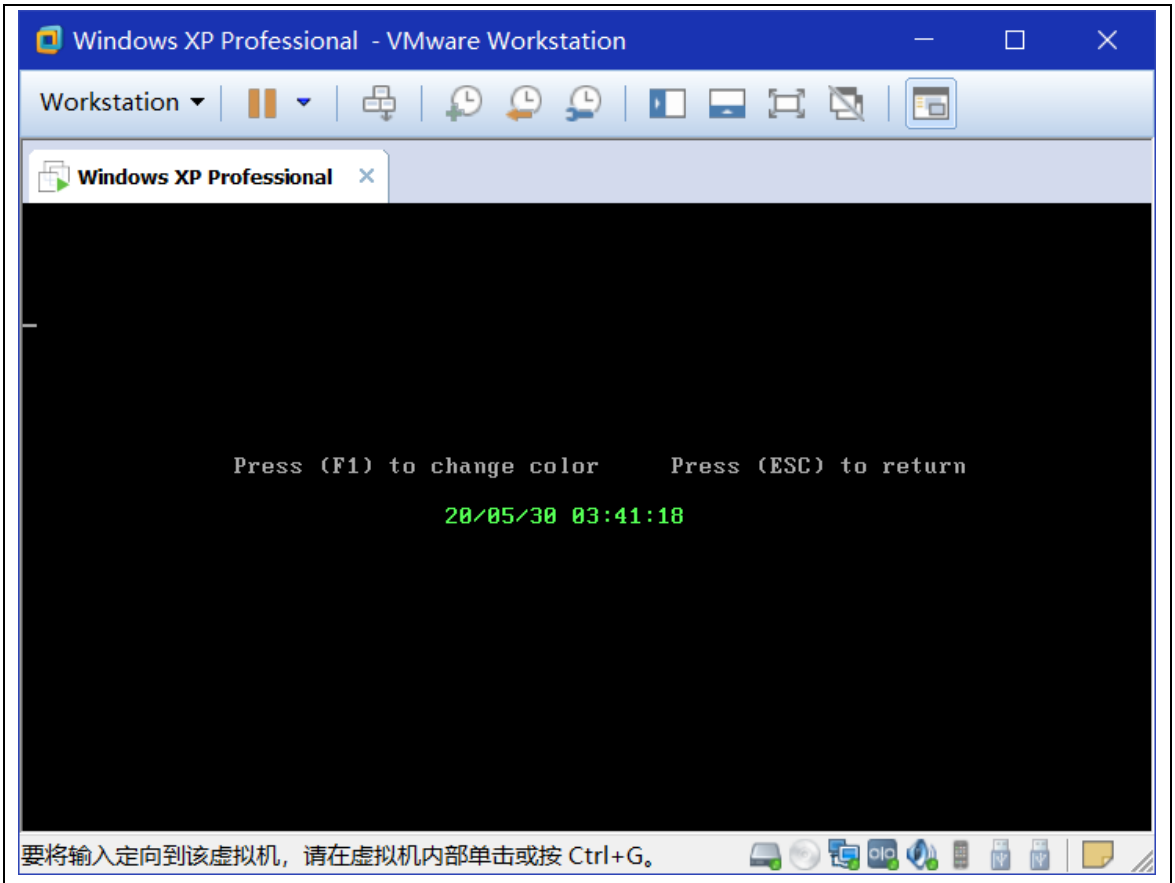




可以看到时间在变化。

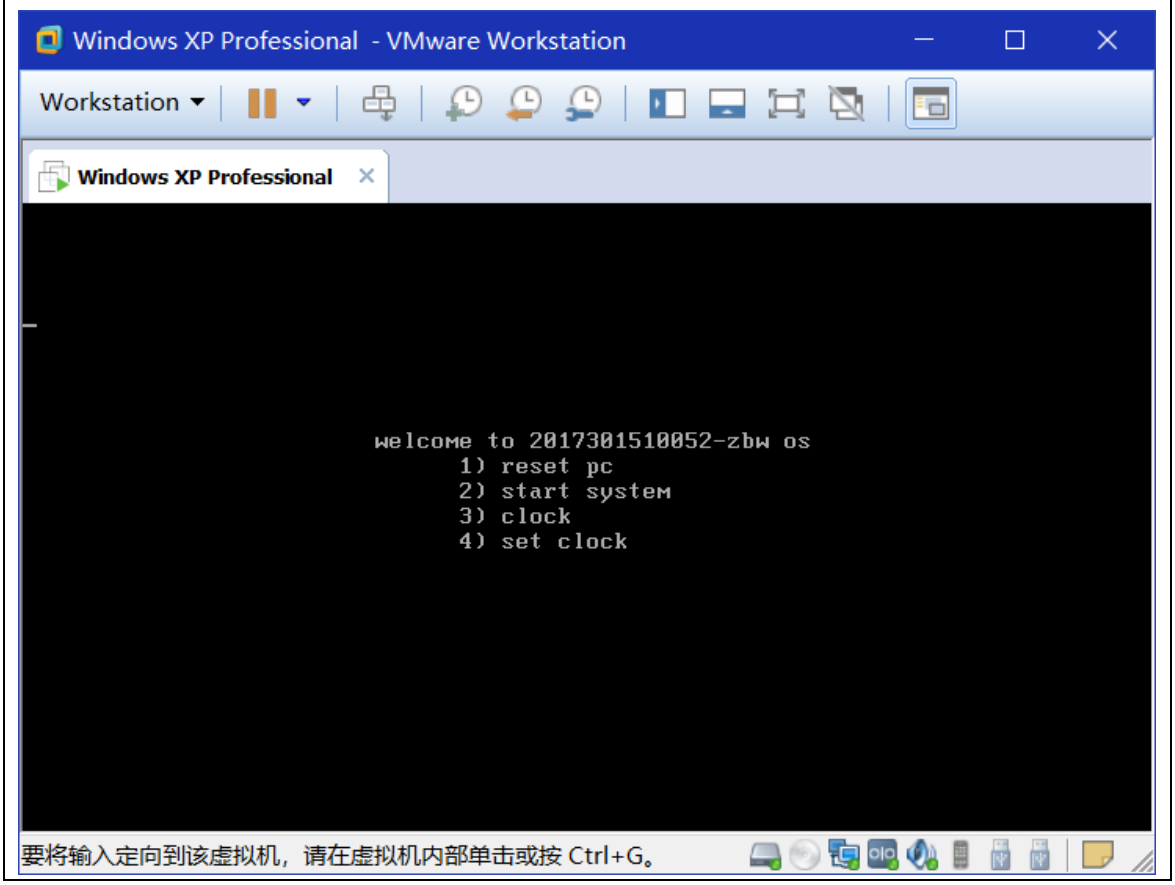
按下 F1:




Windows XP Professional - VMware Workstation
Workstation
Windows XP Professional
Press (F1) to change color Press (ESC) to return
20/05/30 03:41:18
要将输入定向到该虚拟机，请在虚拟机内部单击或按 Ctrl+G。

