

## 汇编语言第二版(王爽)检测点答案详解

本文档包含汇编语言第二版课中的每一处“检测点”、“实验题”的答案。且每一道题目和实验都有详细的解题过程！如果你还是新手，那么你值得拥有这个参考文档！

---

### 检测点1.1

- (1) 1个CPU的寻址能力为8KB，那么它的地址总线的宽度为 13 位。
- (2) 1KB的存储器有 1024 个存储单元，存储单元的编号从 0 到 1023。
- (3) 1KB的存储器可以存储 8192 ( $2^{13}$ ) 个bit，1024 个Byte。
- (4) 1GB是 1073741824 ( $2^{30}$ ) 个Byte、1MB是 1048576 ( $2^{20}$ ) 个Byte、1KB是 1024 ( $2^{10}$ ) 个Byte。
- (5) 8080、8088、80286、80386的地址总线宽度分别为16根、20根、24根、32根，则它们的寻址能力分别为：64 (KB)、1 (MB)、16 (MB)、4 (GB)。
- (6) 8080、8088、8086、80286、80386的数据总线宽度分别为8根、8根、16根、16根、32根。则它们一次可以传送的数据为：1 (B)、1 (B)、2 (B)、2 (B)、4 (B)。
- (7) 从内存中读取1024字节的数据，8086至少要读 512 次，80386至少要读 256 次。
- (8) 在存储器中，数据和程序以 二进制 形式存放。

### 解题过程：

- (1)  $1\text{KB}=1024\text{B}$ ， $8\text{KB}=1024\text{B}\times 8=2^N$ ， $N=13$ 。
- (2) 存储器的容量是以字节为最小单位来计算的， $1\text{KB}=1024\text{B}$ 。
- (3)  $8\text{Bit}=1\text{Byte}$ ， $1024\text{Byte}=1\text{KB}$  ( $1\text{KB}=1024\text{B}=1024\text{B}\times 8\text{Bit}$ )。
- (4)  $1\text{GB}=1073741824\text{B}$  (即  $2^{30}$ )  $1\text{MB}=1048576\text{B}$  (即  $2^{20}$ )  $1\text{KB}=1024\text{B}$  (即  $2^{10}$ )。
- (5) 一个CPU有N根地址线，则可以说这个CPU的地址总线的宽度为N。这样的CPU最多可以寻找  $2^N$  个内存单元。(一个内存单元=1Byte)。
- (6) 8根数据总线一次可以传送8位二进制数据 (即一个字节)。
- (7) 8086的数据总线宽度为16根 (即一次传送的数据为2B)  $1024\text{B}/2\text{B}=512$ ，同理  $1024\text{B}/4\text{B}=256$ 。
- (8) 在存储器中指令和数据没有任何区别，都是二进制信息。

---

### 检测点 2.1

- (1) 写出每条汇编指令执行后相关寄存器中的值。

```
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

```

Microsoft(R) Windows DOS

(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\ADMINI~1>debug

-a

```

0C1C:0100 mov ax,f4a3
0C1C:0103 mov ah,31
0C1C:0105 mov al,23
0C1C:0107 add ax,ax
0C1C:0109 mov bx,826c
0C1C:010C mov cx,ax
0C1C:010E mov ax,bx
0C1C:0110 add ax,bx
0C1C:0112 mov al,bh
0C1C:0114 mov ah,bl
0C1C:0116 add ah,ah
0C1C:0118 add al,6
0C1C:011A add al,al
0C1C:011C mov ax,cx
0C1C:011E

```

-r

```

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0100 NV UP EI PL NZ NA PO NC
0C1C:0100 B8A3F4      MOV     AX,F4A3

```

-t

```

AX=F4A3 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0103 NV UP EI PL NZ NA PO NC
0C1C:0103 B431      MOV     AH,31

```

-t

```

AX=31A3 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0105 NV UP EI PL NZ NA PO NC
0C1C:0105 B023      MOV     AL,23

```

-t

```

AX=3123 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0107 NV UP EI PL NZ NA PO NC
0C1C:0107 01C0      ADD     AX,AX

```

-t

```

AX=6246 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0109 NV UP EI PL NZ NA PO NC
0C1C:0109 BB6C82      MOV     BX,826C

```

```

-t
AX=6246 BX=826C CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=010C NV UP EI PL NZ NA PO NC
0C1C:010C 89C1 MOV CX, AX
-t
AX=6246 BX=826C CX=6246 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=010E NV UP EI PL NZ NA PO NC
0C1C:010E 89D8 MOV AX, BX
-t
AX=826C BX=826C CX=6246 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0110 NV UP EI PL NZ NA PO NC
0C1C:0110 01D8 ADD AX, BX
-t
AX=04D8 BX=826C CX=6246 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0112 OV UP EI PL NZ AC PE CY
0C1C:0112 88F8 MOV AL, BH
-t
AX=0482 BX=826C CX=6246 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0114 OV UP EI PL NZ AC PE CY
0C1C:0114 88DC MOV AH, BL
-t
AX=6C82 BX=826C CX=6246 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0116 OV UP EI PL NZ AC PE CY
0C1C:0116 00E4 ADD AH, AH
-t
AX=D882 BX=826C CX=6246 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0118 OV UP EI NG NZ AC PE NC
0C1C:0118 0406 ADD AL, 06
-t
AX=D888 BX=826C CX=6246 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=011A NV UP EI NG NZ NA PE NC
0C1C:011A 00C0 ADD AL, AL
-t
AX=D810 BX=826C CX=6246 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=011C OV UP EI PL NZ AC PO CY
0C1C:011C 89C8 MOV AX, CX
-t
AX=6246 BX=826C CX=6246 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=011E OV UP EI PL NZ AC PO CY
0C1C:011E 0B0C OR CX, [SI] DS:0000=20CD
-q

```

(2) 只能使用目前学过的汇编指令，最多使用4条指令，编程计算2的4次方。

```
mov ax,2      AX=2
add ax,ax     AX=4
add ax,ax     AX=8
add ax,ax     AX=16
```

Microsoft(R) Windows DOS

(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\ADMINI~1>debug

-a

```
0C1C:0100 mov ax,2
0C1C:0103 add ax,ax
0C1C:0105 add ax,ax
0C1C:0107 add ax,ax
0C1C:0109
```

-r

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0100 NV UP EI PL NZ NA PO NC
0C1C:0100 B80200      MOV     AX,0002
```

-t

```
AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0103 NV UP EI PL NZ NA PO NC
0C1C:0103 01C0      ADD     AX,AX
```

-t

```
AX=0004 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0105 NV UP EI PL NZ NA PO NC
0C1C:0105 01C0      ADD     AX,AX
```

-t

```
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0107 NV UP EI PL NZ NA PO NC
0C1C:0107 01C0      ADD     AX,AX
```

-t

```
AX=0010 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0109 NV UP EI PL NZ AC PO NC
0C1C:0109 20881615    AND     [BX+SI+1516],CL      DS:1516=00
```

-q

---

## 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1000FH。

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  $(SA*16+0h) \sim (SA*16+ffffh)$

现在  $SA=0001h$ , 那么寻址范围为

$(0001h*16+0h) \sim (0001h*16+ffffh)$

$=0010h \sim 1000fh$

---

### 检测点2.2

(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

这里的  $ffffh / 16 = ffffh$  是通过WIN自带计算器算的

按位移来算确实应该为  $fff.fh$ , 这里小数点后的f应该是省略了

单就除法来说, 应有商和余数, 但此题要求的是地址最大和最小, 所以余数忽略了

如果根据位移的算法 (段地址  $*16 = 16$  进制左移一位), 小数点后应该是不能省略的

我们可以反过来再思考下, 如果SA为1000h的话, 小数点后省略

$SA=1000h$ , EA取最大  $ffffh$ , 物理地址为  $1ffffh$ , 将无法寻到20000H单元

这道题不应看成是单纯的计算题

---

### 检测点2.3

下面的3条指令执行后, cpu几次修改IP? 都是在什么时候? 最后IP中的值是多少?

