
介绍

这本书的每个检测点，实验和课程设计都是经过我亲自编写运行的，因此在运行方面没有问题。如果大家在做题当中，发现有错误，请告知我，我会及时更改，谢谢！因为 Word 没办法保存 gif 所以有些图片会有点奇怪，哈哈！自己去探索。对于《汇编语言》（王爽）中每一章的知识和问题，我也有自己的理解和分析，如果大家需要可以到：

<https://www.cnblogs.com/Mayfly-nymph/category/1483573.htm>

查看，下面的答案也是从这里面提取的。

环境和配置

这里推荐两款 Windows 下的汇编编译器：

<https://www.lanzous.com/i81269g>

<https://www.lanzous.com/i8126xa>

前面那款是我一直在用的，里面有我学习这本书的源码。

划重点，敲黑板

联系方式，欢迎各路神仙来交流。



欢迎各位小伙伴关注的的公众号：Star Bottle，主要为大家分享我的数学建模，CTF 经历以及学习当中的一些收获与感受等。



| | |
|------------------------------------|----|
| 第二章 寄存器..... | 4 |
| 检测点 2.1..... | 4 |
| 检测点 2.2..... | 4 |
| 检测点 2.3..... | 5 |
| 实验 1 查看 CPU 和内存, 用机器指令和汇编指令编程..... | 5 |
| 2.实验任务..... | 9 |
| 第三章 寄存器..... | 12 |
| 检测点 3.2..... | 14 |
| 实验 2 用机器指令和汇编指令编程..... | 15 |
| 2.实验任务..... | 15 |
| 第四章 第一个程序..... | 16 |
| 实验 3 编程, 编译连接, 跟踪..... | 16 |
| 第五章 [bx]和 loop 指令..... | 18 |
| 实验 4 [bx]和 loop 的使用..... | 18 |
| 第六章 包含多个段的程序..... | 20 |
| 检测点 6.1..... | 20 |
| 第七章 更灵活的定位内存地址的方法..... | 26 |
| 实验 6 实践课程中的程序..... | 26 |
| 第八章 数据处理的两个基本问题..... | 27 |
| 实验 7 寻址方式在结构化数据访问的应用..... | 27 |
| 第九章 转移指令的原理..... | 29 |
| 检测点 9.1..... | 29 |
| 检测点 9.2..... | 32 |
| 检测点 9.3..... | 33 |
| 实验 8 分析一个奇怪的程序..... | 33 |
| 实验 9 根据材料编程..... | 34 |
| 第十章 CALL 和 RET 指令..... | 38 |
| 检测点 10.1..... | 38 |
| 检测点 10.2..... | 39 |
| 检测点 10.3..... | 40 |
| 检测点 10.4..... | 41 |
| 检测点 10.5..... | 42 |
| 实验 10 编写子程序..... | 43 |
| 课程设计 1..... | 52 |
| 第十一章 标志寄存器..... | 57 |
| 检测点 11.1..... | 57 |
| 检测点 11.2..... | 58 |
| 检测点 11.3..... | 59 |
| 检测点 11.4..... | 60 |
| 实验 11 编写子程序..... | 60 |
| 第十二章 内中断的产生..... | 61 |
| 检测点 12.1..... | 61 |
| 实验 12 编写 0 号中断的处理程序..... | 61 |
| 第十三章 int 指令..... | 63 |

| | |
|--------------------------------|----|
| 检测点 13.1 | 63 |
| 检测点 13.2 | 65 |
| 实验 13 编写, 应用中断例程 | 66 |
| 第十四章 端口 | 72 |
| 检测点 14.2 | 72 |
| 实验 14 访问 CMOS RAM | 73 |
| 第十五章 外中断 | 75 |
| 检测点 15.1 | 75 |
| 实验 15 安装新的 int 9 中断例程 | 76 |
| 第十六章 直接定址表 | 78 |
| 检测点 16.1 | 78 |
| 检测点 16.2 | 79 |
| 实验 16 编写包含多个功能子程序的中断例程。 | 80 |
| 第十七章 使用 BIOS 进行键盘输入和磁盘读写 | 84 |
| 检测点 17.1 | 84 |
| 试验 17 编写包含多个功能子程序的中断例程 | 84 |
| 课程设计 2 | 87 |
| 总结 | 97 |

第二章 寄存器

检测点 2.1

(1)

| | |
|--------------|----------|
| mov ax,62627 | ax=f4a3h |
| mov ah,31h | ax=31A3h |
| mov al,23h | ax=3123h |
| add ax,ax | ax=6246h |
| mov bx,826ch | bx=826ch |
| mov cx,ax | cx=6246h |
| mov ax,bx | ax=826ch |
| add ax,bx | ax=04d8h |
| mov al,bh | ax=0482h |
| mov ah,bl | ax=6c82h |
| add ah,ah | ax=d882h |
| add al,6 | ax=d888h |
| add al,al | ax=d810h |
| mov ax,cx | ax=6246h |

高低位计算各管各。

(2)

```
mov ax,2
add ax,ax
add ax,ax
add ax,ax
```

检测点 2.2

(1) 给定段地址为 0001H, 仅通过变化偏移地址寻址, CPU 的寻址范围为 0010H 到 100FH。

解题过程:

物理地址 = SA*16+EA

EA 的变化范围为 0h~ffffh

物理地址范围为(SA*16+0h)~(SA*16+ffffh)

现在 SA=0001h,那么寻址范围为

(0001h*16+0h)~(0001h*16+ffffh)

=0010h~1000fh

(2) 有一数据存放在内存 20000H 单元中, 现给定段地址为 SA, 若想用偏移地址寻到此单元。则 SA 应满足的条件是: 最小为 1001H, 最大为 2000H。

当段地址给定为 1001H 以下和 2000H 以上, CPU 无论怎么变化偏移地址都无法寻到 20000H 单元。

解题过程:

物理地址 = SA*16+EA

20000h = SA*16+EA

SA=(20000h-EA)/16=2000h-EA/16

EA 取最大值时,SA=2000h-ffffh/16=1001h,SA 为最小值

EA 取最小值时,SA=2000h-0h/16=2000h,SA 为最大值

检测点 2.3

四次

1) mov ax,bx

2) sub ax,ax

3) 第三项指令读入之后, IP 自动增加

4) jmp ax

实验 1 查看 CPU 和内存, 用机器指令和汇编指令编程

Debug 命令:

- R: 查看, 改变 CPU 寄存器的内容
- D: 查看内存中的内容
- E: 改写内存中的内容
- U: 将内存中的机器指令翻译成汇编指令
- T: 执行一条机器指令
- A: 以汇编指令格式在内存中写入一条机器指令

R 指令查看, 改变

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
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 7403          JZ      0105
-r ax
AX 0000
:200
-r
AX=0200 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 7403          JZ      0105
-r cs
CS 073F
:02c2
-r
AX=0200 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=02C2 IP=0100  NU UP EI PL NZ NA PO NC
02C2:0100 8B0E234A      MOV     CX,[4A23]          DS:4A23=5D08
-r ip
IP 0100
:0200
-r
AX=0200 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=02C2 IP=0200  NU UP EI PL NZ NA PO NC
02C2:0200 BC1D4A      MOV     SP,4A1D

```

D 查看内存中的内容

d 1000:0

从地址 1000:0 开始显示 128 个内存单元

即 1000:0~1000:7F

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
C:\>debug
-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 7403          JZ      0105
-d 1000:0
1000:0000  69 76 65 20 25 63 20 61-6E 64 20 70 72 65 73 73   ive %c and press
1000:0010  20 3C 45 4E 54 45 52 3E-0A 00 6B 00 41 62 6F 75   <ENTER>..k.Abou
1000:0020  74 20 74 6F 20 67 65 6E-65 72 61 74 65 20 2E 45   t to generate .E
1000:0030  58 45 20 66 69 6C 65 00-6D 00 56 61 6C 69 64 20   XE file.m.Valid
1000:0040  6F 70 74 69 6F 6E 73 20-61 72 65 3A 00 FF FF 00   options are:....
1000:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00   .....
1000:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00   .....
1000:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00   .....
-d 1000:9
1000:0000                                64 20 70 72 65 73 73   d press
1000:0010  20 3C 45 4E 54 45 52 3E-0A 00 6B 00 41 62 6F 75   <ENTER>..k.Abou
1000:0020  74 20 74 6F 20 67 65 6E-65 72 61 74 65 20 2E 45   t to generate .E
1000:0030  58 45 20 66 69 6C 65 00-6D 00 56 61 6C 69 64 20   XE file.m.Valid
1000:0040  6F 70 74 69 6F 6E 73 20-61 72 65 3A 00 FF FF 00   options are:....
1000:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00   .....
1000:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00   .....
1000:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00   .....
1000:0080  00 00 00 00 00 00 00 00-00

```

1000:0000---- 69 ---- i

1000:0011-----3C-----空格

中间"- "分隔前八位和后八位

每个位置的数值（转换成 10 进制）依次对应后面的 ASCII 码，没有可显示 ASCII 码用"."表

示

指定范围查看:

d 1000:0 9

```
-d 1000:0
1000:0000 69 76 65 20 25 63 20 61-6E 64 20 70 72 65 73 73   ive %c and press
1000:0010 20 3C 45 4E 54 45 52 3E-0A 00 6B 00 41 62 6F 75   <ENTER>..K.Abou
1000:0020 74 20 74 6F 20 67 65 6E-65 72 61 74 65 20 2E 45   t to generate .E
1000:0030 58 45 20 66 69 6C 65 00-6D 00 56 61 6C 69 64 20   XE file.m.Valid
1000:0040 6F 70 74 69 6F 6E 73 20-61 72 65 3A 00 FF FF 00   options are:....
1000:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .....
1000:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .....
1000:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .....
-e 1000:0 1 2 3 4 5 6 7 8 9
-d 1000:0
1000:0000 01 02 03 04 05 06 07 08-09 64 20 70 72 65 73 73   .....d press
1000:0010 20 3C 45 4E 54 45 52 3E-0A 00 6B 00 41 62 6F 75   <ENTER>..K.Abou
1000:0020 74 20 74 6F 20 67 65 6E-65 72 61 74 65 20 2E 45   t to generate .E
1000:0030 58 45 20 66 69 6C 65 00-6D 00 56 61 6C 69 64 20   XE file.m.Valid
1000:0040 6F 70 74 69 6F 6E 73 20-61 72 65 3A 00 FF FF 00   options are:....
1000:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .....
1000:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .....
1000:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   .....
```

一个一个修改内存中的内容

空格表示下一个 enter 表示结束

```
-d 1000:10 19
1000:0010 20 3C 45 4E 54 45 52 3E-0A 00   <ENTER>..
-e 1000:10
1000:0010 20.88 3C. 45.3 4E.2 54. 45.1 52. 3E.
1000:0018 0A. 00. 6B. 00. 41. 62. 6F. 75.
1000:0020 74.

-d 1000:10 19
1000:0010 88 3C 05 02 54 01 52 3E-0A 00   .<..T.R>..
```

写入字符和字符串

```
-d 1000:20 27
1000:0020 74 20 74 6F 20 67 65 6E   t to gen
-e 1000:20 1 'h' 'y' 2
-d 1000:20 27
1000:0020 01 68 79 02 20 67 65 6E   .hy. gen
-d 1000:28 2f
1000:0020 65 72 61 74 65 20 2E 45   erate .E
-e 1000:28 1 "hello" "world!" 2
-d 1000:28 2f
1000:0020 01 68 65 6C 6C 6F 77 6F   .helloworld
```

将机器码写入内存

```
b80100 mov ax,0001
b90200 mov cx,0002
01c8 add ax,cx
```

```

C:\>debug
-u 1000:0
1000:0000 0102      ADD     [BP+SI],AX
1000:0002 0304      ADD     AX,[SI]
1000:0004 050607     ADD     AX,0706
1000:0007 0809      OR      [BX+DI],CL
1000:0009 64         DB      64
1000:000A 207072     AND     [BX+SI+72],DH
1000:000D 65         DB      65
1000:000E 7373      JNB     0083
1000:0010 883C      MOV     [SI],BH
1000:0012 0302      ADD     AX,[BP+SI]
1000:0014 54         PUSH   SP
1000:0015 01523E     ADD     [BP+SI+3E],DX
1000:0018 0A00      OR      AL,[BX+SI]
1000:001A 6B         DB      6B
1000:001B 004162     ADD     [BX+DI+62],AL
1000:001E 6F         DB      6F
1000:001F 7501      JNZ     0022

```

```

-e 1000:0 b8 01 00 b9 02 00 01 c8
-u 1000:0
1000:0000 B80100     MOV     AX,0001
1000:0003 B90200     MOV     CX,0002
1000:0006 01CB      ADD     AX,CX
1000:0008 096420     OR      [SI+20],SP
1000:000B 7072      JO      007F
1000:000D 65         DB      65
1000:000E 7373      JNB     0083
1000:0010 883C      MOV     [SI],BH
1000:0012 0302      ADD     AX,[BP+SI]
1000:0014 54         PUSH   SP
1000:0015 01523E     ADD     [BP+SI+3E],DX
1000:0018 0A00      OR      AL,[BX+SI]
1000:001A 6B         DB      6B
1000:001B 004162     ADD     [BX+DI+62],AL
1000:001E 6F         DB      6F
1000:001F 7501      JNZ     0022

```

第二列就是指令对应的机器码

在上面修改了内存的指令的条件下，我们执行这三条指令

1.修改 CS:IP 指向第一条指令

```

-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 7403      JZ      0105
-r cs
CS 073F
:1000
-r ip
IP 0100
:0
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0000  NU UP EI PL NZ NA PO NC
1000:0000 B80100     MOV     AX,0001

```


2. 执行内存中的指令

```
-t
AX=0001 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0003  NU UP EI PL NZ NA PO NC
1000:0003 B90200          MOV     CX,0002
-t
AX=0001 BX=0000 CX=0002 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0006  NU UP EI PL NZ NA PO NC
1000:0006 01C8          ADD     AX,CX
-t
AX=0003 BX=0000 CX=0002 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0008  NU UP EI PL NZ NA PE NC
1000:0008 096420          OR     [SI+20],SP          DS:0020=FFFF
-
```

A 命令以汇编指令形式在内存中写入机器指令

```
-a 1000:0
1000:0000 mov ax,8
1000:0003 mov bx,6
1000:0006 sub ax,2
1000:0009 add ax,bx
1000:000B
-u 1000:0
1000:0000 B80800          MOV     AX,0008
1000:0003 BB0600          MOV     BX,0006
1000:0006 2D0200          SUB     AX,0002
1000:0009 01D8          ADD     AX,BX
1000:000B 7072          JO     007F
1000:000D 65          DB     65
1000:000E 7373          JNB    0083
1000:0010 883C          MOV     [SI],BH
1000:0012 0302          ADD     AX,[BP+SI]
1000:0014 54          PUSH   SP
1000:0015 01523E        ADD     [BP+SI+3E],DX
1000:0018 0A00          OR     AL,[BX+SI]
1000:001A 6B          DB     6B
1000:001B 004162        ADD     [BX+DI+62],AL
1000:001E 6F          DB     6F
1000:001F 7501          JNZ    0022
-
```

2. 实验任务

(1)

```

-e 1000:0 b8 20 4e 05 16 14 bb 00 20 01 d8 89 c3 01 d8 b8 1a 00 bb 26 00 00 d8
-u 1000:0
  ^ Error
-u 1000:0
1000:0000 B8204E      MOV     AX,4E20
1000:0003 051614      ADD     AX,1416
1000:0006 BB0020      MOV     BX,2000
1000:0009 01D8        ADD     AX,BX
1000:000B 89C3        MOV     BX,AX
1000:000D 01D8        ADD     AX,BX
1000:000F B81A00      MOV     AX,001A
1000:0012 BB2600      MOV     BX,0026

```

```

-e 1000:15 00 d8 00 dc 00 c7 b4 00 00 d8 04 8c
-u 1000:0
1000:0000 B8204E      MOV     AX,4E20
1000:0003 051614      ADD     AX,1416
1000:0006 BB0020      MOV     BX,2000
1000:0009 01D8        ADD     AX,BX
1000:000B 89C3        MOV     BX,AX
1000:000D 01D8        ADD     AX,BX
1000:000F B81A00      MOV     AX,001A
1000:0012 BB2600      MOV     BX,0026
1000:0015 00D8        ADD     AL,BL
1000:0017 00DC        ADD     AH,BL
1000:0019 00C7        ADD     BH,AL
1000:001B B400        MOV     AH,00
1000:001D 00D8        ADD     AL,BL
1000:001F 048C        ADD     AL,8C

```

运行

```

-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0000  NU UP EI PL NZ NA PE NC
1000:0000 B8204E      MOV     AX,4E20
-t
AX=4E20 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0003  NU UP EI PL NZ NA PE NC
1000:0003 051614      ADD     AX,1416
-t
AX=6236 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0006  NU UP EI PL NZ NA PE NC
1000:0006 BB0020      MOV     BX,2000
-t
AX=6236 BX=2000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=0009  NU UP EI PL NZ NA PE NC
1000:0009 01D8        ADD     AX,BX
-t
AX=8236 BX=2000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=1000 IP=000B  OU UP EI NG NZ NA PE NC
1000:000B 89C3        MOV     BX,AX

```

(2)

将内存与定位地址到 2000:0 之后

```

AX=0040 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP      0003
-t

AX=0040 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0003  NU UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX,AX
-t

AX=0080 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP      0003
-t

AX=0080 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0003  NU UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX,AX
-t

AX=0100 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PE NC
2000:0005 EBFC          JMP      0003

```

(3)

FFF0H 也就是 $FFF0H * 10 + 0$
 段地址为 FFF0H, 偏移地址为 0
 同样 FFFFH= $FFF0H * 10 + FFH$
 所以直接查看
 d fff0:0 ff

```

-d fff0:0 ff
FFF0:0000  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:0010  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:0020  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:0030  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:0040  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:0050  00 00 00 CF 00 50 B0 20-E6 20 58 CF 00 00 00 00  ....P..X....
FFF0:0060  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:0070  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:0080  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:0090  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:00A0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:00B0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:00C0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:00D0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:00E0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
FFF0:00F0  EA C0 12 00 F0 30 31 2F-30 31 2F 39 32 00 FC 55  ....01/01/92..U

```

用的虚拟环境，所以看不了电脑真正的生产日期

(3)

```

-d 1111:1
1111:0000 01 01 02 02 03 03 04-04 6F 72 20 25 63 25 30 .....or %c%0
1111:0010 34 64 3A 20 00 0A 00 20-70 6F 73 3A 20 25 6C 78 4d: ... pos: %lx
1111:0020 20 52 65 63 6F 72 64 20-74 79 70 65 3A 20 25 30 Record type: %0
1111:0030 32 78 0A 00 0A 00 FF FF-00 00 00 00 00 00 00 00 2x.....
1111:0040 37 00 05 42 4C 41 4E 4B-00 07 42 43 5F 56 41 52 7..BLANK..BC_UAR
1111:0050 53 00 0A 5F 5F 61 75 6C-73 74 61 72 74 00 08 46 S..__aulstart..F
1111:0060 41 52 5F 44 41 54 41 00-07 46 41 52 5F 42 53 53 AR_DATA..FAR_BSS
1111:0070 00 04 2E 51 4C 42 00 00-00 00 00 00 00 00 00 00 ...QLB.....
1111:0080 00
-d 2222:2
2222:0000 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
2222:0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2222:0020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2222:0030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2222:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2222:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2222:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2222:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2222:0080 00 00
-e 2222:2 1 2 3 4
-d 2222:2 5
2222:0000 01 02 03 04 .....

