

具有加载COM格式用户程序的监控程序（纯汇编）

具有加载COM格式用户程序的监控程序（纯汇编）

个人信息

实验题目

实验目的

实验要求

实验内容

实验方案

所用工具

虚拟机配置

实验原理

代码关键部分

监控程序

伪指令

初始化寄存器

打印字符串

加载盘块

用户程序

伪指令org

返回监控程序

设计表格

实验过程

DosBox

安装

使用

运行结果

实验总结

参考资料

个人信息

- 院系：数据科学与计算机学院
- 年纪：2018
- 姓名：王天龙
- 学号：18340168

实验题目

具有加载COM格式用户程序的监控程序（纯汇编）

实验目的

1. 了解监控程序执行用户程序的主要工作
2. 了解一种用户程序的格式与运行要求
3. 加深对监控程序概念的理解
4. 掌握加载用户程序方法
5. 掌握几个BIOS调用和简单的磁盘空间管理

实验要求

1. 知道引导扇区程序实现用户程序加载的意义
2. 掌握COM/BIN等一种可执行的用户程序格式与运行要求
3. 将自己实验一的引导扇区程序修改为3-4个不同版本的COM格式程序，每个程序缩小显示区域，在屏幕特定区域显示，用以测试监控程序，在1.44MB软驱映像中存储这些程序。
4. 重写1.44MB软驱引导程序，利用BIOS调用，实现一个能执行COM格式用户程序的监控程序。
5. 设计一种简单命令，实现用命令交互执行在1.44MB软驱映像中存储几个用户程序。
6. 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

实验内容

1. 将自己实验一的引导扇区程序修改为一个的COM格式程序，程序缩小显示区域，在屏幕第一个1/4区域显示，显示一些信息后，程序会结束退出，可以在DOS中运行。在1.44MB软驱映像中制定一个或多个扇区，存储这个用户程序a。相似地、将自己实验一的引导扇区程序修改为第二、第三、第四个的COM格式程序，程序缩小显示区域，在屏幕第二、第三、第四个1/4区域显示，在1.44MB软驱映像中制定一个或多个扇区，存储用户程序b、用户程序c、用户程序d。
2. 重写1.44MB软驱引导程序，利用BIOS调用，实现一个能执行COM格式用户程序的监控程序。程序可以按操作选择，执行一个或几个用户程序。解决加载用户程序和返回监控程序的问题，执行完一个用户程序后，可以执行下一个。
3. 设计一种命令，可以在一个命令中指定某种顺序执行若干个用户程序。可以反复接受命令。
4. 在映像盘上，设计一个表格，记录盘上有几个用户程序，放在那个位置等等信息，如果可以，让监控程序显示出表格信息。
5. 拓展自己的软件项目管理目录，管理实验项目相关文档








实验方案

所用工具

实验平台是windows10系统，使用的虚拟机软件是vmware workstation pro，使用的汇编器是nasm，使用的可视化编译十六进制文件内容工具是winhex，使用visual studio 2019编写汇编程序代码，.com文件调试工具为DosBox。

虚拟机配置

虚拟机为1cpu，内存为4MB，使用一个1.44MB大小的软盘，选择从软盘启动

设备	摘要
 内存	4 MB
 处理器	1
 CD/DVD (IDE)	自动检测
 软盘	正在使用文件 D:\computer_op...
 网络适配器	NAT
 声卡	自动检测
 显示器	自动检测

实验原理

实验的大致流程是先利用vmware生成一个1.44MB大小的虚拟软盘，把软盘划分成2880个扇区，每个扇区大小为512个字节。然后利用nasm把监控程序转换成.bin文件，然后利用winHex写到软盘的首扇区（编号为1），即把监控程序当作引导程序进入内存，接管机器。然后把四个用户程序分别写到3、4、5、6号扇区，2号扇区是一个表格，用于记录盘上的用户程序的位置。虚拟机启动后，监控程序接管

了虚拟机，然后根据用户的输入，把用户程序从相应的盘块加载到内存的0x8100处，然后跳转到用户程序执行。用户程序执行完成后跳回到监控程序入口0x7c00。

代码关键部分

监控程序

监控程序实现的功能有，提供输入给用户选择要执行的用户程序，并把用户程序转载到内存的0x8100处。监控程序的设计和实验一的引导程序差不多，主要多了一个把程序从外存加载内存里。