```
mov ax, bx
```

```
sub ax, ax
```

```
jmp ax
```

答: 一共修改四次

第一次: 读取 `mov ax, bx` 之后

第二次: 读取 `sub ax, ax` 之后

第三次: 读取 `jmp ax` 之后

第四次: 执行 `jmp ax` 修改IP

最后IP的值为0000H, 因为最后ax中的值为0000H, 所以IP中的值也为0000H

---

## 实验一 查看CPU和内存，用机器指令和汇编指令编程

### 2实验任务

(1)使用Debug，将下面的程序段写入内存，逐条执行，观察每条指令执行后，CPU中相关寄存器中内容的变化。

机器码	汇编指令	寄存器
b8 20 4e	mov ax, 4E20H	ax=4E20H
05 16 14	add ax, 1416H	ax=6236H
bb 00 20	mov BX, 2000H	bx=2000H
01 d8	add ax, bx	ax=8236H
89 c3	mov bx, ax	bx=8236H
01 d8	add ax, bx	ax=046CH
b8 1a 00	mov ax, 001AH	ax=001AH
bb 26 00	mov bx, 0026H	bx=0026H
00 d8	add al, bl	ax=0040H
00 dc	add ah, bl	ax=2640H
00 c7	add bh, al	bx=4026H
b4 00	mov ah, 0	ax=0040H
00 d8	add al, bl	ax=0066H
04 9c	add al, 9CH	ax=0002H

Microsoft(R) Windows DOS

(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\ADMINI~1>debug

-a

0C1C:0100 mov ax, 4e20

0C1C:0103 add ax, 1416

0C1C:0106 mov bx, 2000

0C1C:0109 add ax, bx

0C1C:010B mov bx, ax

0C1C:010D add ax, bx

0C1C:010F mov ax, 001a

0C1C:0112 mov bx, 0026

0C1C:0115 add al, bl

0C1C:0117 add ah, bl

0C1C:0119 add bh, al

0C1C:011B mov ah, 0

0C1C:011D add al, bl

0C1C:011F add al, 9c

0C1C:0121

-r

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0100 NV UP EI PL NZ NA PO NC

0C1C:0100 B8204E MOV AX, 4E20

-t

AX=4E20	BX=0000	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=0103	NV UP EI PL NZ NA PO NC		
0C1C:0103 051614		ADD		AX, 1416			
-t							
AX=6236	BX=0000	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=0106	NV UP EI PL NZ NA PE NC		
0C1C:0106 BB0020		MOV		BX, 2000			
-t							
AX=6236	BX=2000	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=0109	NV UP EI PL NZ NA PE NC		
0C1C:0109 01D8		ADD		AX, BX			
-t							
AX=8236	BX=2000	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=010B	OV UP EI NG NZ NA PE NC		
0C1C:010B 89C3		MOV		BX, AX			
-t							
AX=8236	BX=8236	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=010D	OV UP EI NG NZ NA PE NC		
0C1C:010D 01D8		ADD		AX, BX			
-t							
AX=046C	BX=8236	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=010F	OV UP EI PL NZ NA PE CY		
0C1C:010F B81A00		MOV		AX, 001A			
-t							
AX=001A	BX=8236	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=0112	OV UP EI PL NZ NA PE CY		
0C1C:0112 BB2600		MOV		BX, 0026			
-t							
AX=001A	BX=0026	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=0115	OV UP EI PL NZ NA PE CY		
0C1C:0115 00D8		ADD		AL, BL			
-t							
AX=0040	BX=0026	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=0117	NV UP EI PL NZ AC PO NC		
0C1C:0117 00DC		ADD		AH, BL			
-t							
AX=2640	BX=0026	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=0119	NV UP EI PL NZ NA PO NC		
0C1C:0119 00C7		ADD		BH, AL			
-t							
AX=2640	BX=4026	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=011B	NV UP EI PL NZ NA PO NC		
0C1C:011B B400		MOV		AH, 00			
-t							
AX=0040	BX=4026	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000
DS=0C1C	ES=0C1C	SS=0C1C	CS=0C1C	IP=011D	NV UP EI PL NZ NA PO NC		
0C1C:011D 00D8		ADD		AL, BL			

```

-t
AX=0066 BX=4026 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=011F NV UP EI PL NZ NA PE NC
0C1C:011F 049C          ADD     AL, 9C
-t
AX=0002 BX=4026 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0121 NV UP EI PL NZ AC PO CY
0C1C:0121 D3990075      RCR     WORD PTR [BX+DI+7500], CL      DS:B526=0000
-q

```

### 实验一 查看CPU和内存，用机器指令和汇编指令编程

(2) 将下面的3条指令写入从2000: 0开始的内存单元中，利用这3条指令计算2的8次方。

```

mov ax, 1
add ax, ax
jmp 2000:0003

```

Microsoft(R) Windows DOS

(C) Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\ADMINI~1>debug

-a 2000:0

2000:0000 mov ax, 1

2000:0003 add ax, ax

2000:0005 jmp 2000:0003

2000:0007

-r cs

CS 0C1C

:2000

-r ip

IP 0100

:0000

-r

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=2000 IP=0000 NV UP EI PL NZ NA PO NC

2000:0000 B80100 MOV AX, 0001

-t

AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=2000 IP=0003 NV UP EI PL NZ NA PO NC

2000:0003 01C0 ADD AX, AX

-t

AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=2000 IP=0005 NV UP EI PL NZ NA PO NC

2000:0005 EBFC JMP 0003

-t

AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000



```

DS=0C1C ES=0C1C SS=0C1C CS=2000 IP=0003 NV UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX, AX
-t
AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=2000 IP=0003 NV UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX, AX
-t
AX=0004 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=2000 IP=0005 NV UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP     0003
-t
AX=0004 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=2000 IP=0003 NV UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX, AX
-t
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=2000 IP=0005 NV UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP     0003
-t
AX=0008 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=2000 IP=0003 NV UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX, AX
-q

```

## 实验一 查看CPU和内存，用机器指令和汇编指令编程

### (3) 查看内存中的内容

PC主板上的ROM中有个一出产日期，在内存FFF00H-FFFFFH的某几个单元中，请找到这个出产日期并试图改变它。

```

Microsoft(R) Windows DOS
(C)Copyright Microsoft Corp 1990-2001.
C:\DOCUME~1\ADMINI~1>debug
-d ffff:0 f
FFFF:0000 EA 5B E0 00 F0 31 32 2F-32 35 2F 30 37 00 FC 59 . [...12/25/07..Y
-q

```

地址C0000~FFFFF的内存单元为只读存储器，写入数据操作是无效的。  
因此出产日期无法改变。

下面内容摘自于网上

还有另一种情况，如果你发现你能修改ROM中的生产日期，那么原因如下：

每个计算机的结构都不一样，教材考虑的是普通8086PC机上的效果，个别计算机的效果可能不同。

也就是说 在你的计算机中 这个内存是可修改的  
所以，认为所有的计算机某些地址的内存不能修改是片面的。

书上说rom是只读的你就不去验证了吗？如何验证呢？

我觉得这个实验最大的好处不是让我们验证了这个知识点，而是提醒我们要有怀疑的精神，怀疑之后再去验证才能跟深刻的理解知识，提升自己的能力，甚至还会发现有些书上描述的不准确甚至错误的地方。

一引用这几本书综合研究的三个问题：

都再用，我们就非得用吗？

规定了，我们就只知道遵守吗？

司空见惯，我们就不怀疑了吗？

尽信书不如无书大概也有这个道理吧^\_^

---

### 检测点3.1

(1) 在DEBUG中,用 “D 0:0 1f” 查看内存,结果如下:

0000:0000 70 80 F0 30 EF 60 30 E2-00 80 80 12 66 20 22 60

0000:0010 62 26 E6 D6 CC 2E 3C 3B-AB BA 00 00 26 06 66 88

下面的程序执行前,AX=0,BX=0,写出每条汇编指令执行完后相关寄存器中的值

```
mov ax,1
mov ds,ax
mov ax,[0000] ax= 2662H
mov bx,[0001] bx= E626H
mov ax,bx ax= E626H
mov ax,[0000] ax= 2662H
mov bx,[0002] bx= D6E6H
add ax,bx ax= FD48H
add ax,[0004] ax= 2C14H
mov ax,0 ax= 0
mov al,[0002] ax= 00e6H
mov bx,0 bx= 0
mov bl,[000c] bx= 0026H
add al,bl ax= 000CH
```

用DEBUG进行验证:

Microsoft(R) Windows DOS

(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\000>debug

-e 0000:0

0000:0000	68.70	10.80	A7.f0	00.30	8B.ef	01.60	70.30	00.e2
0000:0008	16.00	00.80	AF.80	03.12	8B.66	01.20	70.22	00.60
0000:0010	8B.62	01.26	70.e6	00.d6	B9.cc	06.2e	14.3c	02.3b
0000:0018	40.ab	07.ba	14.00	02.00	FF.26	03.06	14.66	02.88