```

第三章 寄存器

检测点 3.1

(1)

| | |
|------------|-------|
| mov ax,1 | |
| mov ds,ax | |
| mov ax,[0] | 2662h |
| mov bx,[1] | e626h |
| mov ax,bx | e626h |
| mov ax,[0] | 2662h |
| mov bx,[2] | d6e6h |
| add ax,bx | fd48h |
| add ax,[4] | 2c14h |
| mov ax,0 | 0h |
| mov al,[2] | 00e6h |
| mov bx,0 | 0h |

| | |
|------------|-------|
| mov bl,[c] | 0026h |
| add al,bl | 000ch |

段地址：偏移地址只是表示物理地址的方式，我们查看数据地址只看 段地址*16+偏移地址的结果，例如 0110:0000 和 0000:1100 就是同一地址，数据相同。

```

-d 110:0
0110:0000  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0110:0010  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0110:0020  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0110:0030  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0110:0040  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0110:0050  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0110:0060  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0110:0070  4D 18 01 12 00 00 00 00 00-00 00 00 00 00 00 00  M.....
-d 0:1100
0000:1100  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0000:1110  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0000:1120  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0000:1130  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0000:1140  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0000:1150  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0000:1160  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
0000:1170  4D 18 01 12 00 00 00 00 00-00 00 00 00 00 00 00  M.....

```

- (2)
- 第一问：执行序列：
- 1) mov ax,6622h
 - 2) jmp 0ff0:0100
 - 3) mov ax,2000
 - 4) mov ds,ax
 - 5) mov ax,[8]
 - 6) mov ax,[2]

第二问：

```

-u 2000:0 8
2000:0000 B82266      MOV     AX,6622
2000:0003 EA0001F00F      JMP     0FF0:0100
2000:0008 89C3          MOV     BX,AX
-u 1000:0 8
1000:0000 B80020      MOV     AX,2000
1000:0003 8ED8      MOV     DS,AX
1000:0005 A10800      MOV     AX,[0008]
1000:0008 A10200      MOV     AX,[0002]

```

```

-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=2000 IP=0000 NU UP EI PL NZ NA PO NC
2000:0000 B82266          MOV     AX,6622
-t
AX=6622 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=2000 IP=0003 NU UP EI PL NZ NA PO NC
2000:0003 EA0001F00F     JMP     0FF0:0100
-t
AX=6622 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=0FF0 IP=0100 NU UP EI PL NZ NA PO NC
0FF0:0100 B80020          MOV     AX,2000
-t
AX=2000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=0FF0 IP=0103 NU UP EI PL NZ NA PO NC
0FF0:0103 8ED8          MOV     DS,AX
-t
AX=2000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=2000 ES=073F SS=073F CS=0FF0 IP=0105 NU UP EI PL NZ NA PO NC
0FF0:0105 A10800          MOV     AX,[0008]                      DS:0008=C389

```

```

AX=2000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=2000 ES=073F SS=073F CS=0FF0 IP=0105 NU UP EI PL NZ NA PO NC
0FF0:0105 A10800          MOV     AX,[0008]                      DS:0008=C389
-t
AX=C389 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=2000 ES=073F SS=073F CS=0FF0 IP=0108 NU UP EI PL NZ NA PO NC
0FF0:0108 A10200          MOV     AX,[0002]                      DS:0002=EA66
-t
AX=EA66 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=2000 ES=073F SS=073F CS=0FF0 IP=010B NU UP EI PL NZ NA PO NC
0FF0:010B 007265          ADD     [BP+SI+65],DH                  SS:0065=20

```

第三问：数据和地址有区别，程序(指令)是对 8086CPU 运行顺序的改变，数据是对地址中数据的改变，显著区别就是操作指令的不同。

检测点 3.2

(1)

```

mov ax,2000H
mov ss,ax
mov sp,10H

```

(2)

```

mov ax,1000H
mov ss,ax
mov sp,0H

```

实验 2 用机器指令和汇编指令编程

(2)

d ds:10 18 ;查看 1000:0010~1000:0018 中的内容

d cs:0 ;查看当前代码段中的指令代码

d ss:0 ;查看当前栈段中的内容

u cs:0 ;以汇编指令的形式, 显示当前代码段中的代码

a ds:0 ;以汇编形式从 1000:0 开始的内存写入指令

1--(3)

Debug 的 T 命令在执行修改寄存器 SS 指令之后, 下一条指令也紧接着被执行, 也就是不会再 DOSBOX 上显示

2.实验任务

(1)

代码存入之后, T 指令执行, D 指令查看内存就行

```
AX=0012 BX=30F0 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=2200 CS=1000 IP=0017 NU UP EI PL NZ NA PO NC
1000:0017 8B1E0600 MOV BX,[0006] DS:0006=2F31
-t
AX=0012 BX=2F31 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=2200 CS=1000 IP=001B NU UP EI PL NZ NA PO NC
1000:001B 50 PUSH AX
-t
AX=0012 BX=2F31 CX=0000 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=2200 CS=1000 IP=001C NU UP EI PL NZ NA PO NC
1000:001C 53 PUSH BX
-d 2200:FE
2200:00F0 12 00 ..
2200:0100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
```

```

2200:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 .....
-T
AX=0012 BX=2F31 CX=0000 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=2200 CS=1000 IP=001D NU UP EI PL NZ NA PO NC
1000:001D 5B POP AX
-D 2200:FC
2200:00F0 31 2F 12 00 1/..
2200:0100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
2200:0170 00 00 00 00 00 00 00 00-00 00 00 00 .....
-T
AX=2F31 BX=2F31 CX=0000 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=FFFF ES=073F SS=2200 CS=1000 IP=001E NU UP EI PL NZ NA PO NC
1000:001E 5B POP BX

```

(2)

猜测栈空间被写入了数据

第四章 第一个程序

实验 3 编程，编译连接，跟踪

(1) (2)

```

-u CS: R
^ Error
-U CS:0
076A:0000 B80020 MOV AX,2000
076A:0003 8ED0 MOV SS,AX
076A:0005 BC0000 MOV SP,0000
076A:0008 83C40A ADD SP,+0A
076A:000B 5B POP AX
076A:000C 5B POP BX
076A:000D 50 PUSH AX
076A:000E 53 PUSH BX
076A:000F 5B POP AX
076A:0010 B8004C MOV AX,4C00
076A:0013 CD21 INT 21
076A:0015 0000 ADD [BX+SI],AL
076A:0017 0000 ADD [BX+SI],AL
076A:0019 0000 ADD [BX+SI],AL
076A:001B 0000 ADD [BX+SI],AL
076A:001D 0000 ADD [BX+SI],AL
076A:001F 0000 ADD [BX+SI],AL

```



```

-R
AX=FFFF BX=0000 CX=0015 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0000 NU UP EI PL NZ NA PO NC
076A:0000 B80020 MDU AX,2000
-T
AX=2000 BX=0000 CX=0015 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0003 NU UP EI PL NZ NA PO NC
076A:0003 8ED0 MDU SS,AX
-T
AX=2000 BX=0000 CX=0015 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=2000 CS=076A IP=0008 NU UP EI PL NZ NA PO NC
076A:0008 83C40A ADD SP,+0A
-D 2000:0 F
2000:0000 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-T
AX=2000 BX=0000 CX=0015 DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=2000 CS=076A IP=000B NU UP EI PL NZ NA PE NC
076A:000B 5B POP AX
-D 2000:A F
2000:0000 00 00 00 00 00 00 .....
-T
AX=0000 BX=0000 CX=0015 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=2000 CS=076A IP=000C NU UP EI PL NZ NA PE NC
076A:000C 5B POP BX
-D 2000:C F
2000:0000 00 00 00 00 ....
-T
AX=0000 BX=0000 CX=0015 DX=0000 SP=000E BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=2000 CS=076A IP=000D NU UP EI PL NZ NA PE NC
076A:000D 50 PUSH AX
-D 2000:E F
2000:0000 00 00 ..
-T
AX=0000 BX=0000 CX=0015 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=2000 CS=076A IP=000E NU UP EI PL NZ NA PE NC
076A:000E 53 PUSH BX
-D 2000:C F
2000:0000 00 00 00 00 ....
-T
AX=0000 BX=0000 CX=0015 DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=2000 CS=076A IP=000F NU UP EI PL NZ NA PE NC
076A:000F 58 POP AX
-D 2000:A F
2000:0000 00 00 00 00 00 00 .....
-T
AX=0000 BX=0000 CX=0015 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=2000 CS=076A IP=0010 NU UP EI PL NZ NA PE NC
076A:0010 B8004C MDU AX,4C00
-D 2000:C F
2000:0000 00 00 00 00 ....

```

```

-T
AX=4C00 BX=0000 CX=0015 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=2000 CS=076A IP=0013  NU UP EI PL NZ NA PE NC
076A:0013 CD21          INT    21
-D 2000:C
2000:0000                00 00 00 00                ....
2000:0010  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
2000:0020  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
2000:0030  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
2000:0040  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
2000:0050  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
2000:0060  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
2000:0070  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
2000:0080  00 00 00 00 00 00 00 00 00-00 00 00 00  .....

```

(3)

```

Program terminated normally
-D DS:0
075A:0000  CD 20 FF 9F 00 EA FF FF-AD DE 4F 03 A3 01 8A 03  .....0.....
075A:0010  A3 01 17 03 A3 01 92 01-FF FF FF FF FF FF FF FF  .....
075A:0020  FF FF FF FF FF FF FF FF-FF FF FF FF 50 07 F4 FF  .....P...
075A:0030  00 20 14 00 18 00 5A 07-FF FF FF FF 00 00 00 00  .....Z.....
075A:0040  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
075A:0050  CD 21 CB 00 00 00 00 00-00 00 00 00 00 00 00  .!.....
075A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....
075A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  .....

```

第五章 [bx]和 loop 指令

实验 4 [bx]和 loop 的使用

(1)



```

assume cs:code
code segment
    mov ax,20h
    mov ds,ax

    mov bx,0
    mov cx,64
s:   mov [bx],bl
    inc bl
    loop s

    mov ax,4c00h

```

int 21h

code ends

end

```
-u cs:0
076A:0000 B82000      MOV     AX,0020
076A:0003 8ED8             MOV     DS,AX
076A:0005 BB0000      MOV     BX,0000
076A:0008 B94000      MOV     CX,0040
076A:000B 8B1F             MOV     [BX],BL
076A:000D FEC3             INC     BL
076A:000F E2FA             LOOP   000B
076A:0011 B8004C      MOV     AX,4C00
076A:0014 CD21             INT     21
076A:0016 0000             ADD     [BX+SI],AL
076A:0018 0000             ADD     [BX+SI],AL
076A:001A 0000             ADD     [BX+SI],AL
076A:001C 0000             ADD     [BX+SI],AL
076A:001E 0000             ADD     [BX+SI],AL

-d 20:0 3f
0020:0000  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0020:0010  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0020:0020  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0020:0030  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-g 11
AX=0020 BX=0040 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0020 ES=075A SS=0769 CS=076A IP=0011 NU UP EI PL NZ AC PO NC
076A:0011 B8004C      MOV     AX,4C00
-d 20:0 3f
0020:0000  00 01 02 03 04 05 06 07-08 09 0A 0B 0C 0D 0E 0F .....
0020:0010  10 11 12 13 14 15 16 17-18 19 1A 1B 1C 1D 1E 1F .....
0020:0020  20 21 22 23 24 25 26 27-28 29 2A 2B 2C 2D 2E 2F  !"#$%&'()*+,-./
0020:0030  30 31 32 33 34 35 36 37-38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
-
```

(2) 没变，刚好 9 条，第一题应该地址和传递数据是分开的，所以第二题要求 9 条

assume cs:code

code segment

```
mov ax,20h
mov ds,ax
```

```
mov bx,0
mov cx,64
```

```
s:  mov [bx],bl
    inc bl
    loop s
```

```
mov ax,4c00h
int 21h
```

code ends

end

(3)

```
-u cs:0
076A:0000 B82000      MOV     AX,0020
076A:0003 8ED8             MOV     DS,AX
076A:0005 BB0000      MOV     BX,0000
076A:0008 B94000      MOV     CX,0040
076A:000B 881F             MOV     [BX],BL
076A:000D FEC3             INC     BL
076A:000F E2FA             LOOP   000B
076A:0011 B8004C      MOV     AX,4C00
076A:0014 CD21             INT     21
076A:0016 0000             ADD     [BX+SI],AL
076A:0018 0000             ADD     [BX+SI],AL
076A:001A 0000             ADD     [BX+SI],AL
076A:001C 0000             ADD     [BX+SI],AL
076A:001E 0000             ADD     [BX+SI],AL
```

1) 复制的是存放程序代码的数据，从 076a:0~076a:11

0~10h 一共需要执行 17 次

所以第一个空填 076ah,第二个空填 17

第六章 包含多个段的程序

检测点 6.1

(1)

mov cs:[bx],ax

(2)

栈在程序中，所以段地址和当前地址相同，第一个空填 **cs**

从 cs:10h 开始往后 20 个字节，也就是 $cs:9h+20h=cs:29h$ ，栈空间就是 cs:10h~cs:29h

所以在入栈之前，**SP=30h**(第一次入栈时，栈顶在 cs:28h)

第三个空 **pop cs:[bx]**

实验 5 编写，调试具有多个段的程序

```
assume cs:code,ds:data,ss:stack
```

```
data segment
```

```
    dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h
```

```
data ends
```

```
stack segment
```

```
    dw 0,0,0,0,0,0,0
```

stack ends

code segment

```
start:  mov ax,stack
        mov ss,ax
        mov sp,16
```

```
        mov ax,data
        mov ds,ax
```

```
        push ds:[0]
        push ds:[2]
        pop ds:[2]
        pop ds:[0]
```

```
        mov ax,4c00h
        int 21h
```

code ends

end start

**汇编程序多个段，相当于独立的，也就是地址都是 0 开始
因为在调试代码开始前，只有 CS 地址是已知的，data 和 stack 还不知道，数据段的数据要查看到的话**

1.将代码运行代码执行到 mov ds,ax

2.因为这三个段空间是紧密排列，先数据段，再栈段，最后代码段(顺序与代码中的顺序有关)，栈段占 16 字节，数据段占 16 字节，代码段开始在 076c:0h，所以栈段开始在 076b:0h,数据段开始在 076a:0h

1)

23 01 56 04 89 07 BC 0A-EF 0D ED 0F BA 0C 89 09

2)

cs=076ch

ss=076bh

ds=076ah

3)

X-2H

X-1H

(2)

assume cs:code, ds:data, ss:stack

data segment

dw 0123h, 0456h

data ends

stack segment

dw 0, 0

stack ends

code segment

start: mov ax, stack

mov ss, ax

mov sp, 16

mov ax, data

mov ds, ax

push ds:[0]

push ds:[2]

pop ds:[2]

pop ds:[0]

mov ax, 4c00h

int 21h

code ends

end start

1)

```
-d ds:0 f
076A:0000 23 01 56 04 00 00 00 00-00 00 00 00 00 00 00 00 #.U.....
```

2)

```
-t
AX=076A BX=0000 CX=0042 DX=0000 SP=0010 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=076B CS=076C IP=001D NU UP EI PL NZ NA PO NC
076C:001D B8004C MDU AX,4C00
```

3)

X-2H

X-1H

4)

$(N / 16 + 1) * 16$

N / 16 不进行四舍五入的取整，这个公式实际上想说明，不满 16 字节，也占了 16 字节，也就是一行

(3)

1)

```

-d ds:0 f
076D:0000 23 01 56 04 00 00 00 00-00 00 00 00 00 00 00 00 #.U.....

