操作系统原理实验报告

实验二：加载用户程序的监控程序

数据科学与计算机学院　17大数据与人工智能

17341015　陈鸿峥*

一、 实验目的

- 实现一个最原始的操作系统，即一监控程序，用于调用执行用户程序
- 学会组织并存储监控程序和用户程序

二、 实验要求

- 设计四个（或更多）有输出的用户可执行程序

  设计四个有输出的用户可执行程序，分别在屏幕1/4区域动态输出字符，如展示实验一的内容：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，碰到屏幕相应1/4区域的边后产生反射，改变方向运动，如此类推，不断运动；在此基础上，增加个性扩展，还要在屏幕某个区域特别的方式显示学号姓名等个人信息
- 修改参考原型代码，允许键盘输入，用于指定运行这四个有输出的用户可执行程序之一，要确保系统执行代码不超过512字节，以便放在引导扇区

注意：自行组织映像盘的空间存放四个用户可执行程序

三、 实验环境

具体环境选择原因已在实验一报告中说明。

- Windows 10系统 + Ubuntu 18.04(LTS)子系统
- gcc 7.3.0 + nasm 2.13.02 + gdb
- Oracle VM VirtualBox 5.2.8
- Sublime Text 3

虚拟机配置：内存4M，无硬盘，1.44M虚拟软盘引导。

四、 实验方案

1. 监控程序

(i) 主引导程序

与实验一相同，虚拟软盘的第一个扇区用于存储主引导程序（即监控程序），需要保证最后两个字节为`55aa`。当验证主引导程序有效后，将会跳转到`0x0000:0x7c00h`开始执行。

*本实验报告用LaTeX撰写，创建时间：2019年3月11日

(ii) 调用并执行用户程序

调用用户程序是实验二的核心内容，采用BIOS进行中断。

BIOS中已经含有磁盘读写的调用，即中断号13H和功能号02H的中断。首先设置功能码(ah=2)，要调用的程序占用的扇区数量(al=1)，读取方式(软盘，dl=0)，磁头号(dh=0)，柱面号(ch=0)，扇区开始编号(cl=[sectorNum][1])，读入数据在内存中的存储地址(`[es:bx]`)。

设置好上面的变量之后，调用13H中断(`int 13H`)，则此时磁盘中一个扇区的程序已经被读入内存。接着跳转到用户程序执行。

注意这里需要先设置用户程序的偏移量(`OffSetOfUserPrg=0A100h`)，将`OffSetOfUserPrg`赋值给`bx`，否则数据段位置不正确，用户程序将无法正常执行。

(iii) 字符串显示

这里调用了自己写的`showstring`函数，里面同样调用了BIOS中断(10号)，可以直接将整个字符串在显存中显示出来。详情见程序。

2. 用户程序

一共4个用户程序，分别为`prg1.com`到`prg4.com`。

(i) 程序功能

这里复用了实验一的程序，详情请见实验一的实验报告。但是每个程序会有一定的区别，具体如下：

- 程序一：最简单的单字符反弹无变色显示
- 程序二：两个字符的V形复杂变色显示
- 程序三：连续两个字符('OS')不闪烁的变色显示
- 程序四：两个字符的平行四边形复杂变色显示

同时，每个程序只占用1/4的显存区域，这可以从每个程序最前面的常量(`minX`、`minY`、`maxX`、`maxY`)设置更改。

(ii) 存储组织

主引导程序存储在虚拟软盘的第一个扇区，第一个用户程序存储在第二个扇区，第二个用户程序存储在第三个扇区，以此类推。

注意在每个用户程序开头都应该加上`org 0A100h`，与前面的`OffSetOfUserPrg`相同，表明用户程序读入内存应从内存物理地址`0A100h`开始。

[1]这里采用了一个变量sectorNum，因有多个程序。

(iii) 返回监控程序

DOS环境下中断向量表存放在物理地址0x0000到0x03ff，大小为1KB(1024B)。每个中断在中断向量表中占4B，高位为中断处理程序的段地址(segment)，低位为中断处理程序的偏移量(offset)。且每个中断对应着一个中断标号(int)，如除以0中断为0x0000-0x0003，用int 0调用。因此计算标号为i的中断入口地址可以通过$i \times 4$得到。