```

-d 0000:0 1f
0000:0000  70 80 F0 30 EF 60 30 E2-00 80 80 12 66 20 22 60    p..0.`0.....f "`
0000:0010  62 26 E6 D6 CC 2E 3C 3B-AB BA 00 00 26 06 66 88    b&....<;....&.f.
-a
0DB4:0100 mov ax,1
0DB4:0103 mov ds,ax
0DB4:0105 mov ax,[0000]
0DB4:0108 mov bx,[0001]
0DB4:010C mov ax,bx
0DB4:010E mov ax,[0000]
0DB4:0111 mov bx,[0002]
0DB4:0115 add ax,bx
0DB4:0117 add ax,[0004]
0DB4:011B mov ax,0
0DB4:011E mov al,[0002]
0DB4:0121 mov bx,0
0DB4:0124 mov bl,[000c]
0DB4:0128 add al,bl
0DB4:012A
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0DB4 ES=0DB4 SS=0DB4 CS=0DB4 IP=0100  NV UP EI PL NZ NA PO NC
0DB4:0100 B80100      MOV     AX,0001
-t
AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0DB4 ES=0DB4 SS=0DB4 CS=0DB4 IP=0103  NV UP EI PL NZ NA PO NC
0DB4:0103 8ED8      MOV     DS,AX
-t
AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=0105  NV UP EI PL NZ NA PO NC
0DB4:0105 A10000      MOV     AX,[0000]                      DS:0000=2662
-t
AX=2662 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=0108  NV UP EI PL NZ NA PO NC
0DB4:0108 8B1E0100    MOV     BX,[0001]                      DS:0001=E626
-t
AX=2662 BX=E626 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=010C  NV UP EI PL NZ NA PO NC
0DB4:010C 89D8      MOV     AX,BX
-t
AX=E626 BX=E626 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=010E  NV UP EI PL NZ NA PO NC
0DB4:010E A10000      MOV     AX,[0000]                      DS:0000=2662
-t
AX=2662 BX=E626 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=0111  NV UP EI PL NZ NA PO NC
0DB4:0111 8B1E0200    MOV     BX,[0002]                      DS:0002=D6E6

```

```

-t
AX=2662 BX=D6E6 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=0115 NV UP EI PL NZ NA PO NC
ODB4:0115 01D8 ADD AX, BX
-t
AX=FD48 BX=D6E6 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=0117 NV UP EI NG NZ NA PE NC
ODB4:0117 03060400 ADD AX, [0004] DS:0004=2ECC
-t
AX=2C14 BX=D6E6 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=011B NV UP EI PL NZ AC PE CY
ODB4:011B B80000 MOV AX, 0000
-t
AX=0000 BX=D6E6 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=011E NV UP EI PL NZ AC PE CY
ODB4:011E A00200 MOV AL, [0002] DS:0002=E6
-t
AX=00E6 BX=D6E6 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=0121 NV UP EI PL NZ AC PE CY
ODB4:0121 BB0000 MOV BX, 0000
-t
AX=00E6 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=0124 NV UP EI PL NZ AC PE CY
ODB4:0124 8A1E0C00 MOV BL, [000C] DS:000C=26
-t
AX=00E6 BX=0026 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=0128 NV UP EI PL NZ AC PE CY
ODB4:0128 00D8 ADD AL, BL
-t
AX=000C BX=0026 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0001 ES=0DB4 SS=0DB4 CS=0DB4 IP=012A NV UP EI PL NZ NA PE CY
ODB4:012A C6061799FF MOV BYTE PTR [9917], FF DS:9917=9A
-q

```

### 检测点3.1

(2) 内存中的情况如图3.6所示

各寄存器的初始值：cs=2000h, ip=0, ds=1000h, ax=0, bx=0;

- ① 写出CPU执行的指令序列（用汇编指令写出）。
- ② 写出CPU执行每条指令后，CS、IP和相关寄存器的数值。
- ③ 再次体会：数据和程序有区别吗？如何确定内存中的信息哪些是数据，哪些是程序？

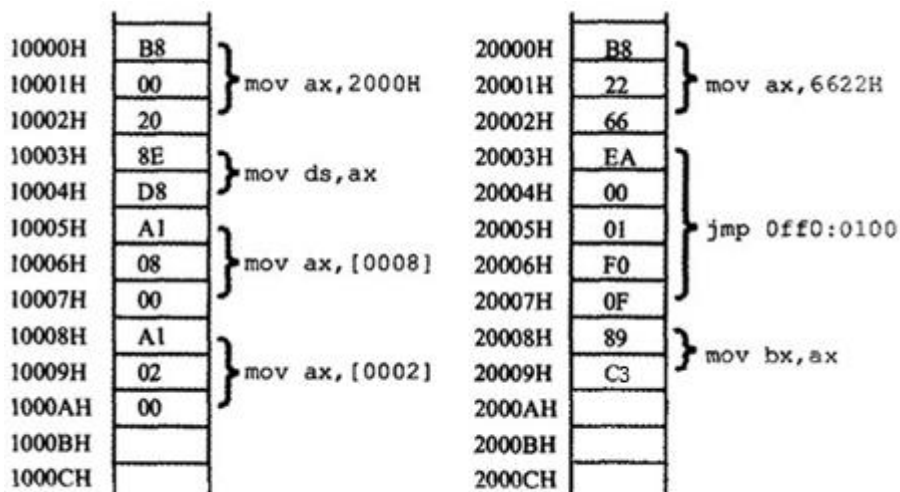


图 3.6 内存情况示意

图3. 6内存情况示意

指令序列		CS	IP	DS	AX	BX
初始值		2000h	0	0	0	0
1	mov ax, 6622h	2000h	3h	0	6622h	0
2	jmp 0ff0:0100	ff0h	100h	0	6622h	0
3	mov ax, 2000h	ff0h	103h	0	2000h	0
4	mov ds, ax	ff0h	105h	2000h	2000h	0
5	mov ax, [8]	ff0h	108h	2000h	c389h	0
6	mov ax, [2]	ff0h	10bh	2000h	ea66h	0

### 检测点3. 2

(1) 补全下面的程序，使其可以将10000H-1000FH中的8个字，逆序拷贝到20000H-2000FH中。

```

mov ax, 1000H
mov ds, ax
mov ax, 2000H
mov ss, ax
mov sp, 10h
push [0]
push [2]
push [4]
push [6]
push [8]
push [A]
push [C]
push [E]

```

---

### 检测点3.2

(2) 补全下面的程序，使其可以将10000H-1000FH中的8个字，逆序拷贝到20000H-2000FH中。

```
mov ax, 2000H
mov ds, ax
mov ax, 1000H
mov ss, ax
mov sp, 0
pop [e]
pop [c]
pop [a]
pop [8]
pop [6]
pop [4]
pop [2]
pop [0]
```

---

### 实验2 用机器指令和汇编指令编程

(1) 使用DEBUG，将上面的程序段写入内存，逐条执行，根据指令执行后的实际运行情况填空。

```
mov ax, ffff
mov ds, ax
mov ax, 2200
mov ss, ax
mov sp, 0100
```

```
mov ax, [0]    ;ax= 5BEAH
add ax, [2]    ;ax= 5CCAH
mov bx, [4]    ;bx= 31F0H
add bx, [6]    ;bx= 6122H
```

```
push ax        ;sp= 00FEH, 修改的内存单元地址是 2200:00FE 内容为 5CCAH
push bx        ;sp= 00FCH, 修改的内存单元地址是 2200:00FC 内容为 6122H
pop ax         ;sp= 00FCH, ax= 6122H
pop bx         ;sp= 00FEH, bx= 5CCAH
```