```

2)

```

AX=076D BX=0000 CX=0044 DX=0000 SP=0010 BP=0000 SI=0000 DI=0000
DS=076D ES=075A SS=076E CS=076A IP=001D NU UP EI PL NZ NA PO NC
076A:001D B8004C MJU AX,4C00
-d ds:0 f

```

3)

X+0003H

X+0004H

(4)

(3) 能够正常正确执行，因为 (1) (2) 在程序中栈段和数据段都在代码段的前面，程序的地址不是代码段的地址

(5)

这道题的寄存器会不够，我们采取入栈出栈的方式来反复利用寄存器，这里的 1,2,3 定义的 db,应该是"define byte",字节型数据

```

assume cs:code
a segment

```

```

    db 1,2,3,4,5,6,7,8
a ends
b segment
    db 1,2,3,4,5,6,7,8
b ends
c segment
    db 0,0,0,0,0,0,0,0
c ends
code segment
start:  mov ax,a      ;a 段地址
        mov ds,ax

        mov ax,b      ;b 段地址
        mov es,ax

        mov cx,8      ;循环次数
        mov bx,0      ;交换数据的偏移地址
        mov ax,0      ;临时储存 a+b 的数据
s:      mov al,es:[bx]
        add al,[bx]

        push ds      ;ds 入栈（这两处入栈相当于找了个地方，保存之前
ds,bx 的值）
        push bx      ;bx 入栈
        mov bx,c      ;c 段地址
        mov ds,bx
        pop bx       ;取出 bx
        mov [bx],al   ;将 a+b 之和赋值给对应的 c 段地址数据
        pop ds       ;ds 出栈，ds 保存 a 段段地址
        inc bx       ;偏移地址+1
        loop s

        mov ax,4c00h
        int 21h
code ends
end start

```



```

C:\>debug pro38.exe;
-r
AX=FFFF BX=0000 CX=005B DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076D IP=0000  NU UP EI PL NZ NA PO NC
076D:0000 B86A07          MOV     AX,076A
-d ds:0 f
075A:0000 CD 20 FF 9F 00 EA FF FF-AD DE 4F 03 A3 01 8A 03  . . . . .0. . . .
-d 076a:0 f
076A:0000 01 02 03 04 05 06 07 08-00 00 00 00 00 00 00  . . . . .
-d 076b:0 f
076B:0000 01 02 03 04 05 06 07 08-00 00 00 00 00 00 00  . . . . .
-d 076c:0 f
076C:0000 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  . . . . .
-g 2b
AX=0010 BX=0008 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=076B SS=0769 CS=076D IP=0026  NU UP EI PL NZ NA PO NC
076D:0026 B8004C          MOV     AX,4C00
-d 076c:0 f
076C:0000 02 04 06 08 0A 0C 0E 10-00 00 00 00 00 00 00  . . . . .

```

(6)

先将 a 段数据入栈，再出栈，储存到 b 段对于位置

```

C:\>debug pro39.exe;
-r
AX=FFFF BX=0000 CX=0059 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076D IP=0000  NU UP EI PL NZ NA PO NC
076D:0000 B86A07          MOV     AX,076A

```

DS=075AH，所以程序入口地址在 076AH,根据程序的代码，我们知道先 a 段，再 b 段，a 段占 16 字，32 字节（1f），所以 076A:0H~076A:1FH 都是 a 段数据，b 段占 8 字，16 字节，所以 b 段在 076C:0H~076C:FH

```

assume cs:code
a segment
    dw 1, 2, 3, 4, 5, 6, 7, 8, 9, 0ah, 0bh, 0ch, 0dh, 0eh, 0fh, 0ffh
a ends
b segment
    dw 0, 0, 0, 0, 0, 0, 0, 0
b ends
code segment
start:  mov ax, a
        mov ds, ax

        mov ax, b
        mov es, ax

```

```

        mov bx, 0
        mov cx, 8
s0:     mov ax, [bx]
        push ax
        add bx, 2
        loop s0

        mov bx, 0
s1:     pop ax
        mov es:[bx], ax
        add bx, 2
        loop s1

        mov ax, 4c00h
        int 21h
code ends
end start

```

```

-d 076a:0 1f
076A:0000  01 00 02 00 03 00 04 00-05 00 06 00 07 00 08 00  .....
076A:0010  09 00 0A 00 0B 00 0C 00-0D 00 0E 00 0F 00 FF 00  .....
-d 076c:0 f
076C:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
-g 24
AX=0006 BX=0000 CX=0000 DX=0000 SP=001C BP=0000 SI=0000 DI=0000
DS=076A ES=076C SS=0769 CS=076D IP=0024  NU UP EI PL ZR AC PE CY
076D:0024 B8004C      MOV     AX,4C00
-d 076c:0 f
076C:0000  08 00 07 00 06 00 05 00-04 00 03 00 02 00 01 00  .....

```

第七章 更灵活的定位内存地址的方法

实验 6 实践课程中的程序

前面已做

第八章 数据处理的两个基本问题

实验 7 寻址方式在结构化数据访问的应用

这道出得很好，很综合的考查了前面所学知识。

```
assume cs:code, ds:data, es:table

data segment
db '1975', '1976', '1977', '1978', '1979', '1980', '1981', '1982', '1983'
db '1984', '1985', '1986', '1987', '1988', '1989', '1990', '1991', '1992'
db '1993', '1994', '1995'
;以上是表示 21 年的字符串 4 * 21 = 84

dd 16, 22, 382, 1356, 2390, 8000, 16000, 24486, 50065, 97479, 140417, 197514
dd
345980, 590827, 803530, 1183000, 1843000, 2759000, 3753000, 4649000, 5937000
;以上是表示 21 年公司总收入的 dword 型数据 4 * 21 = 84

dw 3, 7, 9, 13, 28, 38, 130, 220, 476, 778, 1001, 1442, 2258, 2793, 4037, 5635, 8226
dw 11542, 14430, 15257, 17800
;以上是表示 21 年公司雇员人数的 21 个 word 型数据 2 * 21 = 42
data ends

table segment
db 21 dup ('year summ ne ?? '); 'year summ ne ?? ' 刚好 16 个字节
table ends

code segment
start:  mov ax, data
        mov ds, ax
        mov ax, table
        mov es, ax

        mov cx, 21    ;外层循环每一年的情况
        mov bx, 0    ;表示第几年的情况
        mov di, 0    ;存储年份地址

        mov bp, 0    ;1. 获取到 data 段中雇员数数据 2. 保存 bx 的值
s0:     push cx
        mov cx, 4
```

```

    mov si, 0    ;表示这一年中的第几个

    ;年份
s:    mov al, ds:[di]    ;一位一位保存字符
    mov es:[bx+si], al

    ;收入
    mov al, ds:[di+54h]
    mov es:[bx+si+5h], al

    inc si
    inc di
    loop s

    ;雇员数
    mov ax, ds:[bp+0a8h]    ;单独保存数据
    mov es:[bx+0ah], ax
    add bp, 2    ;到段的下一个字的数据

    ;人均收入
    mov dx, es:[bx+7h]    ;高 16 位被除数
    mov ax, es:[bx+5h]    ;低 16 位被除数
    push bx    ;存放除数
    push bp    ;存放 bx 地址
    mov bp, bx
    mov bx, es:[bp+0ah]
    div bx
    mov es:[bp+0dh], ax    ;余数在 dx 中, 只保存了商
    pop bp
    pop bx

    pop cx
    add bx, 10h
    loop s0

    mov ax, 4c00h
    int 21h
code ends
end start

```

```

-d es:0 14f
0778:0000 31 39 37 35 20 10 00 00-00 20 03 00 20 05 00 20 1975 ....
0778:0010 31 39 37 36 20 16 00 00-00 20 07 00 20 03 00 20 1976 ....
0778:0020 31 39 37 37 20 7E 01 00-00 20 09 00 20 2A 00 20 1977 ~...*.
0778:0030 31 39 37 38 20 4C 05 00-00 20 0D 00 20 68 00 20 1978 L...h.
0778:0040 31 39 37 39 20 56 09 00-00 20 1C 00 20 55 00 20 1979 U...U.
0778:0050 31 39 38 30 20 40 1F 00-00 20 26 00 20 D2 00 20 1980 @...&.
0778:0060 31 39 38 31 20 80 3E 00-00 20 82 00 20 7B 00 20 1981 .>...{.
0778:0070 31 39 38 32 20 A6 5F 00-00 20 DC 00 20 6F 00 20 1982 _...o.
0778:0080 31 39 38 33 20 91 C3 00-00 20 DC 01 20 69 00 20 1983 ....i.
0778:0090 31 39 38 34 20 C7 7C 01-00 20 0A 03 20 7D 00 20 1984 .!...}.
0778:00A0 31 39 38 35 20 81 24 02-00 20 E9 03 20 8C 00 20 1985 .$...
0778:00B0 31 39 38 36 20 8A 03 03-00 20 A2 05 20 88 00 20 1986 ....
0778:00C0 31 39 38 37 20 7C 47 05-00 20 D2 08 20 99 00 20 1987 iG...
0778:00D0 31 39 38 38 20 EB 03 09-00 20 E9 0A 20 D3 00 20 1988 ....
0778:00E0 31 39 38 39 20 CA 42 0C-00 20 C5 0F 20 C7 00 20 1989 .B...
0778:00F0 31 39 39 30 20 18 0D 12-00 20 03 16 20 D1 00 20 1990 ....
0778:0100 31 39 39 31 20 38 1F 1C-00 20 22 20 20 E0 00 20 1991 8...".
0778:0110 31 39 39 32 20 58 19 2A-00 20 16 2D 20 EF 00 20 1992 X.*. -..
0778:0120 31 39 39 33 20 28 44 39-00 20 5E 38 20 04 01 20 1993 (D9. ^8..
0778:0130 31 39 39 34 20 28 F0 46-00 20 99 3B 20 30 01 20 1994 (.F. ;0.
0778:0140 31 39 39 35 20 68 97 5A-00 20 88 45 20 4D 01 20 1995 h.Z. .E M.

```

第九章 转移指令的原理

检测点 9.1

- (1) 我想的这道题就是令 IP=0 就行，也就是 ds:[bx+1]处的值要为 0

```

assume cs:codesg
datasg segment
    dw 5 dup (0)
datasg ends
codesg segment
start:    mov ax,datasg
          mov ds,ax
          mov bx,0
          jmp word ptr [bx+1]

          mov ax,4c00h
          int 21h
codesg ends
end start

```

```

AX=076A BX=0000 CX=0020 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0005 NU UP EI PL NZ NA PO NC
076B:0005 BB0000          MOV     BX,0000
-t

AX=076A BX=0000 CX=0020 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=000B NU UP EI PL NZ NA PO NC
076B:0008 FF6701          JMP     [BX+01]          DS:0001=0000
-t

AX=076A BX=0000 CX=0020 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0000 NU UP EI PL NZ NA PO NC
076B:0000 B86A07          MOV     AX,076A
-t

AX=076A BX=0000 CX=0020 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0003 NU UP EI PL NZ NA PO NC
076B:0003 8ED8          MOV     DS,AX
-t

AX=076A BX=0000 CX=0020 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0005 NU UP EI PL NZ NA PO NC
076B:0005 BB0000          MOV     BX,0000

```

(2)

这道和上面一样的思想，不过多了一个更改 CS 段地址，段地址可以根据 datasg 段来算

```

-r
AX=FFFF BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076B IP=0000 NU UP EI PL NZ NA PO NC
076B:0000 B86A07          MOV     AX,076A

```

我们得到，DS=075AH，所以程序的地址是 076AH

从已经给出的代码中，我们得到 datasg 段占 4 个字节，不满 16 字节（一行），但是在系统中还是占一行。

所以，我们推出程序代码的起始地址在 076BH

```

assume cs:codesg
datasg segment
    dd 12345678h
datasg ends
codesg segment
start:  mov ax,datasg
        mov ds,ax
        mov bx,0
        add ax,1
        mov [bx],bx

```

```

mov [bx+2], ax
jmp dword ptr ds:[0]

mov ax, 4c00h
int 21h
codesg ends
end start

```

```

AX=076B BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=000B  NU UP EI PL NZ NA PO NC
076B:000B 891F          MOV     [BX],BX                      DS:0000=5678
-t
AX=076B BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=000D  NU UP EI PL NZ NA PO NC
076B:000D 894702      MOV     [BX+02],AX                   DS:0002=1234
-t
exe
AX=076B BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0010  NU UP EI PL NZ NA PO NC
076B:0010 FF2E0000  JMP     FAR [0000]                   DS:0000=0000
-t
AX=076B BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0000  NU UP EI PL NZ NA PO NC
076B:0000 B86A07      MOV     AX,076A
-t
AX=076A BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=076A ES=075A SS=0769 CS=076B IP=0003  NU UP EI PL NZ NA PO NC
076B:0003 B8D8          MOV     DS,AX

```

(3)

```

assume cs:codesg
codesg segment
start:  mov ax, 2000h
        mov es, ax
        mov ax, 00beh ;题中是之前在 2000:0 就有数据，我是 0000...，
只能自己动手丰衣足食了。
        mov es:[1000h], ax
        mov ax, 0006h
        mov es:[1002h], ax
        jmp dword ptr es:[1000h]

        mov ax, 4c00h
        int 21h
codesg ends

```

end start

```

-d 2000:1000 100f
2000:1000 BE 00 06 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
-t
AX=0006 BX=0000 CX=001D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=2000 SS=0769 CS=0006 IP=00BE  NU UP EI PL NZ NA PO NC
0006:00BE 00F0          ADD     AL,DH

```

检测点 9.2

```

assume cs:codesg
codesg segment
start:  mov ax,2000h
        mov ds,ax
        mov bx,0

        mov al,'b'
        mov ds:[bx],al
        mov al,'i'
        mov ds:[bx+1],al

s:      mov cl,[bx]
        mov ch,0
        jcxz s0
        inc bx
        loop s

s0:     mov dx,bx
        mov ax,4c00h
        int 21h
codesg ends
end start

```



```

DS=2000 ES=075A SS=0769 CS=076A IP=0011  NU UP EI PL NZ NA PO NC
076A:0011 8A0F          MOV     CL,[BX]          DS:0002=00
-t

AX=2069 BX=0002 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=2000 ES=075A SS=0769 CS=076A IP=0013  NU UP EI PL NZ NA PO NC
076A:0013 B500          MOV     CH,00
-t

AX=2069 BX=0002 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=2000 ES=075A SS=0769 CS=076A IP=0015  NU UP EI PL NZ NA PO NC
076A:0015 E303          JCXZ   001A
-t

AX=2069 BX=0002 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=2000 ES=075A SS=0769 CS=076A IP=001A  NU UP EI PL NZ NA PO NC
076A:001A 8BD3          MOV     DX,BX
-t

AX=2069 BX=0002 CX=0000 DX=0002 SP=0000 BP=0000 SI=0000 DI=0000
DS=2000 ES=075A SS=0769 CS=076A IP=001C  NU UP EI PL NZ NA PO NC
076A:001C B8004C        MOV     AX,4C00
-d 2000:0 f
2000:0000 62 69 00 00 00 00 00 00-00 00 00 00 00 00 00 00 00  bi.....