通过wiki得知，中断标号0x00到0x13已经具有功能，0x14到0x1f为保留标号，0x20到0xff提供给用户自定义中断。因此，我定义了0x20h号中断（用上面计算地址的方法将返回函数地址放入中断向量表），用于从用户程序返回监控程序。

做法是用int 16中断监测用户是否有键入Ctrl+C（扫描码2e03h），如果有则返回监控程序。

并且，在每个用户程序的show循环中要加入软件中断int 20h，用于监测是否有键盘返回，如果有则清空显存并返回监控程序，否则则继续执行该用户程序。

3. 镜像文件写入

由于一共有5个程序需要编译并写入，如果每次都要重新输入命令则非常麻烦。

因此在这里本人自己写了一个Linux环境下的批处理程序work.sh，用于虚拟软盘的创建、汇编程序的编译与写入，如下所示。

```
#!/bin/bash

rm mydisk.img
/sbin/mkfs.msdos -C mydisk.img 1440
nasm os.asm -o os.com
dd if=os.com of=mydisk.img conv=notrunc
nasm prg1.asm -o prg1.com
dd if=prg1.com of=mydisk.img seek=1 conv=notrunc
nasm prg2.asm -o prg2.com
dd if=prg2.com of=mydisk.img seek=2 conv=notrunc
nasm prg3.asm -o prg3.com
dd if=prg3.com of=mydisk.img seek=3 conv=notrunc
nasm prg4.asm -o prg4.com
dd if=prg4.com of=mydisk.img seek=4 conv=notrunc
```