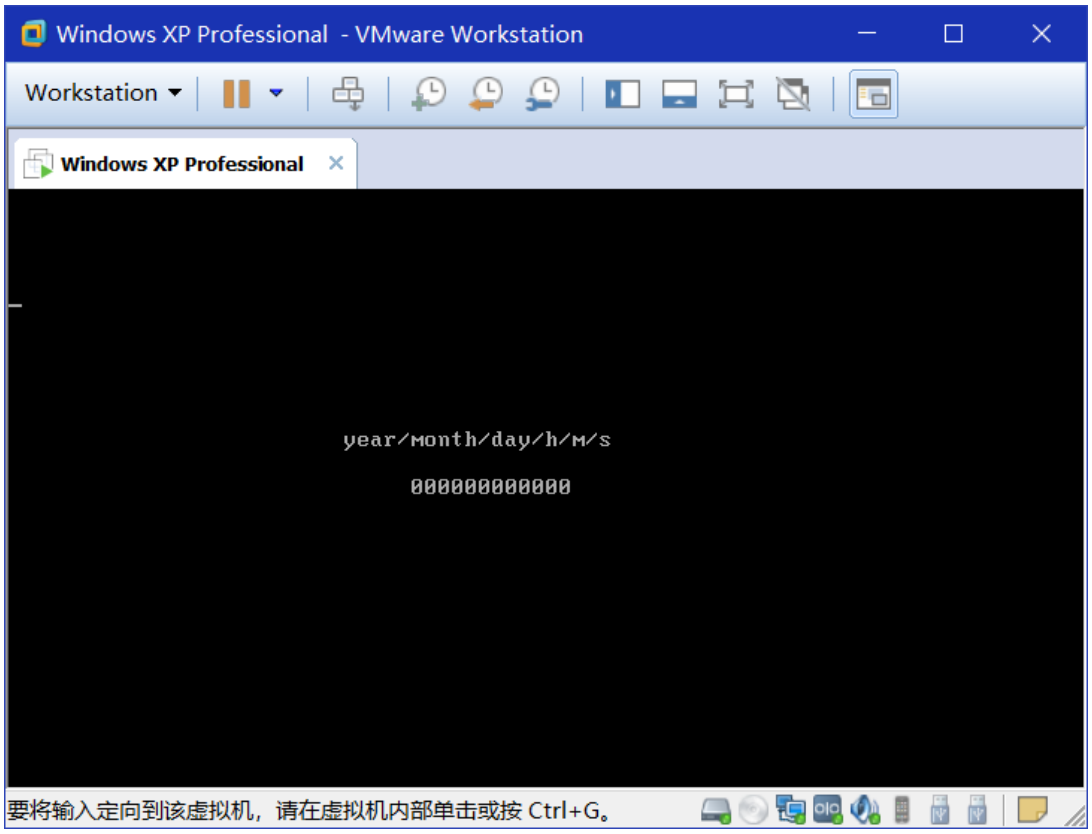
在变颜色。

按下 esc:

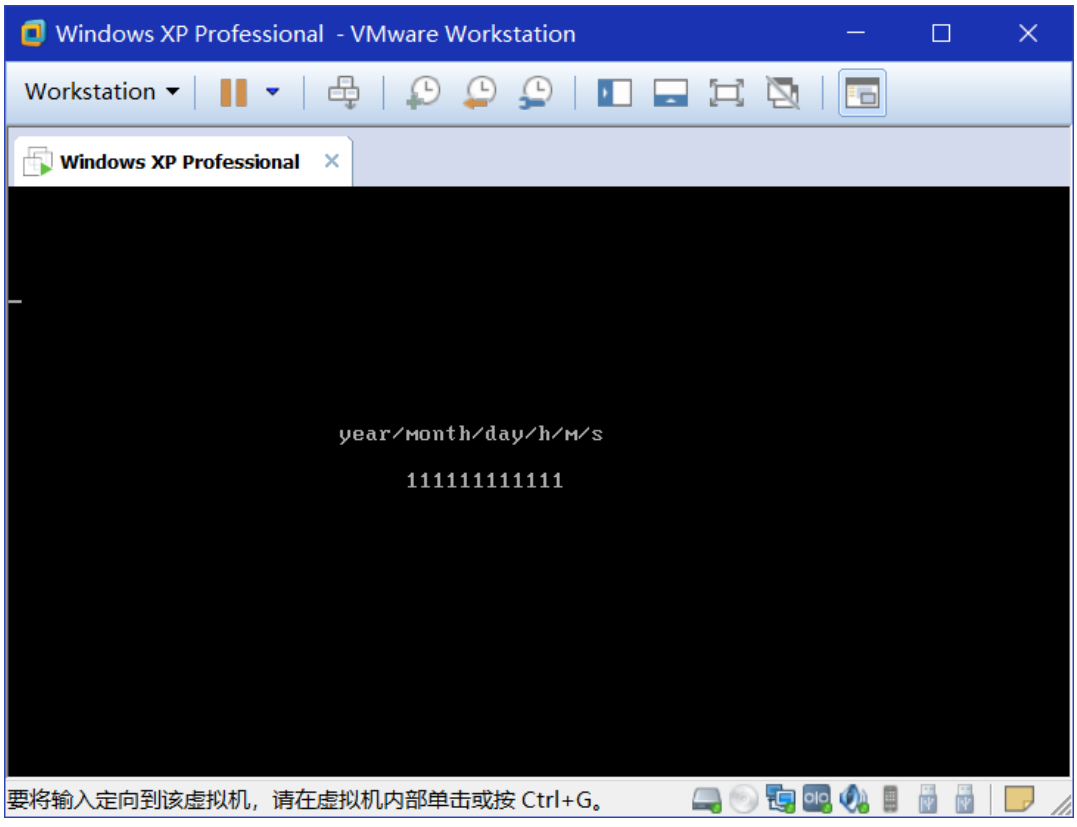

Windows XP Professional - VMware Workstation
Workstation
Windows XP Professional
welcome to 20173015100052-zbw os
1) reset pc
2) start system
3) clock
4) set clock
要将输入定向到该虚拟机，请在虚拟机内部单击或按 Ctrl+G。