```

检测点 9.3

dec bx ;和 inc 相反，自减一

inc cx

实验 8 分析一个奇怪的程序

可以

```

assume cs:codesg
codesg segment
    mov ax,4c00h
    int 21h

start:  mov ax,0
s:      nop
        nop

        mov di,offset s

```

```

        mov si,offset s2
        mov ax,cs:[si]
        mov cs:[di],ax

s0:      jmp short s

s1:      mov ax,0
        int 21h
        mov ax,0

s2:      jmp short s1
        nop

codesg ends
end start

```

```

C:\>debug pro63.exe;
-u cs:0 23
076A:0000 B8004C      MOV     AX,4C00
076A:0003 CD21        INT     21
076A:0005 B80000      MOV     AX,0000
076A:0008 90          NOP
076A:0009 90          NOP
076A:000A BF0800      MOV     DI,0008
076A:000D BE2000      MOV     SI,0020
076A:0010 2E          CS:
076A:0011 8B04        MOV     AX,[SI]
076A:0013 2E          CS:
076A:0014 8905        MOV     [DI],AX
076A:0016 EBF0        JMP     000B
076A:0018 B80000      MOV     AX,0000
076A:001B CD21        INT     21
076A:001D B80000      MOV     AX,0000
076A:0020 EBF6        JMP     001B
076A:0022 90          NOP
076A:0023 0000        ADD     [BX+SI],AL

```

注意红色部分的变化

在我们运行指令时，会发现运行到 `jmp short s` 之后，会运行 `jmp 0000`，而原本应该跳到 `s` 标号处执行 `NOP`，也就是说 `s` 处的指令发生的变化。

实验 9 根据材料编程

注意：

在一页显示缓冲区中：

偏移 000~09F 对应显示器上的第 1 行(80 个字符占 160 个字节)；

偏移 0A0~13F 对应显示器上的第 2 行；

偏移 140~1DF 对应显示器上的第 3 行；

依此类推，可知，偏移 F00~F9F 对应显示器上的第 25 行。

在一行中，一个字符占两个字节的存储空间(一个字)，低位字节存储码，高位字节存储字符的属性。一行共有 80 个字符，占 160 个字节。

即在一行中：

00~01 单元对应显示器上的第 1 列；

02~03 单元对应显示器上的第 2 列；

04~05 单元对应显示器上的第 3 列；

依此类推，可知，9E~9F 单元对应显示器上的第 80 列。

例：在显示器的 0 行 0 列显示黑底绿色的字符串'ABCDEF'

('A'的 ASCII 码值为 41H, 02H 表示黑底绿色)

显示缓冲区里的内容为：

| | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|
| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | ... | 0E |
| B800:0000 | 41 | 02 | 42 | 02 | 43 | 02 | 44 | 02 | 45 | 02 | 46 | 02 | ... | .. |
| : | | | | | | | | | | | | | | |
| : | | | | | | | | | | | | | | |
| B800:00A0 | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. | .. |

在做第十章课后题时，发现个问题，最上面红框和下面两个红框，对偏移地址 000h 的叫法不同，我根据实际显示，es:000h 叫 0 行 0 列（这一行不会在显示器上显示）

```
assume cs:codesg, ds:datasg, ss:stacksg
datasg segment
    db 'welcome to masm!'
    db 02h, 24h, 71h; 三种颜色
datasg ends
stacksg segment
    db 16 dup (0)
stacksg ends
codesg segment
    ;datasg 段地址
start:  mov ax, datasg
        mov ds, ax
```

```

;stacksg 段地址
mov ax, stacksg
mov ss, ax
mov sp, 10h

;目标地址
mov ax, 0b800h
mov es, ax

mov cx, 3
mov di, 10h;在 datasg 中的偏移量
mov bx, 780h;表示第 12 行
;第一层循环, 3 种颜色
s:   mov si, 0;在显示缓冲区中的偏移地址
     mov ah, ds:[di]
     push cx
     push di

     mov di, 0
     mov cx, 16
     ;第二层循环, 字符串
s0:  mov al, ds:[di]
     mov es:[bx+si+64], al;低位存字符
     mov es:[bx+si+64+1], ah;高位存属性
     inc di
     add si, 2
     loop s0

     pop di
     pop cx
     inc di
     add bx, 0a0h
     loop s

     mov ax, 4c00h
     int 21h
codesg ends
end start

```

要理解材料中的几个点:

- 1.每行可以表示 80 个字符, 160 字节
- 2.每行的偏移地址规律----**起始偏移地址: 行数 * 160** **结束偏移地址: 起始地址 +159** (这里都是十进制表示, 需要转换为十六进制)

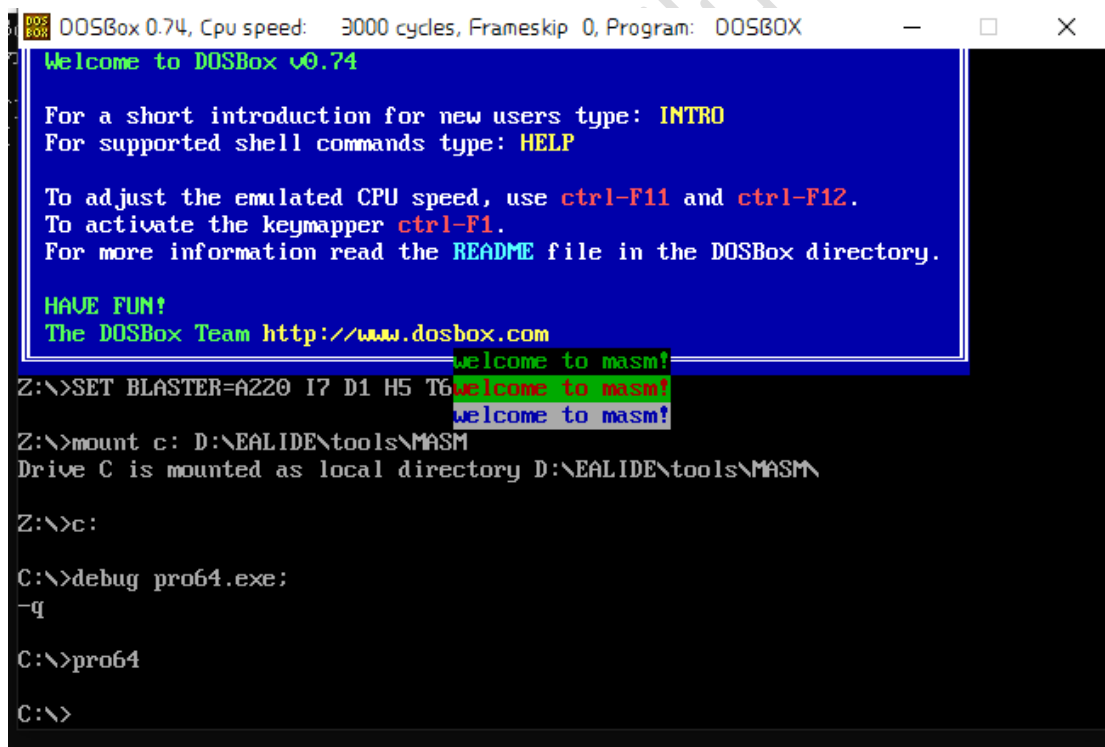
- 3.属性字节格式，一共 8 位，每四位对应十六进制的字节的高位和低位。1 表示有这个属性，0 表示没有。

概括

这道题简单的说就是将**字符数据**和**属性数据**写入 b800h(显示缓冲区)，并居中显示。

分步解析

- 1.要居中，首先行居中为 12 行开始，根据上面的公式得到起始偏移地址 **780h**，而下一行起始地址就是 **原偏移地址+0a0h--0a0h** 为 160 字节；
- 2.列居中，字符串占 16 字符，总共 80 字符，还剩 64 字符，也就是 128 字节，字符串前后等距，为 $128/2=64$ 字节，所以需要在偏移地址基础上加 64
- 2.从自定义的 datasg 段中取字符串和属性数据，字符串写入低位：**0b800h:[起始偏移地址+64]** 属性写入高位：**0b800h:[起始偏移地址+1+64]**



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Welcome to DOSBox v0.74
For a short introduction for new users type: INTRO
For supported shell commands type: HELP
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.
HAVE FUN!
The DOSBox Team http://www.dosbox.com
welcome to masm!
Z:\>SET BLASTER=A220 I7 D1 H5 T6
welcome to masm!
welcome to masm!
Z:\>mount c: D:\EALIDE\tools\MASM
Drive C is mounted as local directory D:\EALIDE\tools\MASM\
Z:\>c:
C:\>debug pro64.exe;
-q
C:\>pro64
C:\>
```

```

-t
AX=F6EB BX=0000 CX=0023 DX=0000 SP=0000 BP=0000 SI=0020 DI=0008
DS=075A ES=075A SS=0769 CS=076A IP=0013  NU UP EI PL NZ NA PO NC
076A:0013 2E          CS:
076A:0014 8905          MOV     EDI,AX          CS:0008=9090
-t
AX=F6EB BX=0000 CX=0023 DX=0000 SP=0000 BP=0000 SI=0020 DI=0008
DS=075A ES=075A SS=0769 CS=076A IP=0016  NU UP EI PL NZ NA PO NC
076A:0016 EBF0          JMP     0008
-t
AX=F6EB BX=0000 CX=0023 DX=0000 SP=0000 BP=0000 SI=0020 DI=0008
DS=075A ES=075A SS=0769 CS=076A IP=0008  NU UP EI PL NZ NA PO NC
076A:0008 EBF6          JMP     0000
-t
AX=F6EB BX=0000 CX=0023 DX=0000 SP=0000 BP=0000 SI=0020 DI=0008
DS=075A ES=075A SS=0769 CS=076A IP=0000  NU UP EI PL NZ NA PO NC
076A:0000 B8004C        MOV     AX,4C00

```

产生这种变化的原因，我们可以理解

```

mov di,offset s
mov si,offset s2
mov ax,cs:[si]
mov cs:[di],ax

```

这里实际上是将，标号 `s2` 处的指令移动到了 `s` 处，而 `jmp` 跳转地址是根据偏移地址改变的，在 `s` 处被替换的指令偏移位置，是 `s2` 处的偏移量。

`s2` 处偏移量为 `18H-22H=-AH`，因为 `jmp` 指令占两个内存单元空间，所以 `s` 处两个 `nop` 变成一条 `jmp` 指令，也就是从 `076A:000AH` 处，向上移动 `AH`，也就是 `0`，所以在跳转到 `s` 处，会显示 `jmp 0000`，进而执行返回指令。

第十章 CALL 和 RET 指令

检测点 10.1

```

assume cs:codesg
stacksg segment
    db 16 dup (0)
stacksg ends
codesg segment
start:  mov ax,stacksg
        mov ss,ax
        mov sp,10h

```

```

    mov ax,1000h
    push ax
    mov ax,0
    push ax
    retf
codesg ends
end start

```

```

AX=1000 BX=0000 CX=0021 DX=0000 SP=0010 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=076A CS=076B IP=000B  NU UP EI PL NZ NA PO NC
076B:000B 50          PUSH  AX
-t
AX=1000 BX=0000 CX=0021 DX=0000 SP=000E BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=076A CS=076B IP=000C  NU UP EI PL NZ NA PO NC
076B:000C B80000     MOV   AX,0000
-t
AX=0000 BX=0000 CX=0021 DX=0000 SP=000E BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=076A CS=076B IP=000F  NU UP EI PL NZ NA PO NC
076B:000F 50          PUSH  AX
-t
AX=0000 BX=0000 CX=0021 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=076A CS=076B IP=0010  NU UP EI PL NZ NA PO NC
076B:0010 CB          RETF
-t
AX=0000 BX=0000 CX=0021 DX=0000 SP=0010 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=076A CS=1000 IP=0000  NU UP EI PL NZ NA PO NC
1000:0000 0000     ADD  EBX,SII,HL          DS:0000=CD

```

检测点 10.2

ax=6

在读入指令 `call s` 时，IP 首先自动+3（下一个指令偏移地址），所以入栈的数据为 6，`pop ax` 也就是 `ax=6`

```

assume cs:codesg
codesg segment
start:  mov ax,0
        call s
        inc ax
s:      pop ax

        mov ax,4c00h

```

```

    int 21h
codesg ends
end start

```

```

AX=0006 BX=0000 CX=000D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0000  NU UP EI PL NZ NA PO NC
076A:0000 B80000          MDU      AX,0000
-t

AX=0000 BX=0000 CX=000D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0003  NU UP EI PL NZ NA PO NC
076A:0003 E80100          CALL   0007
-t

AX=0000 BX=0000 CX=000D DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0007  NU UP EI PL NZ NA PO NC
076A:0007 5B             POP    AX
-t

AX=0006 BX=0000 CX=000D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0008  NU UP EI PL NZ NA PO NC
076A:0008 B8004C          MDU      AX,4C00

```

检测点 10.3

```

assume cs:codesg
codesg segment
    mov ax,4c00h
    int 21h

start:    mov ax,0
          call far ptr s;入栈的 IP 为 8, CS 为 1000h
          inc ax

s:        pop ax;ax=8
          add ax,ax;ax=16
          pop bx;bx=1000h
          add ax,bx;ax=1010h

          mov bx,0
          push bx
          ret

codesg ends
end start

```



```

076A:0018 C3          RET
-t
AX=0784 BX=0000 CX=0019 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0000  NU UP EI PL NZ AC PE NC
076A:0000 B8004C      MOV     AX,4C00
-u cs:0
076A:0000 B8004C      MOV     AX,4C00
076A:0003 CD21      INT     21
076A:0005 B80000      MOV     AX,0000
076A:0008 9A0E006A07    CALL   076A:000E
076A:000D 40          INC     AX
076A:000E 58          POP     AX
076A:000F 03C0      ADD     AX,AX
076A:0011 5B          POP     BX
076A:0012 03C3      ADD     AX,BX
076A:0014 B80000      MOV     BX,0000
076A:0017 53          PUSH   BX
076A:0018 C3          RET
076A:0019 0000      ADD     [BX+SI],SI

```

因为在我的代码中，入栈 CS=076AH，IP=DH

所以

```

assume cs:codesg
codesg segment
    mov ax,4c00h
    int 21h

start:    mov ax,0
          call far ptr s;入栈的 IP 为 dh, CS 为 076ah
          inc ax
s:        pop ax;ax=dh
          add ax,ax;ax=lah
          pop bx;bx=076ah
          add ax,bx;ax=0784h

          mov bx,0
          push bx
          ret
codesg ends
end start

```

检测点 10.4

AX=000BH

这个程序稍微修改一下，就很清晰了。

```

assume cs:codesg
codesg segment
    mov ax,6
    ;首先 sp=sp-2=ffffeh, 将 IP=5 压入栈中
    ;跳转到 ax 处, jmp 6
    call ax
    inc ax
    ;在 bp 的段地址是 ss, 下面两句相当于
    ;pop bp
    ;add ax,bp
    mov bp,sp
    add ax,ss:[bp]

    mov ax,4c00h
    int 21h
codesg ends
end

```

```

C:\>debug prob69.exe;
-u cs:0 f
076A:0000 B80600      MOV     AX,0006
076A:0003 FFD0             CALL   AX
076A:0005 40             INC    AX
076A:0006 8BEC           MOV    BP,SP
076A:0008 034600        ADD    AX,[BP+001]
076A:000B B8004C        MOV    AX,4C00
076A:000E CD21      INT    21
-g b
AX=000B BX=0000 CX=0010 DX=0000 SP=FFFE BP=FFFE SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=000B  NU UP EI PL NZ NA PO NC
076A:000B B8004C      MOV    AX,4C00
-d ss:ffff
0769:FFF0      05 00
-

```

检测点 10.5

(1)

ax=3

在 `call word ptr ds:[0eh]` 时, `ax=0`, 将下一条指令 IP 压入栈中, 然后指令跳到 `ds:[0eh]` (实际上就是栈顶), 所以实际上就是跳到了 `inc ax` 执行, 最后得到 `ax = 3`

(2)

```

assume cs:code
data segment
    dw 8 dup (0)
data ends
code segment
start:  mov ax,data
        mov ss,ax
        mov sp,16
        mov word ptr ss:[0],offset s;将标号 s 的地址存入 ss:[0]
        mov ss:[2],cs
        call word ptr ss:[0]
        ;这里首先将 CS=076bh 和 IP=19h 压入栈中
        ;再跳转到 ss:[0]中的地址处,也就是标号 s 处
        nop
s:      mov ax,offset s
        ;将标号 s 的地址放到 ax=001ah
        sub ax,ss:[0ch]
        ;ss:[0ch]也就是 IP=0019h
        ;ax=1ah-19h=1
        mov bx,cs
        ;bx=076bh
        sub bx,ss:[0eh]
        ;ss:[0eh]也就是 CS=076bh
        ;bx=076bh-076bh=0

        mov ax,4c00h
        int 21h
code ends
end start

```

实验 10 编写子程序

1.显示字符串

步骤:

1. 将行列值转换为正确的偏移地址,保存到寄存器(起始偏移地址: $(\text{行数} - 1) * 160$, 列数偏移地址: $(\text{列数} - 1) * 2$)
2. 读取字符和属性值,判断是否为 0,不是,则存入显示缓冲区

```
assume cs:code
```

```

data segment
    db 'Welcome to masm!',0
data ends
code segment
start:  mov dh,8
        mov dl,3
        mov cl,2
        mov ax,data
        mov ds,ax
        mov si,0
        call show_str

        mov ax,4c00h
        int 21h

show_str:    ;计算行的偏移地址，利用上次得到的结论，起始偏移地址：行数 * 160
            mov al,dh
            mov ah,0
            ;dec ax
            mov bx,160
            push dx;因为下面乘法是16位的，因此 dx 会被用作高位，原来保存颜色的值会消除
            mul bx
            pop dx
            mov di,ax

            ;结算列的偏移地址，地址0开始
            mov bl,dl
            mov bh,0
            dec bx
            add bx,bx

            ;显示缓冲区地址
            mov ax,0b800h
            mov es,ax

            ;颜色参数值(不变)和字符
            mov ah,cl
str:        mov al,ds:[si]