五、 实验结果

调用work.sh生成虚拟软盘mydisk.img后，将其加载入虚拟机运行。

首先进入的是监控程序，如图1所示。输入数字1 ~ 4，可以选择不同的用户程序。各用户程序的功能已在前面说明，结果如图2到图5所示。

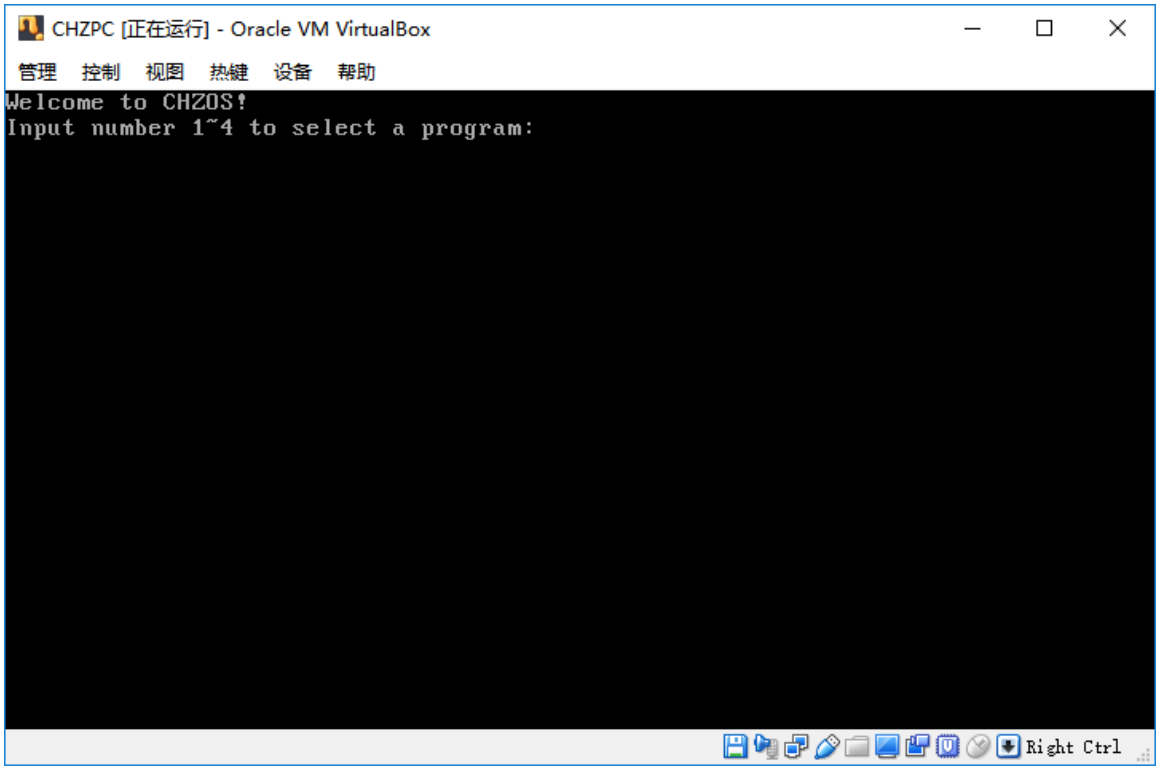

图 1: 监控程序入口

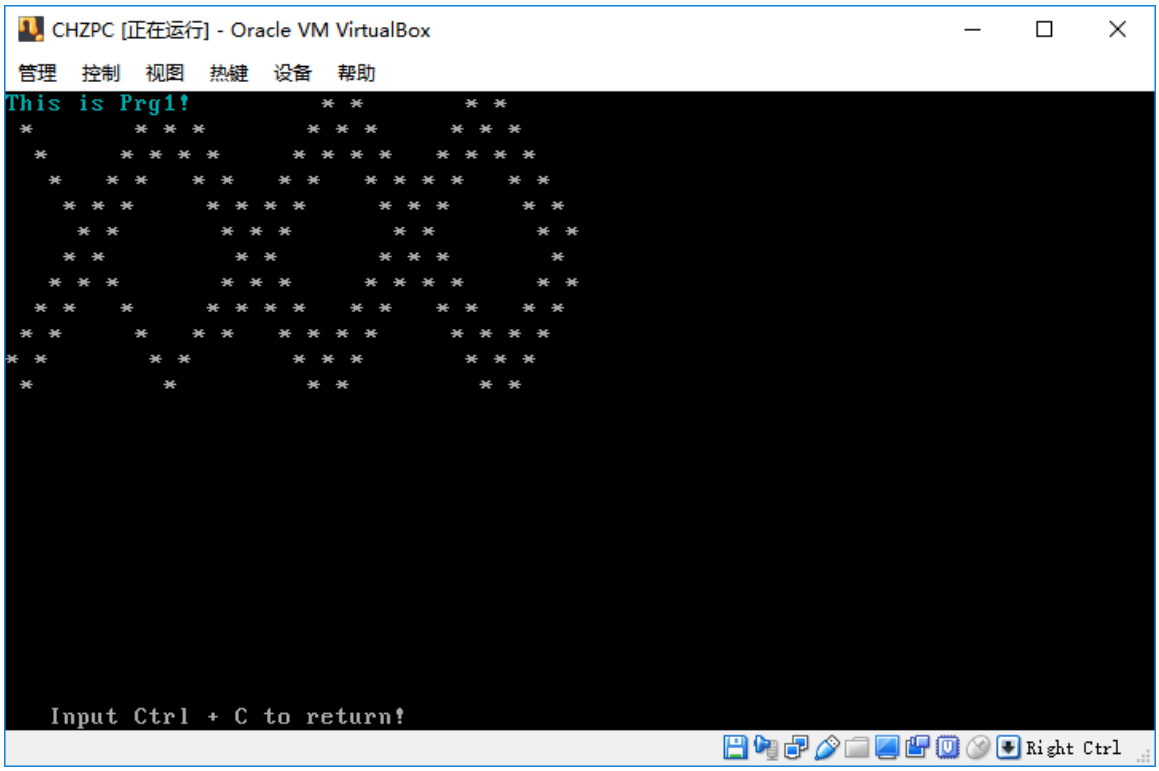

图 2: 用户程序1

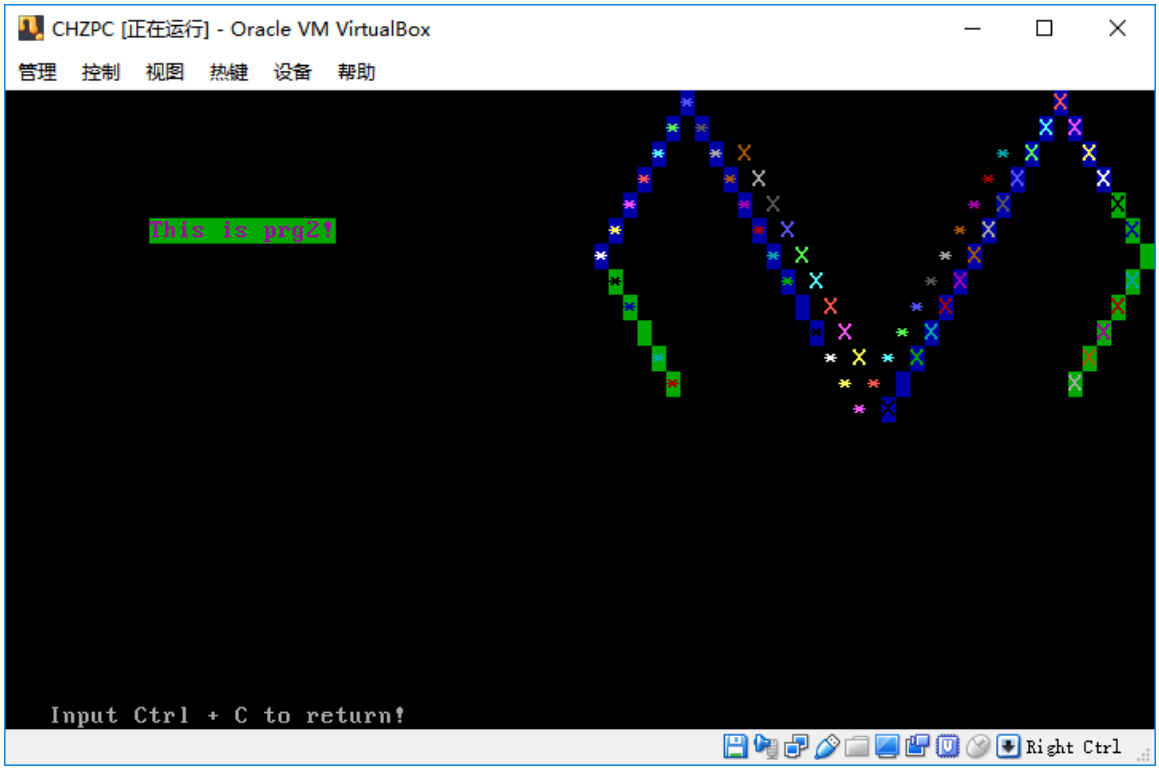

图 3: 用户程序2

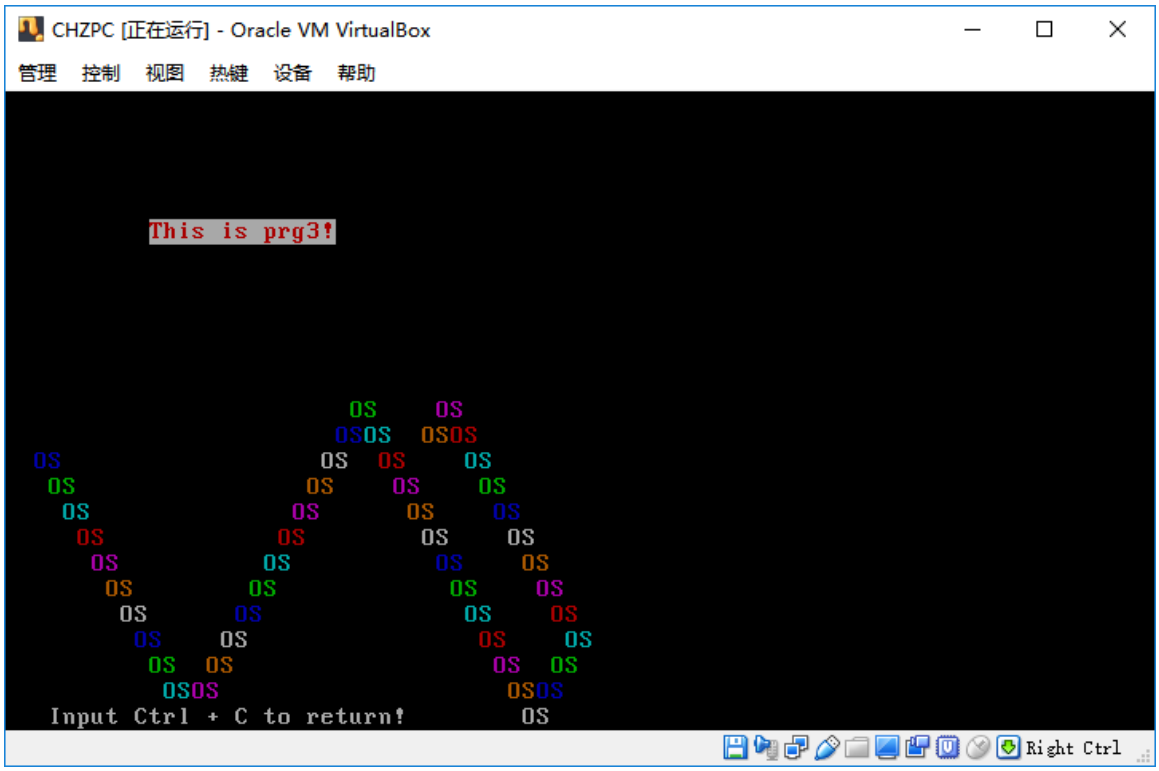

图 4: 用户程序3

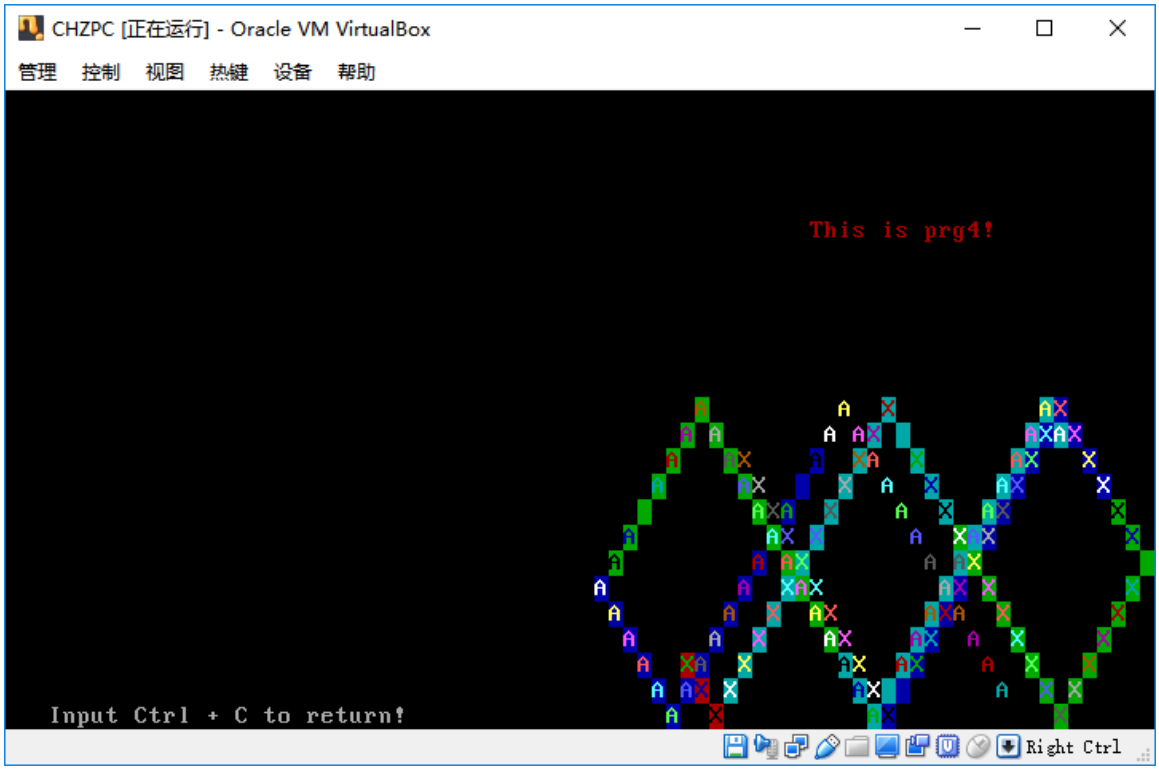

图 5: 用户程序4

六、 实验总结

第二次实验有了第一次实验的基础，会相对好做很多。