```
push [4]       ;sp= 00FEH, 修改的内存单元地址是 2200:00FE 内容为 31F0
push [6]       ;sp= 00FCH, 修改的内存单元地址是 2200:00FC 内容为 2F32
```

此实验答案不定，需根据每台机器的实际运行情况。

Microsoft(R) Windows DOS

(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\ADMINI~1>debug

-a

0C1C:0100 mov ax,ffff

0C1C:0103 mov ds,ax

0C1C:0105 mov ax,2200

0C1C:0108 mov ss,ax

0C1C:010A mov sp,0100

0C1C:010D mov ax,[0]

0C1C:0110 add ax,[2]

0C1C:0114 mov bx,[4]

0C1C:0118 add bx,[6]

0C1C:011C push ax

0C1C:011D push bx

0C1C:011E pop ax

0C1C:011F pop bx

0C1C:0120 push [4]

0C1C:0124 push [6]

0C1C:0128

-r

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0100 NV UP EI PL NZ NA PO NC

0C1C:0100 B8FFFF MOV AX,FFFF

-t

AX=FFFF BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0103 NV UP EI PL NZ NA PO NC

0C1C:0103 8ED8 MOV DS,AX

-t

AX=FFFF BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=FFFF ES=0C1C SS=0C1C CS=0C1C IP=0105 NV UP EI PL NZ NA PO NC

0C1C:0105 B80022 MOV AX,2200

-t

AX=2200 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=FFFF ES=0C1C SS=0C1C CS=0C1C IP=0108 NV UP EI PL NZ NA PO NC

0C1C:0108 8ED0 MOV SS,AX

-t

AX=2200 BX=0000 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000

DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=010D NV UP EI PL NZ NA PO NC

0C1C:010D A10000 MOV AX,[0000] DS:0000=5BEA

-d ffff:0 f

FFFF:0000 EA 5B E0 00 F0 31 32 2F-32 35 2F 30 37 00 FC 59 . [...12/25/07..Y

-t

AX=5BEA BX=0000 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000

DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=0110 NV UP EI PL NZ NA PO NC

0C1C:0110 03060200 ADD AX,[0002] DS:0002=00E0

-t

AX=5CCA BX=0000 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000

```

DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=0114 NV UP EI PL NZ NA PE NC
0C1C:0114 8B1E0400 MOV BX, [0004] DS:0004=31F0
-t
AX=5CCA BX=31F0 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=0118 NV UP EI PL NZ NA PE NC
0C1C:0118 031E0600 ADD BX, [0006] DS:0006=2F32
-t
AX=5CCA BX=6122 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=011C NV UP EI PL NZ NA PE NC
0C1C:011C 50 PUSH AX
-t
AX=5CCA BX=6122 CX=0000 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=011D NV UP EI PL NZ NA PE NC
0C1C:011D 53 PUSH BX
-t
AX=5CCA BX=6122 CX=0000 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=011E NV UP EI PL NZ NA PE NC
0C1C:011E 58 POP AX
-t
AX=6122 BX=6122 CX=0000 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=011F NV UP EI PL NZ NA PE NC
0C1C:011F 5B POP BX
-t
AX=6122 BX=5CCA CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=0120 NV UP EI PL NZ NA PE NC
0C1C:0120 FF360400 PUSH [0004] DS:0004=31F0
-t
AX=6122 BX=5CCA CX=0000 DX=0000 SP=00FE BP=0000 SI=0000 DI=0000
DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=0124 NV UP EI PL NZ NA PE NC
0C1C:0124 FF360600 PUSH [0006] DS:0006=2F32
-t
AX=6122 BX=5CCA CX=0000 DX=0000 SP=00FC BP=0000 SI=0000 DI=0000
DS=FFFF ES=0C1C SS=2200 CS=0C1C IP=0128 NV UP EI PL NZ NA PE NC
0C1C:0128 16 PUSH SS
-q

```

## 实验2 用机器指令和汇编指令编程

(2) 仔细观察图3.19中的实验过程，然后分析：为什么2000: 0~2000: F中的内容会发生改变？



```

C:\>debug
-a
0B39:0100  nov ax,2000
0B39:0103  nov ss,ax
0B39:0105  nov sp,10
0B39:0108  nov ax,3123
0B39:010B  push ax
0B39:010C  nov ax,3366
0B39:010F  push ax
0B39:0110
-
-e 2000:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-d 2000:0 f
2000:0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B39  ES=0B39  SS=0B39  CS=0B39  IP=0100  NU UP EI PL NZ NA PO NC
0B39:0100  B80020      MOV     AX,2000
-t
AX=2000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B39  ES=0B39  SS=0B39  CS=0B39  IP=0103  NU UP EI PL NZ NA PO NC
0B39:0103  8ED0      MOV     SS,AX
-t
AX=2000  BX=0000  CX=0000  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=0B39  ES=0B39  SS=2000  CS=0B39  IP=0108  NU UP EI PL NZ NA PO NC
0B39:0108  B82331      MOV     AX,3123
-d 2000:0 f
2000:0000  00 00 00 00 00 00 00 20 00 00 00 01 39 0B 7D 05  ....

```

图 3.19 用 Debug 进行实验的示例

图3.19 用Debug进行实验的示例

答：因为在debug使用T等指令引发了中断造成的，中断过程使用当前栈空间存放cpu关键数据，所以，你的栈里就有些不是你操作的数据了。

这个问题后面会学到，不过这里也要有个印象，因为如果是在中断过程中压栈是栈越界的话，在windows下的命令窗口会强制关闭的。这个可能在你跟踪一些程序的时候会遇到，到时候有个思考方向。

### 实验3 编程、编译、连接、跟踪

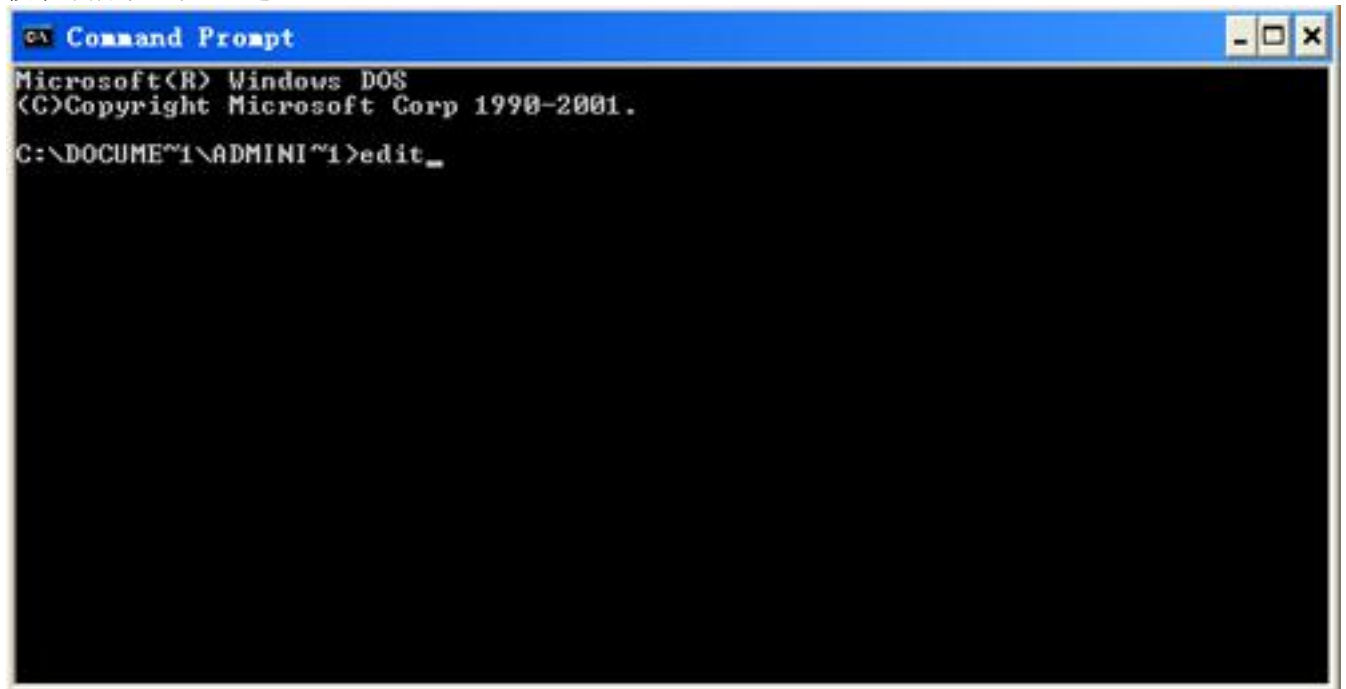
(1) 将下面的程序保存为t1.asm文件，将其生成可执行文件t1.exe。

```

assume cs:codesg
codesg segment
    mov ax,2000H
    mov ss,ax
    mov sp,0
    add sp,10
    pop ax
    pop bx
    push ax
    push bx
    pop ax
    pop bx
    mov ax, 4c00h
    int 21H
codesg ends
end