            ;检查是否遇到尾部的0
            mov cl,al
            mov ch,0

```

```
jcxz ok
```

```
;将字符和属性存入显示缓冲区
```

```
mov es:[bx+di],al
```

```
mov es:[bx+di+1],ah
```

```
inc si;字符偏移地址
```

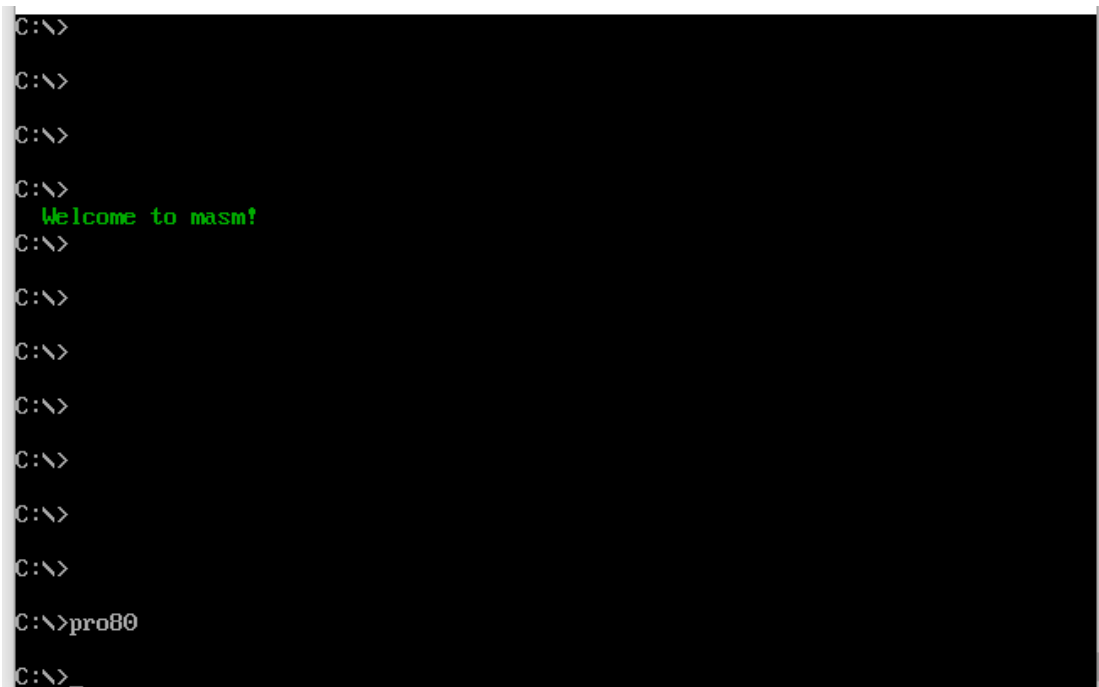
```
add bx,2;列的位置
```

```
loop str
```

```
ok: ret
```

```
code ends
```

```
end start
```



```
C:\>  
C:\>  
C:\>  
C:\>  
Welcome to masm!  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>pro80  
C:\>_
```

2.解决除法溢出

(1) 除数：有 8 位和 16 位两种，在一个 reg 或者内存单元中

(2) 被除数：默认放在 AX 或者 DX 和 AX 中，如果除数为 8 位，被除数为 16 位，默认放在 AX 中；如果除数为 16 位，被除数为 32 位，在 DX 和 AX 中存放，DX 存高六位，AX 存低六位。

(3) 结果：如果除数为 8 位。则 AL 储存除法操作的商，AH 储存除法操作的余数；如果除数为 16 位，则 AX 存储除法操作的商，DX 存储除法操作的余数

1000/1 溢出, al 无法存下 1000

```
assume cs:code
code segment
start:  mov bh, 1
        mov ax, 1000
        div bh

        mov ax, 4c00h
        int 21h
code ends
end start
```

```
076A:0000 B701          MOV     BH,01
-t
AX=FFFF BX=0100 CX=000C DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0002  NU UP EI PL NZ NA PO NC
076A:0002 B8E803          MOV     AX,03EB
-t
AX=03EB BX=0100 CX=000C DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0005  NU UP EI PL NZ NA PO NC
076A:0005 F6F7          DIV     BH
-t
AX=03EB BX=0100 CX=000C DX=0000 SP=FFFA BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=F000 IP=1060  NU UP DI PL NZ NA PO NC
F000:1060 FE3B          ???    [BX+SI]          DS:0100=B7
-
```

11000H/1, ax 存放不下商

```
assume cs:code
code segment
start:  mov ax, 1000h
        mov dx, 1
        mov bx, 1
        div bx

        mov ax, 4c00h
        int 21h
code ends
end start
```

```

-t
AX=1000 BX=0000 CX=0010 DX=0001 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0006  NU UP EI PL NZ NA PO NC
076A:0006 BB0100          MDU      BX,0001
-t
AX=1000 BX=0001 CX=0010 DX=0001 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0009  NU UP EI PL NZ NA PO NC
076A:0009 F7F3          DIU      BX
-t
AX=1000 BX=0001 CX=0010 DX=0001 SP=FFFA BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=F000 IP=1060  NU UP DI PL NZ NA PO NC
F000:1060 FE3B          ???      [BX+SI]          DS:0001=20

```

解决方法:

实际上, 我们解决的方法是将除法拆分成了两个 16 位除数, 32 位被除数的除法

例如: 1000000/10 (F4240H/0AH)

我们将这个除法拆分成: 高 16 位: 000FH/0AH 低 16 位: 4240H/0AH 两个除法式子

然后将第一个的结果放到 dx,第二个的结果放到 ax,余数放到 cx

这里唯一不同的就是, 我们高 16 位计算的时候, 我们相当于把尾数的 0 省略了, 所以真正结果需要乘上 $16^{\text{尾数 } 0 \text{ 的个数}}$, 上面例子就是 $16^4=65536$ 。

通过理解 $12341000/10 = \text{int}(1234/10) * 10^4 + (\text{rem}(1234/10) * 10^4 + 1000) / 10$ 给的公式也就很好理解了

这里还需要知道, 溢出的基本概念: 超过寄存器表示范围的值被舍弃。

```

assume cs:code
stack segment
    dw 8 dup (0)
stack ends
code segment
start:    mov ax,stack
          mov ss,ax
          mov sp,16
          mov ax,4240h
          mov dx,00Fh
          mov cx,0ah
          call divdw

          mov ax,4c00h
          int 21h

```

```

divdw:  push ax
        ;高 16 位计算
        mov ax, dx
        mov dx, 0
        div cx
        mov bx, ax
        ;低 16 位计算
        pop ax
        ;高 16 位计算的余数作为了低 16 位计算的高位，对应公式的
        rem(H/N)*65536
        div cx
        ;低 16 位
        mov cx, dx
        ;余数
        mov dx, bx
        ret

code ends
end start

```

The screenshot displays a debugger window with assembly code and a calculator. The assembly code shows the execution of the 'divdw' procedure. The calculator shows the value 100,000 in hexadecimal (186A0), decimal (100,000), octal (303240), and binary (0001100001101010000). The calculator interface includes buttons for logical operations (Lsh, Rsh, Or, Xor, Not, And) and arithmetic operations (Mod, CE, C, <, ÷).

3.数值显示

实际上，方法就是，数值除 10，取余数，就得到数值的每一位，再观察得到，数字 $xH(x=0,1\dots9)$ 对应，十进制数字 $30H+xH$



```

assume cs:code
data segment
    dw 123,12666,1,8,3,38
data ends
code segment

```

```

start:    mov ax, data
          mov ds, ax
          mov ax, 2000h
          mov es, ax

          mov si, 0;表示数据段中的偏移地址
          mov di, 0;将 16 进制数存储为 10 进制字符的偏移地址
          mov cx, 6

          ;循环取出数据段中的数据
s:        mov ax, ds:[si]
          call show_str
          add si, 2
          loop s

          mov ax, 4c00h
          int 21h

show_str: push cx
          mov bp, 0;bp 记录入栈次数

          ;方法: 将该数除以 10, 得到的余数就是该数的一位数
          ;注意, 我们判断数字转换结束的条件是商为 0, 但是 loop 是先
          减 cx, 再判断, 所以需要 inc cx
          ;处理一个 16 进制数, 将每一位转换为 10 进制字符, 压入栈中
          ;压栈原因: 因为得到的数据是逆序的, 所以需要入栈, 再出栈,
          得到顺序的数
s0:       mov dx, 0
          mov bx, 10
          div bx
          mov cx, ax
          inc cx
          add dx, 30h;数字转字符
          push dx;转换的数据入栈
          inc bp
          loop s0

          ;将栈中的数据取出, 存储到指定位置
          mov cx, bp
          mov bx, 0
store_str: pop dx
          mov byte ptr es:[di], dl
          inc di
          loop store_str

```

```

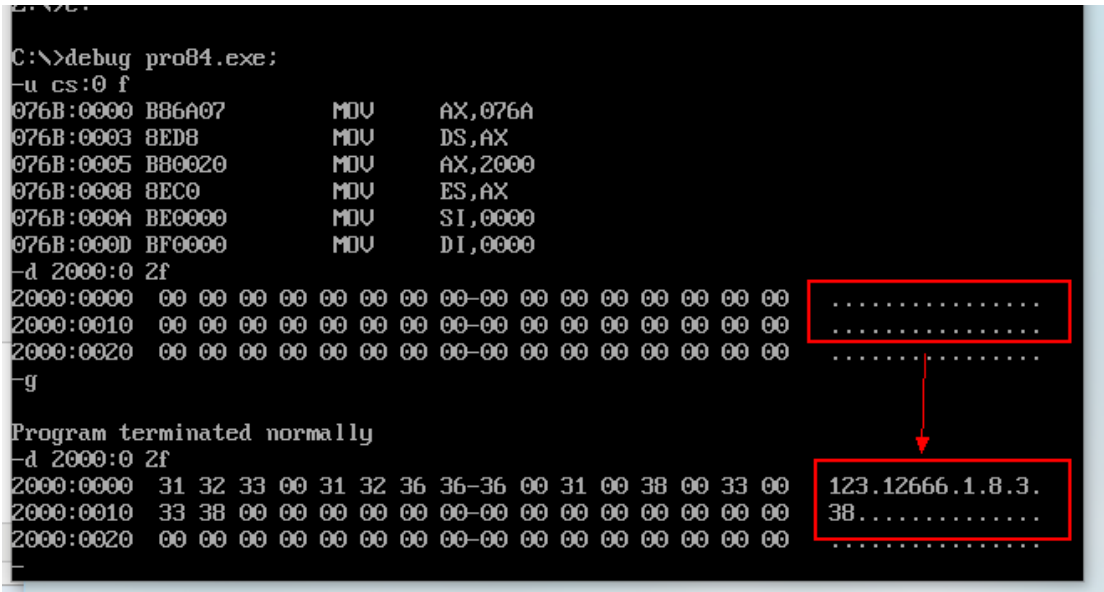
pop cx
inc di;这里 di 表示每个 10 进制字符之间有个间隙，方便查看
ret

```

```

code ends
end start

```



让显示这些数值，实际上就是上个程序，储存位置变成了显示缓冲区

```

assume cs:code
data segment
    dw 123,12666,1,8,3,38
data ends
code segment
start:  mov ax,data
        mov ds,ax
        mov ax,0b800h
        mov es,ax

        mov si,0;表示数据段中的偏移地址
        mov di,0;将 16 进制数存储为 10 进制字符的偏移地址
        mov cx,6

        ;循环取出数据段中的数据
s:      mov ax,ds:[si]
        call show_str

```

```

    add si, 2
    loop s

    mov ax, 4c00h
    int 21h

show_str:    push cx
            mov bp, 0

            ;方法：将该数除以 10，得到的余数就是该数的一位数
            ;注意，我们判断数字转换结束的条件是商为 0，但是 loop 是先
            减 cx，再判断，所以需要 inc cx
            ;处理一个 16 进制数，将每一位转换为 10 进制字符，压入栈中
            ;压栈原因：因为得到的数据是逆序的，所以需要入栈，再出栈，
            得到顺序的数
s0:         mov dx, 0
            mov bx, 10
            div bx
            mov cx, ax
            inc cx
            add dx, 30h; 数字转字符
            push dx; 转换的数据入栈
            inc bp
            loop s0

            ;将栈中的数据取出，存储到显示缓冲区
            mov cx, bp
            mov bx, 0
store_str:  pop dx
            mov byte ptr es:[di+500h+4], dl; 低位数据，500h 表示显示器
            第 8 行，4 表示第 3 列
            mov byte ptr es:[di+500h+4+1], 2; 高位属性，2 就是
            00000010，绿色
            add di, 2
            loop store_str

            pop cx
            add di, 2; 方便查看
            ret

code ends
end start

```


-
1. 获取数据段数据，保存到指定寄存器，利用第一个子程序功能存入显示缓存区。
 2. 平均工资，先获取数据段中的总工资和人数，计算出平均工资，并存入指定寄存器，最后调用第一个子程序存入显示缓冲区。

```
assume cs:code,ds:data,es:table
```

```
data segment
```

```
db '1975','1976','1977','1978','1979','1980','1981','1982','1983'
```

```
db '1984','1985','1986','1987','1988','1989','1990','1991','1992'
```

```
db '1993','1994','1995'
```

```
;以上是表示 21 年的字符串 4 * 21 = 84
```

```
dd 16,22,382,1356,2390,8000,16000,24486,50065,97479,140417,197514
```

```
dd 345980,590827,803530,1183000,1843000,2759000,3753000,4649000,5937000
```

```
;以上是表示 21 年公司总收入的 dword 型数据 4 * 21 = 84
```

```
dw 3,7,9,13,28,38,130,220,476,778,1001,1442,2258,2793,4037,5635,8226
```

```
dw 11542,14430,15257,17800
```

```
;以上是表示 21 年公司雇员人数的 21 个 word 型数据 2 * 21 = 42
```

```
data ends
```

```
table segment
```

```
db 21 dup ('year summ ne ?? '); 'year summ ne ??' 刚好 16 个字节
```

```
table ends
```

```
stack segment
```

```
dw 8 dup (0)
```

```
stack ends
```

```
code segment
```

```
start: mov ax,data
```

```
mov ds,ax
```

```
mov ax,0b800h
```

```
mov es,ax
```

```
mov ax,stack
```

```
mov ss,ax
```

```
mov sp,16
```

```
mov cx,21 ;外层循环每一年的情况
```

```
mov bx,0a0h ;表示第几年的情况
```

```
mov di,0 ;存储年份地址
```

```
mov bp,0 ;1.获取到 data 段中雇员数数据 2.保存 bx 的值
```

```

s0:  push cx
      mov cx,4
      mov si,0      ;表示这一年中的第几个

      ;年份
s:    mov al,ds:[di];一位一位保存字符
      mov es:[bx+si],al
      mov byte ptr es:[bx+si+1],2

      add si,2
      inc di
      loop s
      add si,2

      ;收入
      mov dx,ds:[di+52h]
      mov ax,ds:[di+50h]
      mov si,24;指定列的位置, 13 列
      call show_str

      ;雇员数
      mov ax,ds:[bp+0a8h]
      mov dx,0
      mov si,44;指定列的位置, 23 列
      call show_str