伪指令

首先是使用伪指令org指明地址标号的偏移量，这里要注意的是，虽然监控程序实际上也是一个.com文件，但是我们把它当作监控程序来运行，被加载进来时，cs=0x00，ip=0x7c00。所以为了获取正确的便宜，我们应该写的是org 0x7c00，而不是org 0x100

```
org 0x7c00; 监控程序被加载到cs:ip=0x0:0x7c00处
```

初始化寄存器

然后是初始化寄存器，要注意到不能直接使用cs的值赋给ds、ss

```
mov ax,cs
mov ds,ax; 不能直接mov ds,cs
mov ss,ax
```

打印字符串

打印一串字符串，提示用户输入，这里直接调用BIOS的0x10号功能打印，然后从键盘读取一个字符，相当于一个简单的交互指令，根据字符加载相对应的盘块

```
Start:
    mov bp, Message      ; BP=当前串的偏移地址
    mov ax, ds            ; ES:BP = 串地址
    mov es, ax            ; 置ES=DS
    mov cx, MessageLength ; CX = 串长(=9)
    mov ax, 1301h          ; AH = 13h (功能号)、AL = 01h (光标置于串尾)
    mov bx, 0007h          ; 页号为0(BH = 0) 黑底白字(BL = 07h)
    mov dh, 0              ; 行号=0
    mov dl, 0              ; 列号=0
    int 10h                ; BIOS的10h功能: 显示一行字符
switch:
    ; 读取一个字符
    mov ah, 0x00
    int 0x16
    ; 判断跳转到相应的位置
    cmp al, '1'
    jz LoadUp_Lt
    cmp al, '2'
    jz LoadUp_Rt
    cmp al, '3'
    jz LoadDn_Lt
    cmp al, '4'
    jz LoadDn_Rt
    jmp switch; 如果没有匹配，则重新读
```

其中Message、MessageLength定义如下

```
Message:
    db 'select user program(1-4):'
    MessageLength equ ($-Message)
    times 510-($-$$) db 0
    db 0x55,0xaa
```

加载盘块

然后就是加载一个盘块到内存的0x8100处，并且跳转到哪里，以加载左上角的碰撞程序为例

```
LoadUp_Lt:
    mov cl,3                ;起始扇区号 ; 起始编号为1
    jmp BeginLoad

BeginLoad:
    mov ax,cs               ;段地址 ; 存放数据的内存基地址
    mov es,ax               ;设置段地址（不能直接mov es,段地址）
    mov bx, OffsetOfUserPrg ;偏移地址；存放数据的内存偏移地址0x8100
    mov al,1                ;扇区数
    mov dl,0                ;驱动器号 ; 软盘为0，硬盘和U盘为80H
    mov dh,0                ;磁头号 ; 起始编号为0
    mov ch,0                ;柱面号 ; 起始编号为0
    mov ah,2                ;功能号
    int 13H ;               调用读磁盘BIOS的13h功能，读软盘或硬盘上的若干物理扇区到内存的
    ES:BX处
    jmp 0x800:0x100         ;同时修改了cs和ip的值，不用jmp 0x8100
```

这里要注意一个很容易出错的地方，就是最后的跳转。虽然jmp 0x800:0x100和jmp 0x8100最后跳转到的位置相同，但是它们对cs、ip寄存器的影响不一样。如果用jmp 0x8100，则cs的值不变，ip=0x8100，而用jmp 0x800:0x100的话，cs=0x800，ip=0x100。在这里要用jmp 0x800:0x100。这是因为我们要跳转到一个.com执行程序，它有一个程序头，程序主体一般被加载到段开始偏移0x100个字节的地方，所以.com程序一般都有个伪指令org 0x100。为了跳转到.com程序后，能正确获取偏移量，我们要让跳转后的ip=0x100。

用户程序

这次的用户程序有4个，但他们都是大同小异，功能是在屏幕的左上、左下、右上、右下实现像实验一的字符块碰壁功能。代码也是由实验一修改而来，主要是修改了碰撞的边界，重要算法都不变。这里以在左上角碰壁为例，介绍比实验一修改的部分

伪指令org

org的不同，在实验一中，程序作为引导程序引入，所以用org 0x7c00，而在这里，程序是一个.com执行程序，一般都要用org 0x100

```
org 0x100;实验一为org 0x7c00
```