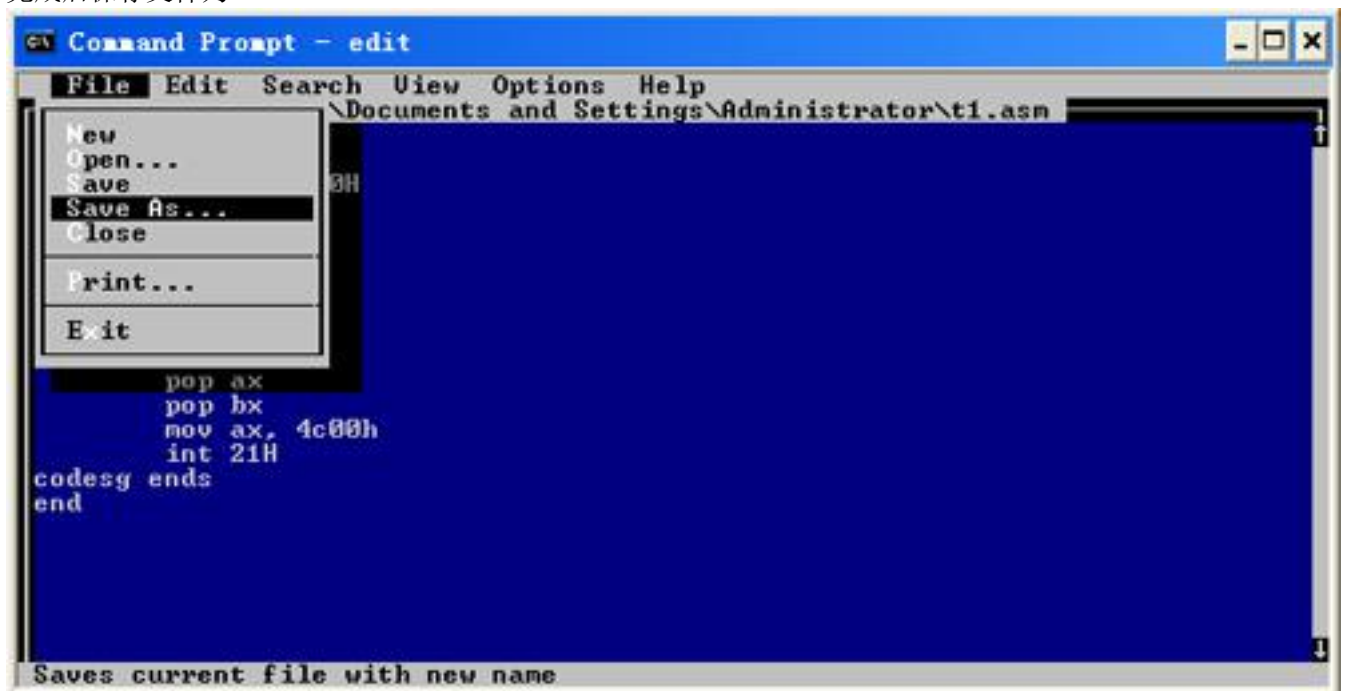
```

按书中所示，从DOS进入EDIT



```
Microsoft(R) Windows DOS
(C)Copyright Microsoft Corp 1990-2001.
C:\DOCUME~1\ADMINI~1>edit_
```

完成后保存文件为t1.asm



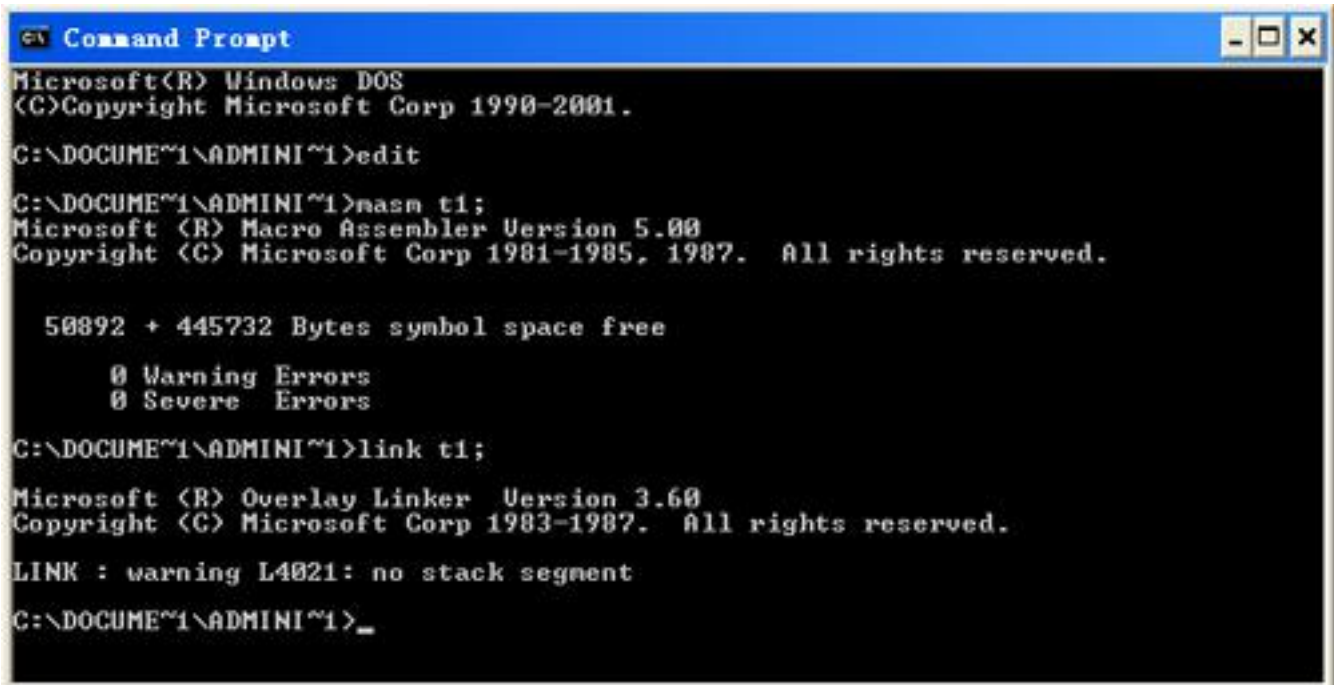
```
File Edit Search View Options Help
\Documents and Settings\Administrator\t1.asm
New
Open...
Save
Save As...
Close
Print...
Exit

    pop ax
    pop bx
    mov ax, 4c00h
    int 21h
codesg ends
end

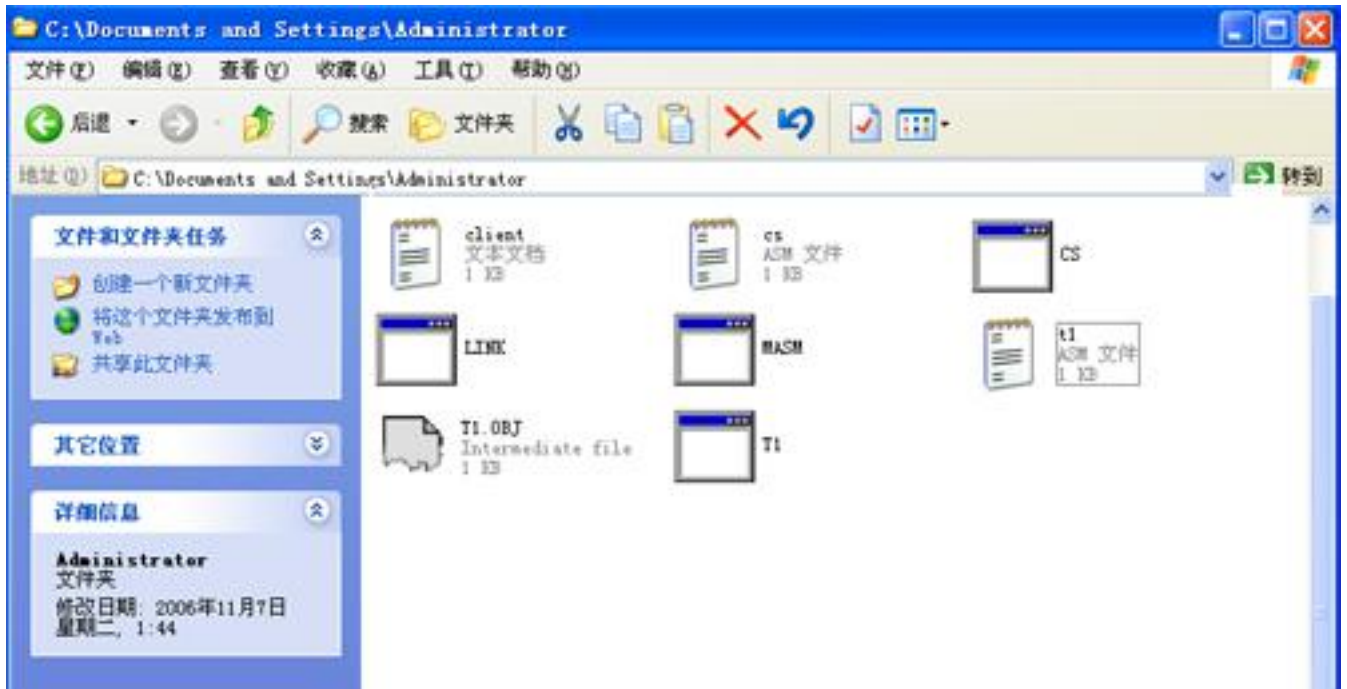
Saves current file with new name
```



退出EDIT，进行编译，连接



生成可执行文件t1.exe



### 实验3 编程、编译、连接、跟踪

(2) 用DEBUG跟踪t1.exe的执行过程，写出第一步执行后，相关寄存器的内容和栈顶内容。

Microsoft(R) Windows DOS

(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\ADMINI~1>debug t1.exe

-r