      ;人均收入
      mov dx,ds:[di+52h]
      mov ax,ds:[di+50h]
      push bx
      mov bx,ds:[bp+0a8h]
      div bx
      mov dx,0
      pop bx;顺序不能变
      mov si,64;指定列的位置, 33 列
      call show_str

      pop cx
      add bp,2
      add bx,0a0h
      loop s0

      mov ax,4c00h
      int 21h

```

```

;调用 divdw 子程序(除 10), 获取余数 (数据段数据的 16 进制形式)
;将余数转换为 10 进制字符, 入栈
;循环上面操作
;判断结束条件: 因为被除数为 dword 形式, 需要先判断 dx(高 16 位)是否为 0, 再判断 ax(低
16 位)是否 0
show_str:push bp
        push bx

        mov bp,0

s1:
    mov cx,0ah
    call divdw
    add cx,30h;数字转字符
    push cx;转换的数据入栈
    mov cx,dx
    inc cx
    inc bp
    loop s1

    mov cx,ax
    inc cx
    loop s1

;获取表示行的数值 bx, 在栈中的位置的偏移地址
mov cx,bp;在 bx 上入了多少次栈
mov bx,sp;现在的栈顶
s3:  add bx,2;找到 bx 的位置
    loop s3
    mov cx,bp

    mov bx,ss:[bx]
store_str:
    pop dx
    mov byte ptr es:[bx+si],dl;低位数据, 500h 表示显示器第 8 行, 4 表示第 3 列
    mov byte ptr es:[bx+si+1],2;高位属性, 2 就是 00000010, 绿色
    add si,2
    loop store_str

    pop bx
    pop bp
    add si,2
    ret

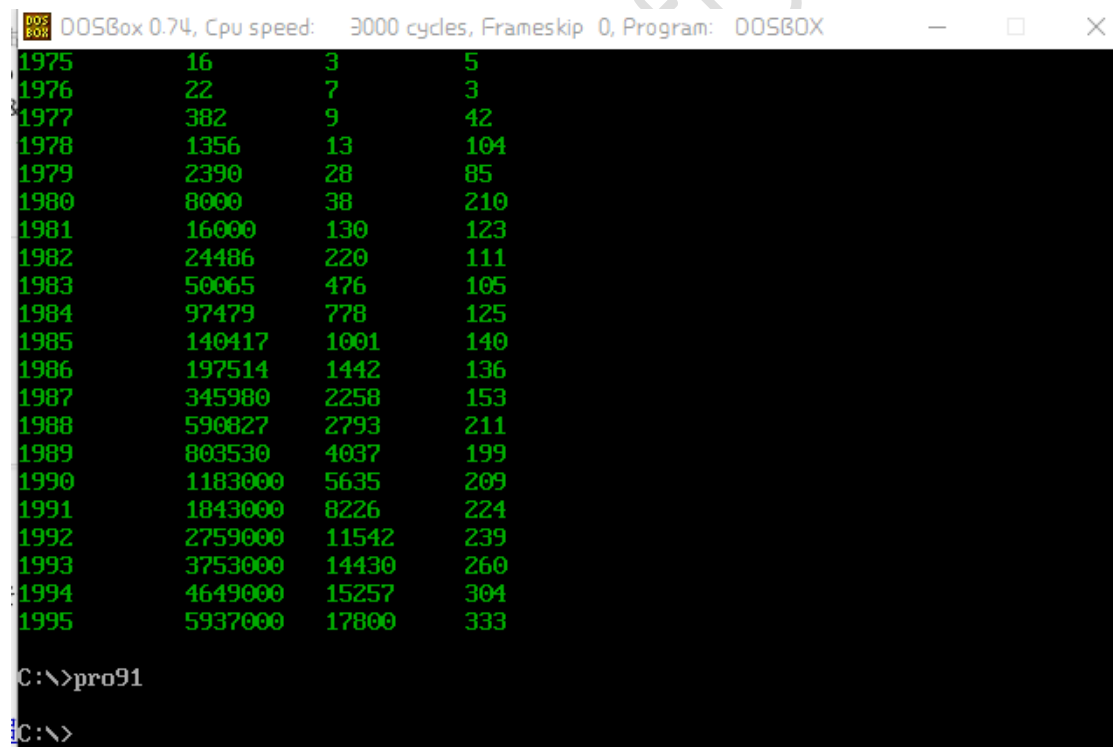
```

```

divdw: push ax
      ;高 16 位计算
      mov ax, dx
      mov dx, 0
      div cx
      mov bx, ax
      ;低 16 位计算
      pop ax
      div cx
      ;低 16 位
      mov cx, dx
      ;余数
      mov dx, bx
      ret

code ends
end start

```



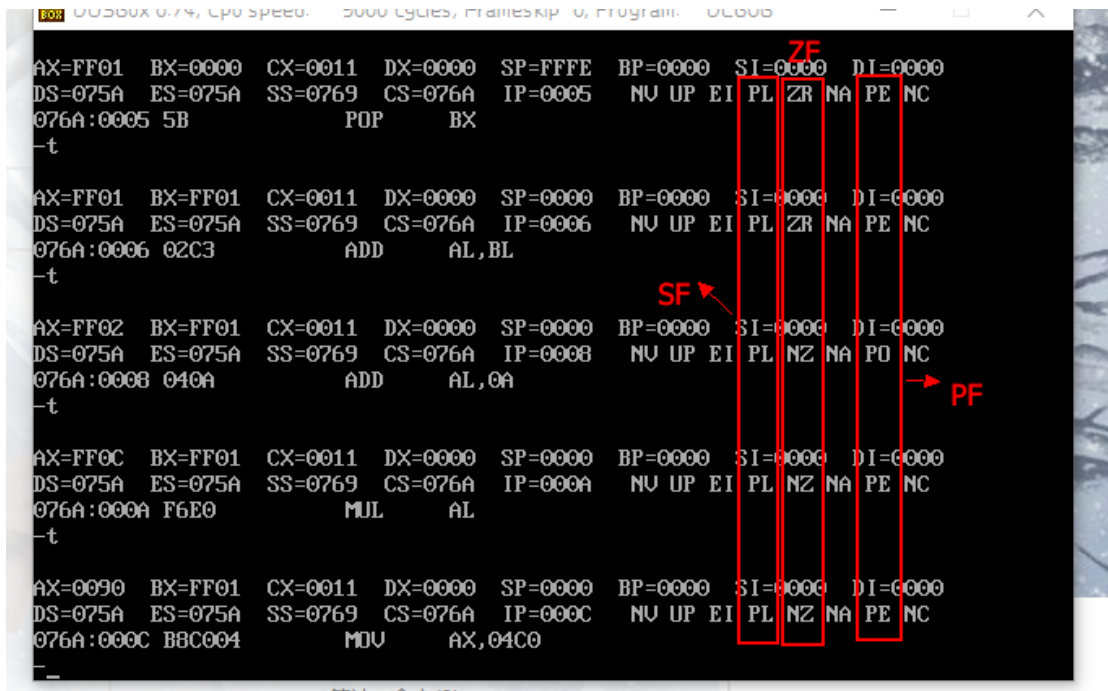
第十一章 标志寄存器

检测点 11.1

| ZF | PF | SF |
|----|----|----|
| 1 | 1 | 0 |
| 1 | 1 | 0 |
| 1 | 1 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |

```
assume cs:code
code segment
start:  sub al,al
        mov al,1
        push ax
        pop bx
        add al,b1
        add al,10
        mul al

        mov ax,4c0h
        int 21h
code ends
end start
```



检测点 11.2

| 操作 | CF | OF | SF | ZF | PF | 原因 |
|------------|----|----|----|----|----|---|
| sub,al,al | 0 | 0 | 0 | 1 | 1 | 无符号 al=0 有符号 al=0 |
| mov al,10H | 0 | 0 | 0 | 1 | 1 | mov 指令不改变标志寄存器值 无符号 al=10H=0001 0000b 有符号 al=10H=0001 0000b |
| add al,90H | 0 | 0 | 1 | 0 | 1 | 无符号 al=0AH 有符号 al=10H + (90H)补=异号相加 |
| mov al,80H | 0 | 0 | 1 | 0 | 1 | mov 指令不改变标志寄存器值 无符号 al=80H 有符号 al=-128 |

| | | | | | | |
|-------------|---|---|---|---|---|---|
| add al,80H | 1 | 1 | 0 | 1 | 1 | 无符号= 100H=0001 0000 0000=0 有符号=-128+(-128)=-256<-128 |
| mov al,0FCH | 1 | 1 | 0 | 1 | 1 | mov 指令不改变标志寄存器值 无符号:al=0FCH 有符号:al=-4 |
| add al,05H | 1 | 0 | 0 | 0 | 0 | 无符号:al=101H 有符号:-4+5=1<127 |
| mov al,7DH | 1 | 0 | 0 | 0 | 0 | mov 指令不改变标志寄存器值 无符号:al=7DH 有符号:125 |
| add al,0BH | 0 | 1 | 1 | 0 | 1 | 无符号:al=88H 有符号:al=125+11=136>127 |

检测点 11.3

(1)

jb s0

ja s0

(2)

jna s0

jnb s0

检测点 11.4

对 flag 寄存器操作

ax=45h

```
mov ax, 0
push ax
popf;将标志寄存器置
mov ax, 0fff0h
add ax, 0010h
;ax 实际结果为 0, 应得结果: -16+16=0
;flag 寄存器值: 0000 0000 0100 0101=69
pushf;将标志寄存器值入栈
pop ax;将标志寄存器值出栈给 ax
and al, 1100 0101b;0100 0101
and ah, 0000 1000b;0000 0000
;所以 ax=0000 0000 0100 0101=45h
```

实验 11 编写子程序

```
assume cs:codesg
datasg segment
    db "Beginner's All-purpose Symbolic Instruction Code.", 0
datasg ends
codesg segment
start:
    mov ax, datasg
    mov ds, ax
    mov si, 0
    call letterc

    mov ax, 4c00h
    int 21h
letterc:
    cmp byte ptr [si], 97
    jb next
    cmp byte ptr [si], 122
    ja next
    and byte ptr [si], 11011111b
next:
```

```

    mov cl,[si]
    mov ch,0
    jcxz ok
    inc si
    jmp short letterc
ok:
    ret
codesg ends
end start

```

```

Z:\>c:
C:\>debug pro99.exe;
-d 076a:0 2f
076A:0000 42 65 67 69 6E 6E 65 72-27 73 20 41 6C 6C 2D 70
076A:0010 75 72 70 6F 73 65 20 53-79 6D 62 6F 6C 69 63 20
076A:0020 49 6E 73 74 72 75 63 74-69 6F 6E 20 43 6F 64 65
-g b
AX=076A BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=0031 DI=0000
DS=076A ES=075A SS=0769 CS=076E IP=000B NU UP EI NG NZ AC PE CY
076E:000B B8004C MDU AX,4C00
-d ds:0 2f
076A:0000 42 45 47 49 4E 4E 45 52-27 53 20 41 4C 4C 2D 50
076A:0010 55 52 50 4F 53 45 20 53-59 4D 42 4F 4C 49 43 20
076A:0020 49 4E 53 54 52 55 43 54-49 4F 4E 20 43 4F 44 45

```

Beginner's All-purpose Symbolic Instruction Code

BEGINNER'S ALL-PURPOSE SYMBOLIC INSTRUCTION CODE

第十二章 内中断的产生

检测点 12.1

- (1) 70:018B
- (2) 4N, 4N+2

实验 12 编写 0 号中断的处理程序

```

assume cs:code
code segment
start:
    mov ax,cs
    mov ds,ax
    mov si,offset do0
    mov ax,0

```

```

mov es, ax
mov di, 200h
mov cx, offset do0end - offset do0
cld
rep movsb

mov word ptr es:[0*4+2], 0
mov word ptr es:[0*4], 200h

mov ax, 4c00h
int 21h

do0:
jmp short do0start
db 'Biu OverFlow!'

do0start:
mov ax, 0b800h
mov es, ax
mov ax, cs
mov ds, ax
mov si, 202h
mov di, 12*160+36*2
mov cx, 13
s:
mov al, ds:[si]
mov es:[di], al
mov byte ptr es:[di+1], 4
add di, 2
inc si
loop s

mov ax, 4c00h
int 21h

do0end:nop

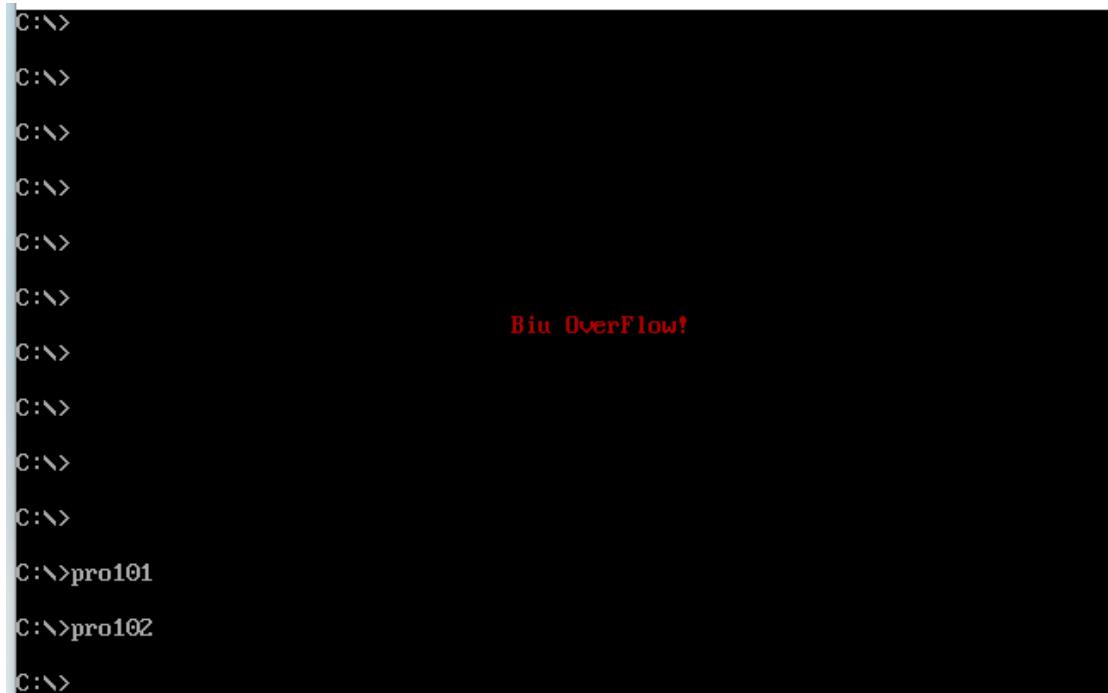
code ends
end start

```

用了个测试程序

```
assume cs:code
```

```
code segment
start:
    int 0
code ends
end start
```



A screenshot of a DOS command prompt window. The background is black, and the text is white. The prompt shows several instances of 'C:\>' followed by the execution of 'pro101' and 'pro102'. A red error message 'Biu Overflow!' is displayed in the center of the screen.

第十三章 int 指令

检测点 13.1

(1) 这道我认为主要是判断

```
add [bp+2], bx
```

这里是修改 IP 地址，即向前跳转距离，也即 bx 的范围。bx 为十六位寄存器，范围在-32768~32767，即最大转移位移为 32768

(2)

测试程序：

```
assume cs:code
data segment
    db 'conversation',0
```

```

data ends
code segment
start:
    mov ax, data
    mov ds, ax
    mov si, 0
    mov ax, 0b800h
    mov es, ax
    mov di, 12*160
    mov bx, offset flag-offset flagend
flag:
    cmp byte ptr ds:[si], 0
    je ok
    mov al, ds:[si]
    mov byte ptr es:[di], al
    mov byte ptr es:[di+1], 2
    inc si
    add di, 2
    int 7ch
flagend:nop
ok:
    mov ax, 4c00h
    int 21h
code ends
end start

```

中断例程:

```

assume cs:code
code segment
start:
    ;将程序移入指定空闲区域
    mov ax, cs
    mov ds, ax
    mov si, offset do7c
    mov ax, 0
    mov es, ax
    mov di, 200h
    mov cx, offset do7cend-offset do7c
    cld
    rep movsb
    ;设置中断向量
    mov ax, 0

```

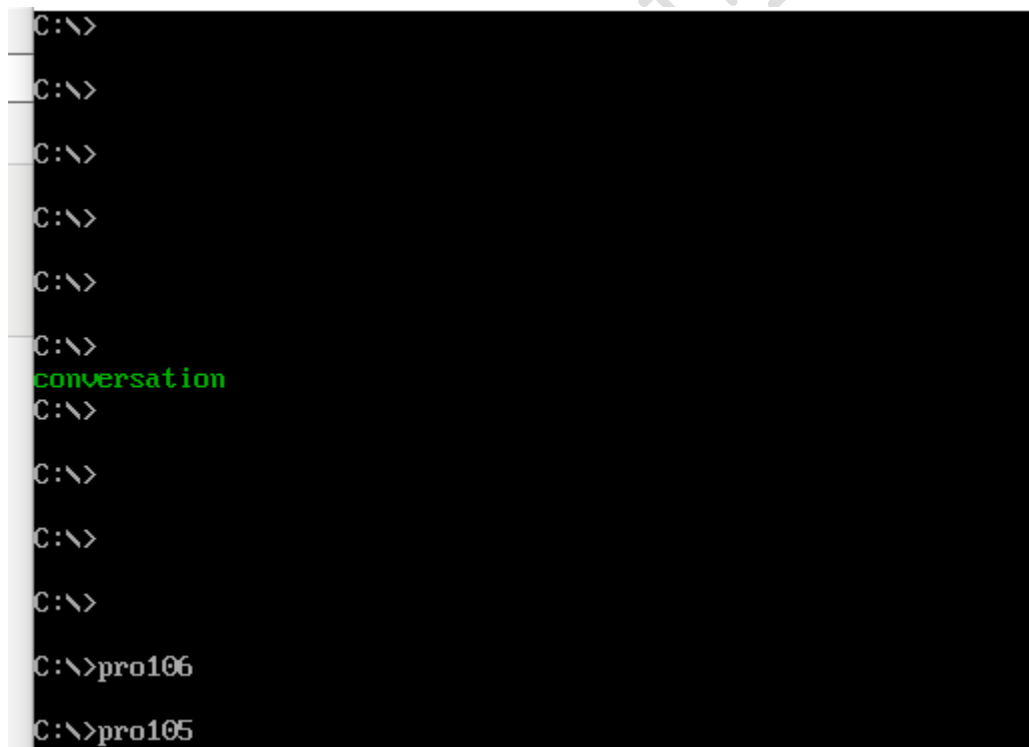


```
mov es, ax
mov word ptr es:[7ch*4+2], 0
mov word ptr es:[7ch*4], 200h

mov ax, 4c00h
int 21h

do7c:
;设置 IP 位置
push bp
mov bp, sp
add ss:[bp+2], bx
pop bp
iret
do7cend:nop

code ends
end start
```



```
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
conversation
C:\>
C:\>
C:\>
C:\>pro106
C:\>pro105
```

检测点 13.2

(1) 错误，无法改变。

(2) 错误，硬件应该是 BIOS 的中断例程

实验 13 编写，应用中断例程

(1) 编写并安装 int 7ch 中断例程，功能为显示一个用 0 结束的字符串，中断例程安装在 0:200 处

参数：(dh)=行号，(dl)=列号，(cl)=颜色，ds:si 指向字符串首地址

测试程序