返回监控程序

程序结束时（打印了100个字符），返回监控程序

```

displaytimes equ 100
mov si,msg
show:
    mov al,[si+5]
    cmp al,displaytimes
    jz return;jz不能进行段间跳转，要用jmp
    ;以下是在屏幕特定位置打印字符
return:
    jmp 0x00:0x7c00
msg:
    db 1;position x si屏幕的第几行
    db 0;position y si+1屏幕的第几列
    db 'A';char si+2
    db Dn_Lt;direction si+3
    db 0;color si+4
    db 0;打印了多少次 si+5
    times 512-($-$$) db 0x00;这里不再是引导程序，不需要0x55 0xAA结束

```

这里返回监控看那个程序需要特别注意。由于监控程序是引导程序，他的偏移是0x7c00，所以我们跳转时，要从0x800:0x100跳到0x00:0x7c00。而jz不能直接进行段间转移，所以我们要中转一下，使用jmp跳转。

用户程序的其他部分都和实验一的相同。

设计表格

在盘号为2的盘块上，有一个表格，表格是32byte一行，记录了程序名和其所在的盘号（从1开始编号），如下图所示

00000512	66 75 6E 63 74 69 6F 6E 3A 20 75 70 5F 6C 74 20 62 6C 6F 63 6B 3A 20 33 00 00 00 00 00 00 00 00	function: up_lt block: 3
00000544	66 75 6E 63 74 69 6F 6E 3A 20 75 70 5F 72 74 20 62 6C 6F 63 6B 3A 20 34 00 00 00 00 00 00 00 00	function: up_rt block: 4
00000576	66 75 6E 63 74 69 6F 6E 3A 20 64 6E 5F 6C 74 20 62 6C 6F 63 6B 3A 20 35 00 00 00 00 00 00 00 00	function: dn_lt block: 5
00000608	66 75 6E 63 74 69 6F 6E 3A 20 64 6E 5F 72 74 20 62 6C 6F 63 6B 3A 20 36 00 00 00 00 00 00 00 00	function: dn_rt block: 6
00000640	00 00	
00000672	00 00	
00000704	00 00	
00000736	00 00	

为了得到这些信息，先编写一个table.asm，然后用nasm转换成二进制码table.bin

```

;table.asm
function1:
    db 'function: up_lt block: 3'
    times 32-($-function1) db 0
function2:
    db 'function: up_rt block: 4'
    times 32-($-function2) db 0
function3:
    db 'function: dn_lt block: 5'
    times 32-($-function3) db 0
function4:
    db 'function: dn_rt block: 6'
    times 512-($-$$) db 0

```

实验过程

DosBox

这次实验只比实验一多了一个DosBox，DosBox是一个Dos模拟器，用它可以运行.com文件，可以帮助我们debug，下面简单介绍他的安装和使用

安装

下载地址为<https://www.dosbox.com/download.php?main=1>

下载解压完成后就可以使用了

DOSBox 0.74 Options	2010/4/10 3:10	WINDOWS 应用程序	1 KB
DOSBox	2010/5/12 17:55	应用程序	3,640 KB
dosbox 0.74	2017/0/22 15:30	CONEX 文件	11 KB

```
DOSBox 0.74, Cpu speed: 3000 cycles, Fra...
Z:\>mount C \Users\36271\Desktop\DOSBox-0.74
Drive C is mounted as local directory \Users\36271\Desktop\DOSBox-0.74\
Z:\>C:
C:\>cd asm
C:\ASM>dir
Directory of C:\ASM\
.                <DIR>                06-05-2020 10:42
..               <DIR>                06-05-2020 11:09
WTL              <DIR>                06-05-2020 10:43
CONFIG1.TXT      10,991 12-09-2017 20:09
DEBUG.EXE        20,634 14-04-2008 12:00
EDIT.COM         72,174 13-05-1998 17:57
EXE2BIN.EXE      3,060 06-04-2001 11:22
LIB.EXE          49,661 02-05-2011 17:26
LINK.EXE         69,133 06-04-2001 11:22
MASM.EXE         103,175 06-04-2001 11:22
MASM.IMG         1,474,560 07-04-2009 22:22
8 File(s)        1,803,378 Bytes.
3 Dir(s)         262,111,744 Bytes free.
C:\ASM>S
```