这次牢记上次实验的bug，注意到一定要在主引导程序中添加`org 7c00h`指令，以告诉汇编器加载入指令和数据到内存的位置。由于虚拟机会将虚拟软盘第0面第0道第1扇区的512B的程序（主引导程序）加载到`7c00h`处，如果不添加`org`指令，则主引导程序指令的段内偏移错误，程序将无法正常运行。虽然记得在监控程序中添加`org`，但本次实验一开始还是在用户程序中忘记添加`org 0A100h`，导致程序调用无法正常进行，说到底还是对`org`的理解不够透彻。既然在监控程序中告诉BIOS将用户程序需加载到内存`0A100h`的位置，即段内偏移起始地址变化了，那么在用户程序中也需要声明这一点，因此才要在每个用户程序内添加`org 0A100h`。

这次一共编写了5个汇编程序，更加了解到x86 CPU寄存器的分配及使用应该怎么办，特别是写中断的时候，常常会怀疑寄存器够不够用，这一点就要合理利用寄存器，把长的寄存器拆分成几个小的寄存器，分开用。如把32位的`eax`，拆分为16位的`ax`；16位的`ax`又可拆分为高8位的`ah`和低8位的`al`。从这一点也可以看出，当今的编译器的功能多么强大，竟然可以将我们那么复杂的高级语言程序中的变量，合理地映射到各个寄存器上进行计算。