```
assume cs:code
data segment
    db 'Welcome to masm!BIU',0
data ends
code segment
start:
    mov dh,12
    mov dl,30
    mov cl,2
    mov ax,data
    mov ds,ax
    mov si,0
    int 7ch

    mov ax,4c00h
    int 21h
code ends
end start
```

中断例程

```
assume cs:code
code segment
start:
    ;将程序写入 0:200h
    mov ax,cs
    mov ds,ax
    mov si,offset func
    mov ax,0
    mov es,ax
```

```
mov di, 200h
mov cx, offset funcend - offset func; 程序长度
cld
rep movsb
```

```
; 程序入口写入中断向量表中
mov ax, 0
mov es, ax
mov word ptr es: [7ch*4+2], 0
mov word ptr es: [7ch*4], 200h
```

```
mov ax, 4c00h
int 21h
```

func:

```
push ax
push di
push es
```

```
; 显示缓冲区地址
```

```
mov ax, 0b800h
mov es, ax
```

```
; 将已知条件转换为正确的显存地址
```

```
mov al, 160
mul dh
mov di, ax
```

```
dec dl
mov dh, 0
add dx, dx
add di, dx
```

show:

```
; 判断是否遇到末尾的 0
cmp byte ptr ds: [si], 0
je ok
; 将显示信息写入显示缓冲区
mov al, ds: [si]
mov byte ptr es: [di], al
mov es: [di+1], cl
inc si
add di, 2
jmp show
```

ok:

```

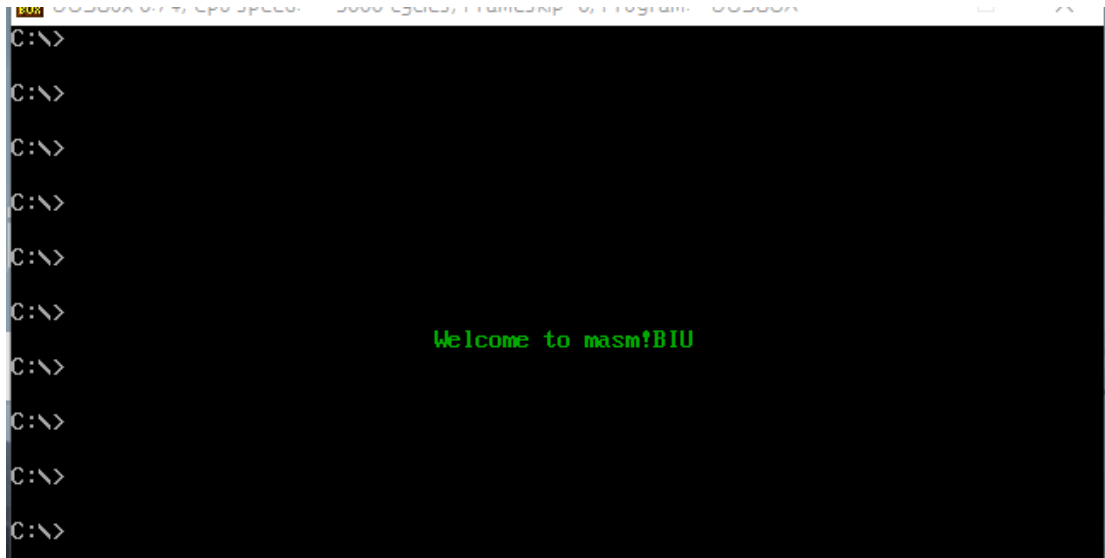
;结束
pop es
pop di
pop ax
iret

```

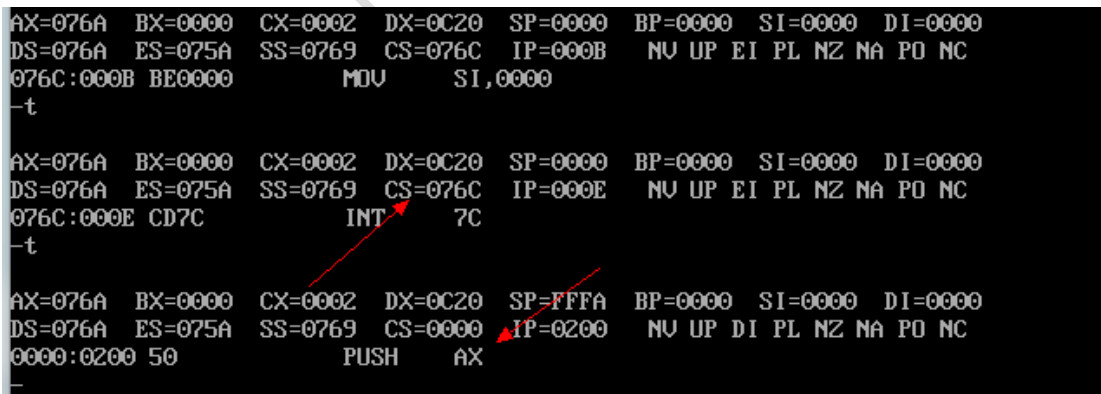
```

funcend:nop
code ends
end start

```



CS:IP 和 SS:SP 在 int 和 iret 前后变化



入栈标志寄存器，CS，IP。SP=FFFAH

SS 为当前段地址，CS:IP 为空闲区域地址

```

AX=0755 BX=0000 CX=0002 DX=003E SP=FFF8 BP=0000 SI=0013 DI=0000
DS=076A ES=075A SS=0769 CS=0000 IP=022C NU UP DI PL ZR NA PE NC
0000:022C 5B POP AX
-t
AX=076A BX=0000 CX=0002 DX=003E SP=FFFA BP=0000 SI=0013 DI=0000
DS=076A ES=075A SS=0769 CS=0000 IP=022D NU UP DI PL ZR NA PE NC
0000:022D CF IRET
-t
AX=076A BX=0000 CX=0002 DX=003E SP=0000 BP=0000 SI=0013 DI=0000
DS=076A ES=075A SS=0769 CS=076C IP=0010 NU UP EI PL NZ NA PO NC
076C:0010 B8004C MOV AX,4C00

```

(2) 在屏幕中间显示 80 个'!'

测试程序

```

assume cs:code
code segment
start:
    mov ax,0b800h
    mov es,ax
    mov di,160*12
    mov bx,offset s-offset se
    mov cx,80
s:
    mov byte ptr es:[di],','
    add di,2
    int 7ch
se:
    nop
    mov ax,4c00h
    int 21h
code ends
end start

```

中断程序

```

assume cs:code
code segment
start:
    ;将程序写入 0:200h
    mov ax,cs
    mov ds,ax
    mov si,offset func
    mov ax,0

```

```

    db 'Good,better,best,',','$'
s2:
    db 'Never let it rest,',','$'
s3:
    db 'Till good is better,',','$'
s4:
    db 'And better,best.',','$'
s:
    dw offset s1,offset s2,offset s3,offset s4
row:
    db 2,4,6,8

start:
    mov ax,cs
    mov ds,ax
    mov bx,offset s
    mov si,offset row
    mov cx,4
ok:
    mov bh,0
    mov dh,ds:[si]
    mov dl,0
    mov ah,2
    int 10h

    mov dx,ds:[bx]
    mov ah,9
    int 21h
    inc si
    add bx,2
    loop ok
    mov ax,4c00h
    int 21h
code ends
end start

```

```
C:\>
Good,better,best,
C:\>
Never let it rest,
C:\>
Till good is better,
C:\>
And better,best.
C:\>

C:\>

C:\>

C:\>

C:\>
```

第十四章 端口

检测点 14.2

```
assume cs:code
code segment
start:
    mov ax,66
    mov bx,ax
    shl ax,1
    mov cl,3
    shl bx,cl
    add ax,bx

    mov ax,4c00h
    int 21h
code ends
end start
```



```

AX=00B4 BX=0042 CX=0003 DX=0000 SP=0000 BP=0000
DS=075A ES=075A SS=0769 CS=076A IP=0009 NU UP
076A:0009 D3E3 SHL BX,CL
-t
AX=00B4 BX=0210 CX=0003 DX=0000 SP=0000 BP=0000
DS=075A ES=075A SS=0769 CS=076A IP=000B NU UP
076A:000B 03C3 ADD AX,BX
-t
AX=0294 BX=0210 CX=0003 DX=0000 SP=0000 BP=0000
DS=075A ES=075A SS=0769 CS=076A IP=000D NU UP
076A:000D B804C MOV AX,4C00

```

Programmer

| | |
|-----|----------------|
| HEX | 294 |
| DEC | 660 |
| OCT | 1 224 |
| BIN | 0010 1001 0100 |

实验 14 访问 CMOS RAM

```

assume cs:code
data segment
    db 9,8,7,4,2,0
data ends
code segment
start:
    mov ax,data
    mov ds,ax
    mov bx,0b800h
    mov es,bx
    mov di,160*12+33*2
    mov si,0
    mov cx,6

s:
    call show

    cmp byte ptr ds:[si],7
    jna s1;不大于 7 就跳转
    ;年,月
    mov byte ptr es:[di], '/'
    jmp stop

s1:
    jb s2;小于 7 就跳转
    ;日
    mov byte ptr es:[di], ' '
    jmp stop

s2:
    cmp byte ptr ds:[si],0

```

```

    je stop;等于 0 就跳转
    ;时, 分
    mov byte ptr es:[di],':'

stop:
    inc si
    add di,2
    loop s
    mov ax,4c00h
    int 21h

show:
    push cx
    mov al,ds:[si]
    out 70h,al
    in al,71h

    ;获取数据
    mov ah,al
    mov cl,4
    shr ah,cl
    and al,00001111b

    ;十进制转 ASCII 码
    add ah,30h
    add al,30h

    ;写入显示缓冲区
    mov es:[di],ah
    mov byte ptr es:[di+2],al
    add di,4
    pop cx
    ret

code ends
end start

```

```
mov ax, 0
mov es, ax
push ds:[0]
pop es:[9*4]
push ds:[2]
pop es:[9*4+2]
```

在执行指令期间可能发生键盘中断，引发错误的地址执行，也就是 IF 寄存器可能为 1，所以只要我们在执行这两段指令之间，令 IF=0，即可保证程序正确。

```
cli
mov word ptr es:[9*4], offset int9
mov es:[9*4+2], cs
sti
```

```
cli
mov ax, 0
mov es, ax
push ds:[0]
pop es:[9*4]
push ds:[2]
pop es:[9*4+2]
sti
```

实验 15 安装新的 int 9 中断例程

```
assume cs:code
stack segment
    db 128 dup (0)
stack ends
code segment
start:
    mov ax, stack
    mov ss, ax
    mov sp, 128

    ;将中断例程写到 0:204h
    push cs
    pop ds
```

```

mov ax, 0
mov es, ax

mov cx, offset int9end - offset int9
mov si, offset int9
mov di, 204h
cld
rep movsb

;将原 int 9 中断例程入口地址保存到 0:200h
push es:[9*4]
pop es:[200h]
push es:[9*4+2]
pop es:[202h]

;将新的中断例程入口地址写入中断向量表
cli
mov word ptr es:[9*4], 204h
mov word ptr es:[9*4+2], 0
sti

mov ax, 4c00h
int 21h

int9:
push ax
push bx
push cx
push es
;从端口读取键盘数据
in al, 60h

;调用原来的 int 9 中断例程, 此时 CS=0
pushf
call dword ptr cs:[200h]

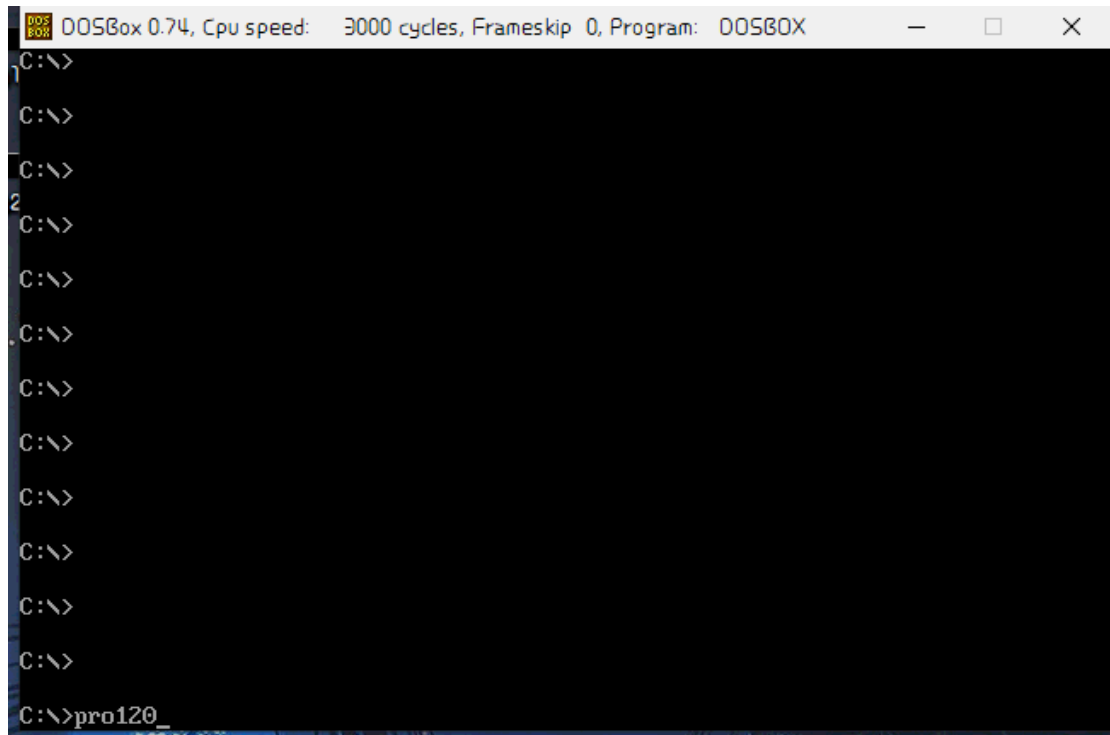
;比较是否收到 A 键值
cmp al, 9eh
jne int9ret

;显示一屏幕 A
mov ax, 0b800h
mov es, ax

```

```
    mov bx, 0
    mov cx, 2000
s:
    mov byte ptr es:[bx], 'A'
    add bx, 2
    loop s
int9ret:
    pop es
    pop cx
    pop bx
    pop ax
    iret

int9end:nop
code ends
end start
```



第十六章 直接定址表

检测点 16.1

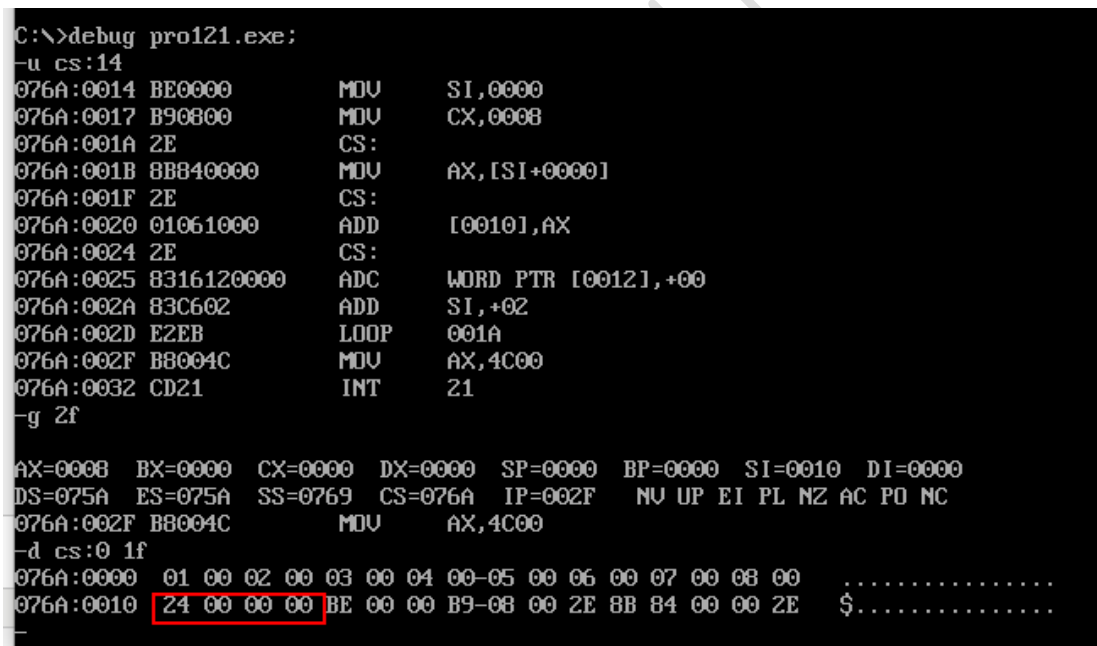
```
assume cs:code
```

```

code segment
    a    dw 1,2,3,4,5,6,7,8
    b    dd 0
start:
    mov si,0
    mov cx,8
s:
    mov ax,a[si]
    add word ptr b[0],ax
    adc word ptr b[2],0
    add si,2
    loop s

    mov ax,4c00h
    int 21h
code ends
end start

```



检测点 16.2

```
mov ax,data
```

```
mov ds,ax
```

实验 16 编写包含多个功能子程序的中断例程。

在这道题中，因为 table...最终会被范围为 cs:N[si]这种形式存储，当我们转移程序到 0:200h 处时，CS 和 IP 改变，但是程序中 table 处没有改变，所以会引发定址错误。

测试程序