使用

用DosBox可以运行汇编指令，帮助我们理解，以执行jmp为例

jmp 200

```
C:\ASM>debug
~a
073F:0100 jmp 200
073F:0103
~r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0100  NU UP EI PL NZ NA PO NC
073F:0100 E9FD00 JMP 0200
~t
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0200  NU UP EI PL NZ NA PO NC
073F:0200 0000 ADD [BX+SI],AL DS:0000=CD
~s
```

jmp 74f:200

```
~a 200
073F:0200 jmp 74f:200
073F:0205
~r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0200  NU UP EI PL NZ NA PO NC
073F:0200 EA00024F07 JMP 074F:0200
~t
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=074F IP=0200  NU UP EI PL NZ NA PO NC
074F:0200 0000 ADD [BX+SI],AL DS:0000=CD
~s
```

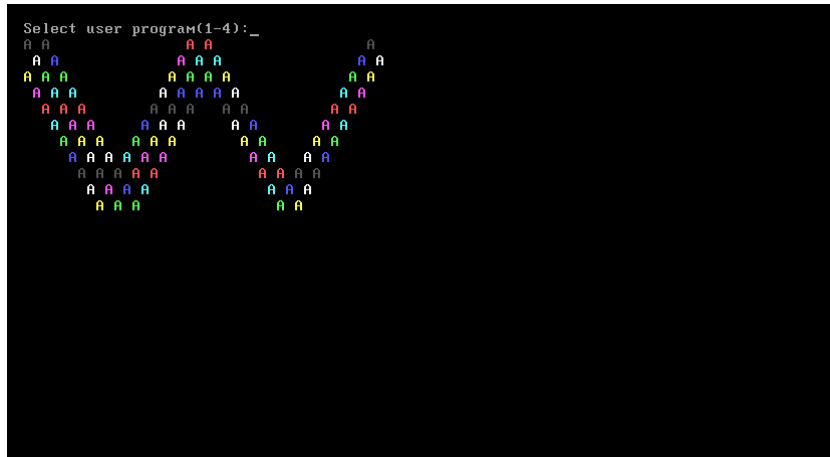
可以看到，jmp 200只改变ip寄存器，jmp 74f:200，cs、ip寄存器都变了，这对我们理解jmp作用很有帮助

运行结果

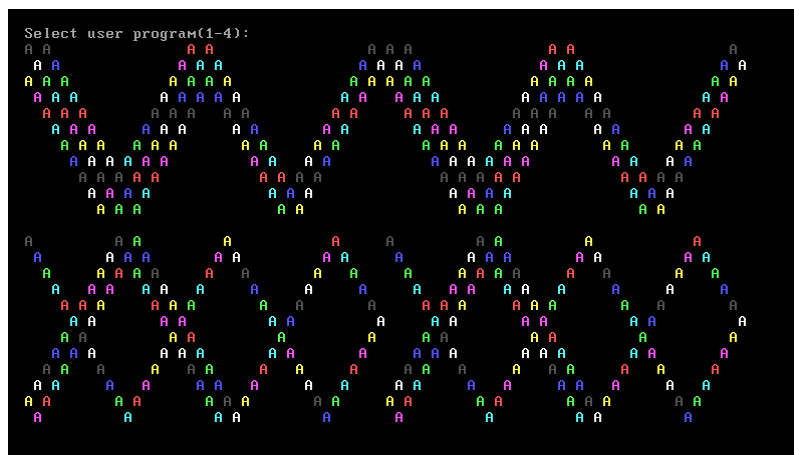
开机，根据提示输入字符(1-4，分别代表一个用户程序)

```
Select user program(1-4):
```

输入1



运行结束后，继续依次输入2，3，4



实验总结

这次实验让我了解到了.com文件和.bin文件的区别，一个是有一个程序头，有一定的格式，另一个就是直接从代码编译成机器码。一开始我不理解为什么要得到.com文件时要加一个org 0x100伪指令，这就是因为.com执行文件通常被加载到一个段开始的地方，即cs:0x00处，但是程序前面0x100个字节可以用作栈，所以执行的代码0x100开始。还有就是加深了对jmp指令的理解。这次实验涉及了程序之间的跳转，如果不处理好就会出错。跳转时要处理好cs、ip的值，比如从监控程序跳转到用户程序，需要用jmp 0x800:0x100，这是由于.com的文件格式。如果把用户程序生成.bin文件，并用org 0x8100指定偏移，则用jmp 0x8100跳转。

参考资料

<https://blog.csdn.net/aurorayqz/article/details/71680753>