```
AX=0000 BX=0000 CX=0016 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=0C79 CS=0C79 IP=0000 NV UP EI PL NZ NA PO NC
0C79:0000 B80020 MOV AX,2000
```

-t

```
AX=2000 BX=0000 CX=0016 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=0C79 CS=0C79 IP=0003 NV UP EI PL NZ NA PO NC
0C79:0003 8ED0 MOV SS,AX
```

-t

```
AX=2000 BX=0000 CX=0016 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=2000 CS=0C79 IP=0008 NV UP EI PL NZ NA PO NC
0C79:0008 83C40A ADD SP,+0A
```

-t

```
AX=2000 BX=0000 CX=0016 DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=2000 CS=0C79 IP=000B NV UP EI PL NZ NA PE NC
0C79:000B 58 POP AX
```

-d 2000:0 f

```
2000:0000 00 20 00 00 0B 00 79 0C-80 06 79 0C 80 06 00 1F . . . . y . . y . . . .
```

```

-t
AX=0C79 BX=0000 CX=0016 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=2000 CS=0C79 IP=000C NV UP EI PL NZ NA PE NC
0C79:000C 5B POP BX
-d 2000:0 f
2000:0000 00 20 79 0C 00 00 0C 00-79 0C 80 06 80 06 00 1F . y.....y.....
-t
AX=0C79 BX=0680 CX=0016 DX=0000 SP=000E BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=2000 CS=0C79 IP=000D NV UP EI PL NZ NA PE NC
0C79:000D 50 PUSH AX
-d 2000:0 f
2000:0000 00 20 79 0C 79 0C 00 00-0D 00 79 0C 80 06 00 1F . y.y.....y.....
-t
AX=0C79 BX=0680 CX=0016 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=2000 CS=0C79 IP=000E NV UP EI PL NZ NA PE NC
0C79:000E 53 PUSH BX
-d 2000:0 f
2000:0000 00 20 79 0C 00 00 0E 00-79 0C 80 06 79 0C 00 1F . y.....y...y...
-t
AX=0C79 BX=0680 CX=0016 DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=2000 CS=0C79 IP=000F NV UP EI PL NZ NA PE NC
0C79:000F 58 POP AX
-d 2000:0 f
2000:0000 79 0C 00 00 0F 00 79 0C-80 06 80 06 79 0C 00 1F y.....y.....y...
-t
AX=0680 BX=0680 CX=0016 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=2000 CS=0C79 IP=0010 NV UP EI PL NZ NA PE NC
0C79:0010 5B POP BX
-d 2000:0 f
2000:0000 79 0C 80 06 00 00 10 00-79 0C 80 06 79 0C 00 1F y.....y...y...
-t
AX=0680 BX=0C79 CX=0016 DX=0000 SP=000E BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=2000 CS=0C79 IP=0011 NV UP EI PL NZ NA PE NC
0C79:0011 B8004C MOV AX,4C00
-t
AX=4C00 BX=0C79 CX=0016 DX=0000 SP=000E BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=2000 CS=0C79 IP=0014 NV UP EI PL NZ NA PE NC
0C79:0014 CD21 INT 21
-p
Program terminated normally
-q

```

---

(3)PSP的头两个字节是CD20，用DEBUG加载t1.exe，查看PSP的内容。

Microsoft(R) Windows DOS  
(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\ADMINI~1>debug t1.exe

-r

AX=0000 BX=0000 CX=0016 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000

DS=0C69 ES=0C69 SS=0C79 CS=0C79 IP=0000 NV UP EI PL NZ NA PO NC

0C79:0000 B80020 MOV AX,2000

-d 0c69:0

0C69:0000 CD 20 FF 9F 00 9A F0 FE-1D F0 4F 03 80 06 8A 03 . . . . . 0 . . . . .

0C69:0010 80 06 17 03 80 06 6F 06-01 01 01 00 02 FF FF FF . . . . . o . . . . .

0C69:0020 FF FF FF FF FF FF FF FF-FF FF FF FF 2D 0C 4C 01 . . . . . - . L .

0C69:0030 40 0B 14 00 18 00 69 0C-FF FF FF FF 00 00 00 00 @ . . . . i . . . . .

0C69:0040 05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 . . . . .

0C69:0050 CD 21 CB 00 00 00 00 00-00 00 00 00 20 20 20 . ! . . . . .

0C69:0060 20 20 20 20 20 20 20 20-00 00 00 00 20 20 20 . . . . .

0C69:0070 20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 . . . . .

-q

---

#### 实验4 [bx]和loop的使用

(1) 编程，向内存0:200~0:23f依次传递数据0~63(3fh)。

这是个比较另类的做法，传统做法请参考实验4(2)

assume cs:code

code segment

mov bx,20h

mov ss,bx

mov sp,40h

mov bx,3f3eh

mov cx,32

s: push bx

sub bx,202h

loop s

mov ax,4c00h

int 21h

code ends

end

C:\DOCUME~1\ADMINI~1>debug sy4-2.exe

-d 0:200 23f

0000:0200 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 . . . . .

0000:0210 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 . . . . .

```

0000:0220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0000:0230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
-u
0C79:0000 BB2000      MOV     BX,0020
0C79:0003 8ED3        MOV     SS,BX
0C79:0005 BC4000      MOV     SP,0040
0C79:0008 BB3E3F      MOV     BX,3F3E
0C79:000B B92000      MOV     CX,0020
0C79:000E 53          PUSH    BX
0C79:000F 81EB0202      SUB     BX,0202
0C79:0013 E2F9        LOOP    000E
0C79:0015 B8004C      MOV     AX,4C00
0C79:0018 CD21        INT     21
0C79:001A 8600        XCHG    AL,[BX+SI]
0C79:001C FF508D      CALL    [BX+SI-73]
0C79:001F 46          INC     SI
-g 0015
AX=0000 BX=FEFE CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=0020 CS=0C79 IP=0015  NV UP EI NG NZ AC PO CY
0C79:0015 B8004C      MOV     AX,4C00
-d 0:200 23f
0000:0200  00 01 02 03 04 05 06 07-08 09 0A 0B 0C 0D 0E 0F .....
0000:0210  10 11 12 13 14 15 16 17-18 19 1A 1B 1C 1D 1E 1F .....
0000:0220  20 21 22 23 24 25 26 27-28 29 2A 2B 2C 2D 2E 2F  !"#$%&'()*+,-./
0000:0230  30 31 32 33 34 35 36 37-38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?
-t
AX=4C00 BX=FEFE CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C69 ES=0C69 SS=0020 CS=0C79 IP=0018  NV UP EI NG NZ AC PO CY
0C79:0018 CD21        INT     21
-p
Program terminated normally
-q
C:\DOCUME~1\ADMINI~1>

```

#### 实验4 [bx]和loop的使用

(2) 编程，向内存0:200~0:23f依次传递数据0~63(3fh)，程序中只能使用9条指令，9条指令中包括“mov ax,4c00h”和“int 21h”。

```

assume cs:code
code segment
    mov ax,20h
    mov ds,ax
    mov bx,0

```

```

        mov cx, 40h    ;或mov cx, 64
s:      mov [bx], bl
        inc bx
        loop s
        mov ax, 4c00h
        int 21h
code ends
end

```

C:\DOCUME~1\ADMINI~1>debug sy4-1.exe

-d 0:200 23f

```

0000:0200  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
0000:0210  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
0000:0220  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
0000:0230  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....

```

-u