然后则是，写Linux批处理文件`.sh`时出现的问题。由于我用的是子系统，编写代码程序时依然在Windows环境下进行，因此没有注意到Windows环境和Linux环境下文本文档的区别。

Windows环境默认换行采用两个字符，即`\r\n`，而Linux环境则是单个字符`\n`。对于批处理程序来说，对字符相当敏感，因而在执行过程中会报错，无法识别好在Sublime Text 3本来就可以设置编辑的选项，点击Preferences-Settings，在弹出的`Preferences.sublime-settings`文件中，添加`"default_line_ending": "unix"`即可实现在Windows环境下编写代码但采用的是Linux环境的换行符。

为了增强程序的鲁棒性，在本次实验中我也添加了一些异常处理机制，如在读取用户输入过程中，判断用户输入是否有效，如果无效则另用户重新输入，有效才开始执行下一步操作。

总的来说，本次实验自己实现了一个最为简单的操作系统，实现调取执行用户程序的功能，更加深刻明白了`org`的含义，以及x86汇编的编写。感受颇深，也非常兴奋，希望通过这样子不断地训练，最终真的可以写出一个完整的操作系统出来。

七、 参考资料

1. 李忠，王晓波，余洁，《x86汇编语言-从实模式到保护模式》，电子工业出版社，2013
2. 中断向量表(IVT)，https://wiki.osdev.org/Interrupt_Vector_Table
3. DOS的古董美，http://www.voidcn.com/article/p-tabpatgs-ps.html
4. INT 16 - Keyboard Scan Codes, http://stanislavs.org/helppc/scan_codes.html
5. `iret`, https://www.felixcloutier.com/x86/iret:iretd
6. (x86 Assembly) Changing Interrupt Vector Table, http://devdocs.inightmare.org/tutorials/x86-assembly-changing-interrupt-vector-table.html