```
assume cs:code
code segment
start:
    mov ah, 2
    mov al, 2
    int 7ch

    mov ax, 4c00h
    int 21h
code ends
end start
```



中断例程



```
assume cs:code
code segment
start:
    ;储存程序到 0:200h
    mov ax, 0
    mov es, ax
    mov ax, cs
    mov ds, ax
    mov cx, offset setscreenend - offset setscreen
    mov di, 200h
    mov si, offset setscreen
    cld
    rep movsb

    ;程序入口写入向量表
    mov word ptr es:[7ch*4], 200h
    mov word ptr es:[7ch*4+2], 0

    mov ax, 4c00h
```

```
int 21h

org 200h
setscreen:
    jmp short set
    table dw sub1, sub2, sub3, sub4
set:
    push bx
    ;判断选择的功能是否越界
    cmp ah, 4
    ja sret
    mov bl, ah
    mov bh, 0
    add bx, bx

    call word ptr table[bx]

sret:
    pop bx
    iret

sub1:
    push es
    push bx
    push cx

    mov bx, 0b800h
    mov es, bx
    mov bx, 0
    mov cx, 2000
sub1s:
    mov byte ptr es:[bx], ' '
    add bx, 2
    loop sub1s

    pop cx
    pop bx
    pop es
    iret

sub2:
    push bx
    push es
    push cx
```

```

    mov bx, 0b800h
    mov es, bx
    mov bx, 1
    mov cx, 2000
sub2s:
    ;设置指定颜色
    and byte ptr es:[bx], 11111000b
    or es:[bx], al
    add bx, 2
    loop sub2s

    pop cx
    pop es
    pop bx
    iret

sub3:
    push bx
    push cx
    push es

    ;因为 al 的范围在 0~7，所以设置背景色时，需要右移 4 位
    mov cl, 4
    shr al, cl
    mov bx, 0b800h
    mov es, bx
    mov cx, 2000
    mov bx, 1
sub3s:
    and byte ptr es:[bx], 10001111b
    or es:[bx], al
    add bx, 2
    loop sub3s

    pop es
    pop cx
    pop bx
    iret

sub4:
    push bx
    push es
    push di

```

```
    push si
    push cx

    mov bx, 0b800h
    mov es, bx
    mov ds, bx
    mov si, 160
    mov di, 0
    mov cx, 24
sub4s:
    push cx
    mov cx, 160
    cld
    rep movsb
    pop cx
    loop sub4s

    pop cx
    pop si
    pop di
    pop es
    pop bx
    iret

setscreenend:nop
code ends
end start
```



转自百度: <http://t.cn/AijCULrC>

伪指令 **org** 用来规定目标程序存放单元的偏移量。比如, 如果在源程序的第一条指令前用了如下指令:

```
org 200h
```

那么, 汇编程序会把指令指针的 **ip** 的值设成 **200h**, 即目标程序的第一个字节放在 **200h** 处, 后面的 内容则顺序存放, 除非遇上另一个 **org** 语句

第十七章 使用 BIOS 进行键盘输入和磁盘读写

检测点 17.1

错误，如果 int 16 是不可屏蔽中断，那 IF=0，如果 int 16 是可屏蔽中断，且要一直检查键盘缓冲区是否存在数据，所以 IF=1 一直存在。

试验 17 编写包含多个功能子程序的中断例程

测试程序

```
assume cs:code
code segment
start:
    mov ah, 0
    mov dx, 36
    mov bx, 0b800h
    mov es, bx
    mov bx, 160*12+40*2

    int 7ch

    mov ax, 4c00h
    int 21h
code ends
end start
```

中断例程

```
assume cs:code
code segment
start:
    mov ax, 0
    mov es, ax
    mov di, 200h
    mov ax, cs
    mov ds, ax
    mov si, offset func
```

```

mov cx, offset funcend - offset func
cld
rep movsb

mov bx, 0
mov es, bx
mov word ptr es:[7ch*4], 200h
mov word ptr es:[7ch*4+2], 0

mov ax, 4c00h
int 21h

org 200h
func:
jmp short main
table dw func0, func1
main:
;cmp ah, 0
;je func0
;cmp ah, 1
;je func1
mov al, ah
mov ah, 0
mov si, ax
add si, si
call word ptr table[si]

mov ax, 4c00h
int 21h

func0:
push bx

;扇区号
mov ax, dx
mov bl, 18
div bl
inc ah
mov cl, ah

;磁道号
mov ah, 0
mov bl, 80
div bl

```

```
mov ch, ah

;面号
mov dh, al

;驱动器号
mov dl, 0

;读取
mov ah, 2

;读取的扇区数
mov al, 1

pop bx
int 13

ret

func1:
push bx

;扇区号
mov ax, dx
mov bl, 18
div bl
inc ah
mov cl, ah

;磁道号
mov ah, 0
mov bl, 80
div bl
mov ch, ah

;面号
mov dh, al

;驱动器号
mov dl, 0

;写入
mov ah, 3
```

```

;写入的扇区数
mov al,1

pop bx
int 13h

ret

funcend:nop
code ends
end start

```

课程设计 2

```

assume cs:code,ss:stack
stack segment
    db 128 dup (0)
stack ends
code segment
start:
    mov ax,stack
    mov ss,ax
    mov sp,128

    call copy_boot

;设置 CS:IP 为 0:7e00h
    mov ax,0
    push ax
    mov ax,7e00h
    push ax
    retf

    mov ax,4c00h
    int 21h
;org 7e00h
;引导程序
boot:
    jmp boot_begin
func0    db 'Hk_Mayfly----XIUXIUXIU~',0
func1    db '1) reset pc',0
func2    db '2) start system',0

```

```

func3    db '3) clock',0
func4    db '4) set clock',0
;相减得到的是标号的相对位置, +7e00h 得到的绝对位置
func_pos  dw offset func0-offset boot+7e00h
          dw offset func1-offset boot+7e00h
          dw offset func2-offset boot+7e00h
          dw offset func3-offset boot+7e00h
          dw offset func4-offset boot+7e00h
time     db 'YY/MM/DD hh:mm:ss',0
cmos     db 9,8,7,4,2,0
clock1   db 'F1----change the color          ESC----return menu',0
clock2   db 'Please input Date and Time, (YY MM DD hh mm ss):',0
change   db 12 dup (0),0

```

boot_begin:

```

call init_boot
call cls_screen
call show_menu
jmp choose
mov ax,4c00h
int 21h

```

choose:

```

call clear_kb_buffer
;获取我们输入的操作, 跳转到对于函数
mov ah,0
int 16h
cmp al,'1'
je choose_func1
cmp al,'2'
je choose_func2
cmp al,'3'
je choose_func3
cmp al,'4'
je choose_func4

```

```

jmp choose

```

;在题中提到了, 开机后进入到 ffff:0 处执行指令
;那我们也可以把重启理解为, 跳转到 ffff:0 执行指令
;所以我们利用 jmp dword 跳转到 ffff:0 地址, 模拟重启

choose_func1:

```

mov bx,0ffffh
push bx

```

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

```
jmp choose
```

;题中对引导现有的操作系统的描述是调用 int 19, 这里为了方便就直接写成函数了

```
choose_func2:
```

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

```
mov al, 1;扇区数
mov ch, 0
mov cl, 1;扇区
mov dl, 80h
mov dh, 0
mov ah, 2;读取
int 13h
```

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

```
jmp choose
```

;获取时间

```
choose_func3:
```

```
call show_time
```

```
jmp choose
```

```
show_time:
```

```
call init_boot
call cls_screen
;显示按键信息
mov si, offset clock1 - offset boot + 7e00h
mov di, 160 * 14 + 10 * 2;在 14 行 10 列显示
call show_line
```

```
show_time_start:
```

;获取时间信息, 并显示 (将 time 中的未知字符替换为当前时间)

```

call get_time_info
mov di, 160*10+30*2; 屏幕显示的偏移地址
mov si, offset time-offset boot+7e00h; time 标号的偏移地址
call show_line

; 获取键盘缓存区的数据
mov ah, 1
int 16h
; 没有数据就跳回 show_time_start
jz show_time_start
; 判断是否按下 F1
cmp ah, 3bh
je change_color
; 判断是否按下 ESC
cmp ah, 1
je Return_Main
; 有数据，但是是无用的键盘中断，清除
cmp al, 0
jne clear_kb_buffer2
; 返回开始，重复之前的操作，达到刷新时间的效果。
jmp show_time_start

change_color:
call change_color_show
clear_kb_buffer2:
call clear_kb_buffer
jmp show_time_start
Return_Main:
; 返回到开始，重新打印菜单
jmp boot_begin
ret

choose_func4:
call set_time
jmp boot_begin

set_time:
call init_boot
call cls_screen
call clear_stack

; 设置提示信息显示位置
mov di, 160*10+13*2
mov si, offset clock2-offset boot+7e00h

```

```

    call show_line
    ;显示修改后 change 中的内容
    mov di, 160*12+26*2
    mov si, offset change - offset boot + 7e00h
    call show_line

    call get_string

get_string:
    mov si, offset change - offset boot + 07e00H
    mov bx, 0
getstring:
    ;获取键盘输入的时间信息
    mov ah, 0
    int 16h

    ;输入的时间为数字 0~9
    cmp al, '0'
    jb error_input
    cmp al, '9'
    ja error_input
    ;将我们输入的时间字符入栈
    call char_push
    ;不能超过输入的数量
    cmp bx, 12
    ja press_ENTER
    mov di, 160*12+26*2
    call show_line
    jmp getstring
error_input:
    ;判断是不是按下退格或回车键
    cmp ah, 0eh
    je press_BS
    cmp ah, 1ch
    je press_ENTER

    jmp getstring
;按下回车
press_BS:
    call char_pop
    mov di, 160*12+26*2
    call show_line
    jmp getstring
;按下 enter 就退出

```

```

press_ENTER:
    ret

char_push:
    ;只能最多输入 12 个梳子
    cmp bx, 12
    ja char_push_end
    ;将数值移动到对应位置
    mov ds:[si+bx], al
    inc bx;表示我们输入了多少个字符
char_push_end:
    ret

char_pop:
    ;判断是否输入了设置时间的数值，没有就相当于删完了
    cmp bx, 0
    je char_pop_end
    ;否则用星号替换，相当于删除
    dec bx
    mov byte ptr ds:[si+bx], '*'
char_pop_end:
    ret

clear_stack:
    push bx
    push cx

    mov bx, offset change_offset boot+7e00h
    mov cx, 12
cls_stack:
    ;替换 change 段中内容
    mov byte ptr ds:[bx], '*'
    inc bx
    loop cls_stack

    pop cx
    pop bx
    ret

;获取时间
get_time_info:
    ;从 cmos ram 获取年月日，时分秒 6 个数据
    mov cx, 6

```

```

;获取存放单元地址
mov bx,offset cmos - offset boot + 7e00H
;通过替换来显示
mov si,offset time - offset boot + 7e00H
next_point:
push cx
;获取单元号
mov al,ds:[bx]
;向 70h 端口写入要访问的单元地址, 并从 71h 端口读取数据
out 70H,al
in al,71H
;右移 4 位获取十位
mov ah,al
mov cl,4
shr al,cl
and ah,00001111b
;将 BCD 码转换为 ASCII 码
add ax,3030H
;写入 time 中
mov word ptr ds:[si],ax
;下一单元号
inc bx
;每个数据之间距离都是 3
add si,3
pop cx
loop next_point
ret

```

```

;改变颜色
change_color_show:
push bx
push cx

mov cx,2000
mov bx,1
next:
;属性值+1, 改变颜色
add byte ptr es:[bx],1
;当超出字体颜色的数值(0~111h)时, 将数值重置
cmp byte ptr es:[bx],00001000b
jne change_end
;因为背景是黑色, 所以文字颜色就不设置成黑色了
mov byte ptr es:[bx],1
change_end:

```

```

    add bx, 2
    loop next

    pop cx
    pop bx
    ret

clear_kb_buffer:
    ;1号程序,用来检测键盘缓冲区是否有数据
    ;如果有的话ZF!=0,没有,ZF=0
    mov ah, 1
    int 16h
    ;通过ZF判断减缓缓冲区是否有数据,没有就跳出
    jz clear_kb_bf_end
    mov ah, 0
    int 16h
    jmp clear_kb_buffer
clear_kb_bf_end:
    ret

init_boot:
    ;基本设置,注意:程序的直接定址表默认段地址是CS
    ;当程序转移到7c00h时,代码中CS值未发生改变,
    ;所以需要我们指明段地址
    mov bx, 0b800h
    mov es, bx
    mov bx, 0
    mov ds, bx
    ret

;清屏
cls_screen:
    mov bx, 0
    mov cx, 2000
    mov dl, ' '
    mov dh, 2;字体为绿色,不设置的话,在我们显示菜单时,字体和背景颜色
    相同
s:    mov es:[bx], dx
    add bx, 2
    loop s
sret:
    ret

;展示界面

```

```
show_menu:
    ;在 10 行, 30 列显示菜单
    mov di, 160*10+30*2
    ;保存在直接定址表的绝对位置
    mov bx, offset func_pos-offset boot+7e00h
    ;菜单有 5 行
    mov cx, 5
```

```
s1:
    ;这里相当于外循环, 每次一行
    ;获取 func_pos 中每行的保存位置的偏移地址
    mov si, ds:[bx]
    ;调用内循环函数, 输出一行的每个字符
    call show_line
    ;下一行偏移地址
    add bx, 2
    ;下一行显示
    add di, 160
    loop s1
    ret
```

```
show_line:
    push ax
    push di
    push si
```

```
show_line_start:
    ;获取这一行的第 si+1 个字符
    mov al, ds:[si]
    ;判断是否到末尾
    cmp al, 0
    je show_line_end
    ;保存字符到显示缓冲区
    mov es:[di], al
    add di, 2
    inc si
    jmp show_line_start
```

```
show_line_end:
    pop si
    pop di
    pop ax
    ret
```

```
boot_end:nop
```

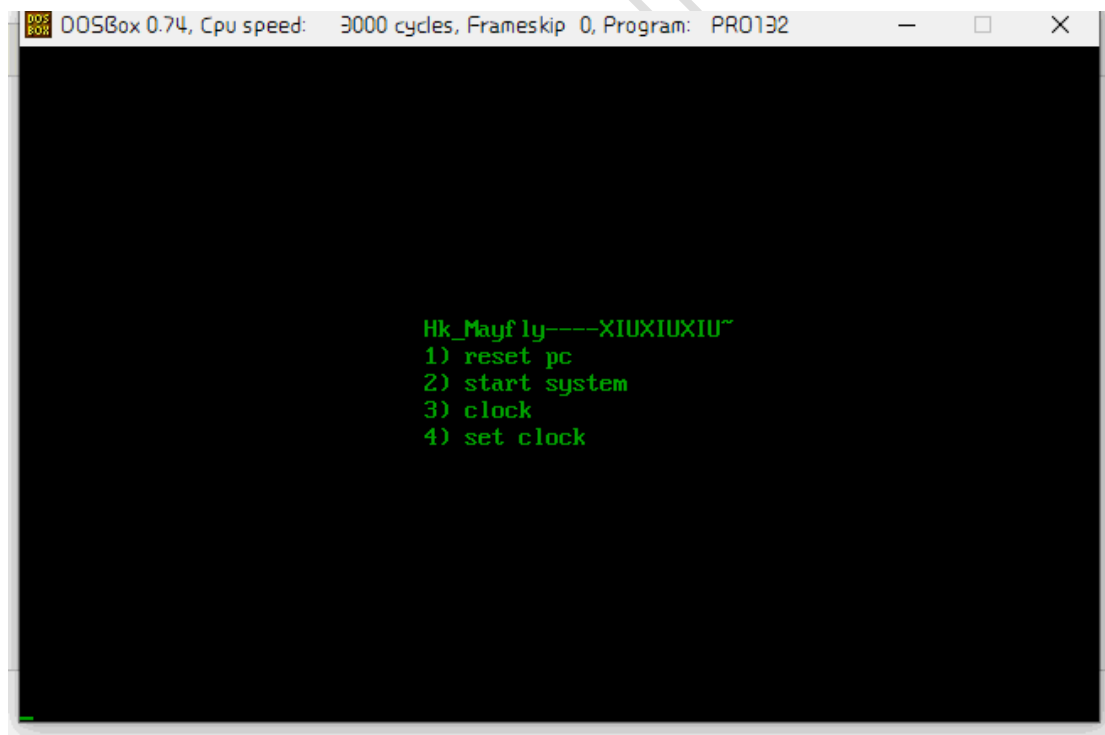
```
;转存引导程序
```

```
copy_boot:
    ;将引导程序储存到指定位置
    mov ax, 0
    mov es, ax
    mov di, 7e00h

    mov ax, cs
    mov ds, ax
    mov si, offset boot
    mov cx, offset boot_end - offset boot
    cld
    rep movsb

    ret

code ends
end start
```



具体的在注释中都说明了。

jz 指令: <https://zhidao.baidu.com/question/564008138.html>

int 16 的 1 号程序: <https://zhidao.baidu.com/question/511189643.html>

总结

汇编的难度并不大，我认为在有编程的基础上，学习汇编要做到细致，细致的理解计算机编程的编译过程，对于我理解其他编程语言也有很大的帮助。欢迎大家关注，一起交流。

HK_Mayfly