```

0C79:0000 B82000      MOV     AX, 0020
0C79:0003 8ED8         MOV     DS, AX
0C79:0005 BB0000      MOV     BX, 0000
0C79:0008 B94000      MOV     CX, 0040
0C79:000B 881F         MOV     [BX], BL
0C79:000D 43          INC     BX
0C79:000E E2FB         LOOP    000B
0C79:0010 B8004C      MOV     AX, 4C00
0C79:0013 CD21      INT     21
0C79:0015 CC        INT     3
0C79:0016 FFFF      ???     DI
0C79:0018 50          PUSH    AX
0C79:0019 8D8600FF     LEA     AX, [BP+FF00]
0C79:001D 50          PUSH    AX
0C79:001E 8D4680      LEA     AX, [BP-80]

```

-g 0010

```

AX=0020 BX=0040 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0020 ES=0C69 SS=0C79 CS=0C79 IP=0010 NV UP EI PL NZ AC PO NC
0C79:0010 B8004C      MOV     AX, 4C00

```

-d 0:200 23f

```

0000:0200  00 01 02 03 04 05 06 07-08 09 0A 0B 0C 0D 0E 0F  .....
0000:0210  10 11 12 13 14 15 16 17-18 19 1A 1B 1C 1D 1E 1F  .....
0000:0220  20 21 22 23 24 25 26 27-28 29 2A 2B 2C 2D 2E 2F  !"#$%&'()*+,-./
0000:0230  30 31 32 33 34 35 36 37-38 39 3A 3B 3C 3D 3E 3F  0123456789:;<=>?

```

-t

```

AX=4C00 BX=0040 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0020 ES=0C69 SS=0C79 CS=0C79 IP=0013 NV UP EI PL NZ AC PO NC
0C79:0013 CD21      INT     21

```

-p

Program terminated normally



¬q

#### 实验4 [bx]和loop的使用

(3)下面的程序功能是将“mov ax, 4c00h”之前的指令复制到内存0:200处，补全程序。上机调试，跟踪运行结果。

```
assume cs:code
code segment
    mov ax, __code__ ;或mov ax, __cs__
    mov ds, ax
    mov ax, 0020h
    mov es, ax
    mov bx, 0
    mov cx, __18h__ ;或mov cx, __17h__ ;或sub cx, 5
s: mov al, [bx]
    mov es:[bx], al
    inc bx
    loop s
    mov ax, 4c00h
    int 21h
code ends
end
```

此题有多个答案，因为mov用在寄存器之间传送数据的指令是2个字节，用在寄存器和立即数之间是3个字节

答案1:mov ax, cs (占2个字节)

mov cx, 17

答案2:mov ax, code (占3个字节)

mov cx, 18

答案3:mov ax, cs 或mov ax, code

把mov cx, \_\_改成 sub cx, 5

(因为在载入程序时, cx保存程序的长度, 减去5是为减去mov ax, 4c00h和int 21h的长度)

此题的目的是:

- 1、理解CS和CODE的关联
- 2、理解CS保存程序的代码段，即“复制的是什么，从哪里到哪里”
- 3、理解CX在载入程序后保存程序的长度。
- 4、理解数据和代码对CPU来说是没区别的，只要CS: IP指向的就是代码

C:\DOCUME~1\ADMINI~1>debug sy4-3. exe

¬u

```
0C79:0000 B8790C      MOV     AX, 0C79
0C79:0003 8ED8             MOV     DS, AX
0C79:0005 B82000      MOV     AX, 0020
```

```

0C79:0008 8EC0      MOV     ES, AX
0C79:000A BB0000    MOV     BX, 0000
0C79:000D B91800    MOV     CX, 0018
0C79:0010 8A07      MOV     AL, [BX]
0C79:0012 26          ES:
0C79:0013 8807      MOV     [BX], AL
0C79:0015 43          INC     BX
0C79:0016 E2F8      LOOP    0010
0C79:0018 B8004C    MOV     AX, 4C00
0C79:001B CD21      INT     21
0C79:001D 50          PUSH    AX
0C79:001E 8D4680    LEA     AX, [BP-80]
-g
Program terminated normally
-d 0:200
0000:0200 B8 79 0C 8E D8 B8 20 00-8E C0 BB 00 00 B9 18 00 .y....
0000:0210 8A 07 26 88 07 43 E2 F8-00 00 00 00 00 00 00 ..&..C.....
0000:0220 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0000:0230 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0000:0240 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0000:0250 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0000:0260 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0000:0270 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-u 0:200
0000:0200 B8790C      MOV     AX, 0C79
0000:0203 8ED8      MOV     DS, AX
0000:0205 B82000    MOV     AX, 0020
0000:0208 8EC0      MOV     ES, AX
0000:020A BB0000    MOV     BX, 0000
0000:020D B91800    MOV     CX, 0018
0000:0210 8A07      MOV     AL, [BX]
0000:0212 26          ES:
0000:0213 8807      MOV     [BX], AL
0000:0215 43          INC     BX
0000:0216 E2F8      LOOP    0210
0000:0218 0000      ADD     [BX+SI], AL
0000:021A 0000      ADD     [BX+SI], AL
0000:021C 0000      ADD     [BX+SI], AL
0000:021E 0000      ADD     [BX+SI], AL
-q

```

#### 检测点6.1

(1) 下面的程序实现依次用内存0:0~0:15单元中的内容改写程序中的数据，完成程序：  
 assume cs:codesg

```

codesg segment
    dw 0123h, 0456h, 0789h, 0abch, 0defh, 0fedh, 0cbah, 0987h
start:  mov ax, 0
        mov ds, ax
        mov bx, 0
        mov cx, 8
    s:   mov ax, [bx]
        mov cs:[bx], ax
        add bx, 2
        loop s
        mov ax, 4c00h
        int 21h
codesg ends
end start

```

C:\DOCUME~1\ADMINI~1>debug jc6-1.exe

~u

```

0C79:0010 B80000      MOV     AX, 0000
0C79:0013 8ED8        MOV     DS, AX
0C79:0015 BB0000      MOV     BX, 0000
0C79:0018 B90800      MOV     CX, 0008
0C79:001B 8B07        MOV     AX, [BX]
0C79:001D 2E          CS:
0C79:001E 8907        MOV     [BX], AX
0C79:0020 83C302      ADD     BX, +02
0C79:0023 E2F6        LOOP    001B
0C79:0025 B8004C      MOV     AX, 4C00
0C79:0028 CD21      INT     21
0C79:002A 7503        JNZ     002F
0C79:002C E97BFF      JMP     FFAA
0C79:002F 5E          POP     SI

```

~g 0025

```

AX=0680 BX=0010 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0000 ES=0C69 SS=0C79 CS=0C79 IP=0025 NV UP EI PL NZ AC PO NC
0C79:0025 B8004C      MOV     AX, 4C00

```

~d 0:0 f

```

0000:0000 68 10 A7 00 BB 13 80 06-16 00 A5 03 B1 13 80 06 h.....

```

~d 0c79:0 f

```

0C79:0000 68 10 A7 00 BB 13 80 06-16 00 A5 03 B1 13 80 06 h.....

```

~t

```

AX=4C00 BX=0010 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0000 ES=0C69 SS=0C79 CS=0C79 IP=0028 NV UP EI PL NZ AC PO NC
0C79:0028 CD21      INT     21

```

~p

Program terminated normally

~q

C:\DOCUME~1\ADMINI~1>

检测点6.1

(2) 下面的程序实现依次用内存0:0~0:15单元中的内容改写程序中的数据，数据的传送用栈来进行。栈空间设置在程序内。完成程序：

```
assume cs:codesg
codesg segment
    dw 0123h, 0456h, 0789h, 0abch, 0defh, 0fedh, 0cbah, 0987h
    dw 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
start: mov ax, codesg      ;或mov ax, cs
        mov ss, ax
        mov sp, 24h ;或mov sp, 36      ;(第一版填1ah或26)
        mov ax, 0
        mov ds, ax
        mov bx, 0
        mov cx, 8
s:      push [bx]
        pop cs:[bx]      ;或 pop ss:[bx]
        add bx, 2
        loop s
        mov ax, 4c00h
        int 21h
codesg ends
end start
```

C:\DOCUME~1\ADMINI~1>debug jc6-1-2.exe

~u

0C86:0024	B8860C	MOV	AX, 0C86
0C86:0027	8ED0	MOV	SS, AX
0C86:0029	BC2400	MOV	SP, 0024
0C86:002C	B80000	MOV	AX, 0000
0C86:002F	8ED8	MOV	DS, AX
0C86:0031	BB0000	MOV	BX, 0000
0C86:0034	B90800	MOV	CX, 0008
0C86:0037	FF37	PUSH	[BX]
0C86:0039	2E	CS:	
0C86:003A	8F07	POP	[BX]
0C86:003C	83C302	ADD	BX, +02
0C86:003F	E2F6	LOOP	0037
0C86:0041	B8004C	MOV	AX, 4C00

-g 0041

AX=0000	BX=0010	CX=0000	DX=0000	SP=0024	BP=0000	SI=0000	DI=0000
DS=0000	ES=0C76	SS=0C86	CS=0C86	IP=0041	NV UP EI PL NZ AC PO NC		
0C86:0041	B8004C	MOV	AX, 4C00				