附录 A. 程序清单

这里只附上主引导程序的代码，其他用户程序请见附件。

```
; Constants
OffSetOfUserPrg equ 0A100h

;;;;; Initialization ;;;;;
    org  7c00h

;;;;; Initializeion ;;;;;
%macro showstring 4
    ; msg, msglen, row, col
    mov ax, cs              ; as = cs
    mov ds, ax              ; data segment
    mov bp, %1              ; bp = string index
    mov ax, ds              ; (BIOS) es:bp = string address
    mov es, ax              ; es = ax
    mov cx, %2              ; (BIOS) set string length
    mov ax, 1301h           ; (BIOS) ah = 13h (BIOS: function code) al = 01h (only
```

```
        ↪ char)
    mov bx, 0007h           ; (BIOS) page/bh = 00h property/bl = 07h
    mov dh, %3              ; (BIOS) row
    mov dl, %4              ; (BIOS) col
    int 10h                 ; (BIOS) 10h: show one string
%endmacro

%macro writeIVT 2           ; write interrupt vector table (IVT)
    ; num, function address
    mov ax, 0000h           ; physical address
    mov es, ax
    mov ax, %1
    mov bx, 4
    mul bx                  ; calculate the IVT address (ax*4)
    mov si, ax
    mov ax, %2
    mov [es:si], ax         ; write segment
    add si, 2
    mov ax, cs
    mov [es:si], ax         ; write offset
%endmacro

writeIVT 20h, INT20H

begin:
    showstring msg, msglen, 0, 0
    showstring msg2, msglen2, 1, 0

input:
    mov ah, 0               ; (BIOS) function code
    int 16h                 ; (BIOS) read keyboard
    sub al, '0'             ; (BIOS) return al = ASCII
    cmp al, 1
    jl input                ; not from 1~4
    cmp al, 4
    jg input                ; not from 1~4
    inc al                  ; start from 2
    mov [sectorNum], al     ; put it into memory

;;;;; Load Program ;;;;;
load:                       ; load [es:bx]
    call clear              ; clear the screen
    mov ax, cs              ; segment address (store data)
    mov es, ax              ; set segment
    mov bx, OffSetOfUserPrg ; user program address
```

```
    mov ah, 2               ; (BIOS) function code
    mov al, 1               ; (BIOS) # of sector that program used
    mov dl, 0               ; (BIOS) driver: floppy disk (0)
    mov dh, 0               ; (BIOS) magnetic head
    mov ch, 0               ; (BIOS) cylinder
    mov cl, [sectorNum]     ; (BIOS) start sector
    int 13H                 ; (BIOS) 13h: read disk
    jmp OffSetOfUserPrg     ; a.com has been loaded into memory

clear:
    mov ax, 0B800h          ; video memory
    mov es, ax
    mov si, 0
    mov cx, 80*25
    mov dx, 0
    clearloop:
        mov [es:si], dx
        add si, 2
    loop clearloop
    ret

INT20H:
    showstring msg3, msglen3, 24, 3
    mov ah, 01h
    int 16h
    jz noclick
    mov ah, 00h             ; click the button
    int 16h
    cmp ax, 2e03h           ; click Ctrl + C
    jne noclick
    call clear              ; clear the screen
    jmp begin               ; return to monitor program
    noclick:
    iret                    ; interrupt return

end:
    jmp $

;;;;; Data Segment ;;;;;
datadef:
    msg db 'Welcome to CHZOS!'
    msglen equ ($-msg)

    msg2 db 'Input number 1~4 to select a program: '
    msglen2 equ ($-msg2)
```

```
msg3 db 'Input Ctrl + C to return!'
msglen3 equ ($-msg3)

sectorNum db '1'
```

附录 B. 附件文件说明

序号	文件	描述
1	os.asm	主引导程序
2	os.com	主引导程序编译出来的.com执行文件
3-6	prgX.asm	四个用户程序
7-10	prgX.com	四个用户程序编译出来的.com执行文件
11	work.sh	Linux环境批处理代码
12	mydisk.img	我的OS虚拟软盘（里面已包含程序）