又回到了主菜单。

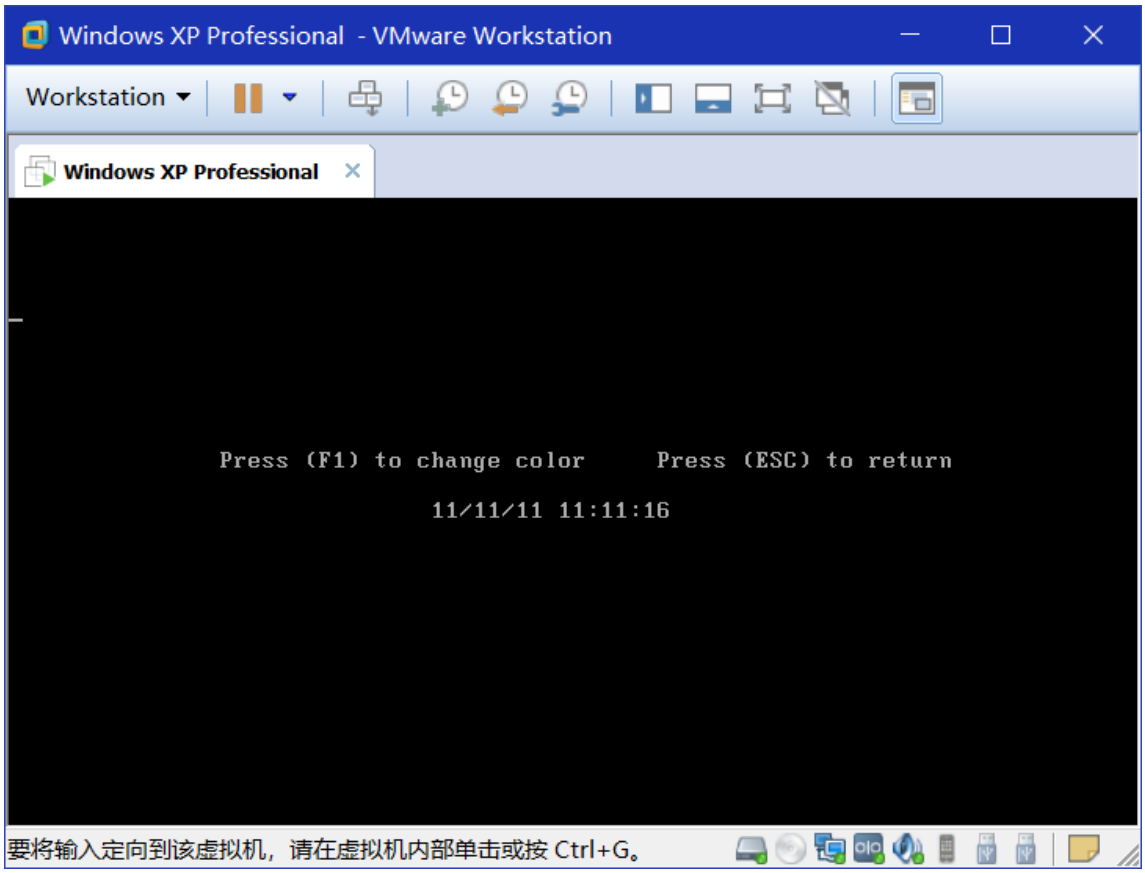
5. 按下 4：



可以设置时间为 11 年 11 月 11 日 11 时 11 分 11 秒：



按下回车后返回到主界面，再次按 3 查看时间：



成功修改时间！

实验完成！

实验总结：

这次的实验工作量有点大，有 400 多行代码，比普通写的 c 程序还要长。不过也是有好处的，就是对汇编更加熟悉了，对寄存器的实验更加灵活了。

其实写完之后发现并不是很难，东西并不算多，大多是课上讲过的或者课下要求自己看的，有很多的子函数，知道入口参数，使用起来很方便。

这次的实验几乎把之前学到的所有东西都用到了，并且能够自己设计一个自行启动的计算机感觉相当不错，而且有了这次的实验基础，以后再做类似的东西就得心应手了。

附录：

完整代码见文件 project2.asm。