```

-d 0:0 f
0000:0000  68 10 A7 00 BB 13 8D 06-16 00 B2 03 B1 13 8D 06  h.....
-d 0c86:0 f
0C86:0000  68 10 A7 00 BB 13 8D 06-16 00 B2 03 B1 13 8D 06  h.....
-q

```

---

### 实验5 编写、调试具有多个段的程序

(1) 将下面的程序编译连接，用Debug加载、跟踪，然后回答问题。

```

assume cs:code,ds:data,ss:stack
data segment
    dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h
data ends
stack segment
    dw 0,0,0,0,0,0,0,0
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

```

①CPU执行程序，程序返回前，data段中的数据 不变。

②CPU执行程序，程序返回前，CS=0C88H，SS=0C87H，DS=0C86H。

③设程序加载后，CODE段的段地址为X，则DATA段的段地址为X-2，STACK段的段地址为X-1。

C:\DOCUME~1\ADMINI~1>debug sy5-1.exe

```

-r
AX=0000 BX=0000 CX=0042 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C76 ES=0C76 SS=0C86 CS=0C88 IP=0000  NV UP EI PL NZ NA PO NC
0C88:0000 B8870C          MOV     AX,0C87
-d 0c86:0 f
0C86:0000  23 01 56 04 89 07 BC 0A-EF 0D ED 0F BA 0C 87 09  #.V.....
-u
0C88:0000 B8870C          MOV     AX,0C87
0C88:0003 8ED0          MOV     SS,AX

```

```

0C88:0005 BC1000      MOV     SP, 0010
0C88:0008 B8860C      MOV     AX, 0C86
0C88:000B 8ED8        MOV     DS, AX
0C88:000D FF360000     PUSH    [0000]
0C88:0011 FF360200     PUSH    [0002]
0C88:0015 8F060200     POP     [0002]
0C88:0019 8F060000     POP     [0000]
0C88:001D B8004C      MOV     AX, 4C00
-g 001d
AX=0C86 BX=0000 CX=0042 DX=0000 SP=0010 BP=0000 SI=0000 DI=0000
DS=0C86 ES=0C76 SS=0C87 CS=0C88 IP=001D NV UP EI PL NZ NA PO NC
0C88:001D B8004C      MOV     AX, 4C00
-d 0c86:0 f
0C86:0000 23 01 56 04 89 07 BC 0A-EF 0D ED 0F BA 0C 87 09 #.V.....
-q

```

---

#### 实验5 编写、调试具有多个段的程序

(2) 将下面的程序编译连接，用Debug加载、跟踪，然后回答问题。

```
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
```

①CPU执行程序，程序返回前，data段中的数据不变。

②CPU执行程序，程序返回前，CS=0C88H，SS=0C87H，DS=0C86H。

③设程序加载后，CODE段的段地址为X，则DATA段的段地址为X-2，STACK段的段地址为X-1。

④对于如下定义的段：

```
name segment
```

.....

name ends

如果段中的数据占N个字节，则程序加载后，该段实际占有的空间为  $\underline{(N+15)/16*16}$ 。

④解析：

N分为被16整除和不被16整除。

当N被16整除时： 占有的空间为  $(N/16)*16$

当N不被16整除时： 占有的空间为  $(N/16+1)*16$ ，N/16得出的是可以整除的部分，还有一个余数，余数肯定小于16，加上一个16。

程序加载后分配空间是以16个字节为单位的，也就是说如果不足16个字节的也分配16个字节。

两种情况总结成一个通用的公式：  $((N+15)/16)*16$

C:\DOCUME~1\ADMINI~1>debug sy5-2.exe

-r

AX=0000 BX=0000 CX=0042 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000

DS=0C76 ES=0C76 SS=0C86 CS=0C88 IP=0000 NV UP EI PL NZ NA PO NC

0C88:0000 B8870C MOV AX,0C87

-d 0c86:0 3

0C86:0000 23 01 56 04 #. V.

-u

0C88:0000 B8870C MOV AX,0C87

0C88:0003 8ED0 MOV SS,AX

0C88:0005 BC1000 MOV SP,0010

0C88:0008 B8860C MOV AX,0C86

0C88:000B 8ED8 MOV DS,AX

0C88:000D FF360000 PUSH [0000]

0C88:0011 FF360200 PUSH [0002]

0C88:0015 8F060200 POP [0002]

0C88:0019 8F060000 POP [0000]

0C88:001D B8004C MOV AX,4C00

-g 001d

AX=0C86 BX=0000 CX=0042 DX=0000 SP=0010 BP=0000 SI=0000 DI=0000

DS=0C86 ES=0C76 SS=0C87 CS=0C88 IP=001D NV UP EI PL NZ NA PO NC

0C88:001D B8004C MOV AX,4C00

-d 0c86:0 3

0C86:0000 23 01 56 04 #. V.

-q

---

## 实验5 编写、调试具有多个段的程序

(3) 将下面的程序编译连接，用Debug加载、跟踪，然后回答问题。

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

```
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
data segment
        dw 0123h, 0456h
data ends
stack segment
        dw 0, 0
stack ends
end start

```

- ①CPU执行程序，程序返回前，data段中的数据 不变。
- ②CPU执行程序，程序返回前，CS=0C86H，SS=0C8AH，DS=0C89H。
- ③设程序加载后，CODE段的段地址为X，则DATA段的段地址为X+3，STACK段的段地址为X+4。

C:\DOCUME~1\ADMINI~1>debug sj5-3.exe

~r

```

AX=0000 BX=0000 CX=0044 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C76 ES=0C76 SS=0C86 CS=0C86 IP=0000 NV UP EI PL NZ NA PO NC
0C86:0000 B88A0C      MOV     AX,0C8A

```

~u

```

0C86:0000 B88A0C      MOV     AX,0C8A
0C86:0003 8ED0        MOV     SS,AX
0C86:0005 BC1000      MOV     SP,0010
0C86:0008 B8890C      MOV     AX,0C89
0C86:000B 8ED8        MOV     DS,AX
0C86:000D FF360000     PUSH    [0000]
0C86:0011 FF360200     PUSH    [0002]
0C86:0015 8F060200     POP     [0002]
0C86:0019 8F060000     POP     [0000]
0C86:001D B8004C      MOV     AX,4C00

```

~g 001d

```

AX=0C89 BX=0000 CX=0044 DX=0000 SP=0010 BP=0000 SI=0000 DI=0000
DS=0C89 ES=0C76 SS=0C8A CS=0C86 IP=001D NV UP EI PL NZ NA PO NC
0C86:001D B8004C      MOV     AX,4C00

```

~d 0c89:0 3

0C89:0000 23 01 56 04

#. V.

~q



---

#### 实验5 编写、调试具有多个段的程序

(4) 如果将(1)、(2)、(3)题中的最后一条伪指令“end start”改为“end”(也就是说,不指明程序的入口),则哪个程序仍然可以正确执行?请说明原因。

答:第三条程序仍然可以正确执行,如果不指明入口位置,则程序从所分配的空间开始执行,前2个是数据段,只有从第3条开始是指令代码。

---

#### 实验5 编写、调试具有多个段的程序

(5) 程序如下,编写code段中代码,将a段和b段中的数据依次相加,将结果存到C段中。  
(三个程序)

程序一:两次循环

```
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
        mov ds,ax
        mov ax,b
        mov es,ax
        mov bx,0
        mov cx,8
    s:  mov al,[bx]
        add es:[bx],al
        inc bx
        loop s
        mov ax,c
        mov ds,ax
        mov bx,0
        mov cx,8
    s0: mov al,es:[bx]
        mov [bx],al
        inc bx
```

```

        loop s0
        mov ax,4c00h
        int 21h
code ends
end start

```

=====华丽的分割线=====

程序二：一次循环

```

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
        mov ds,ax    ;ds指向a段地址
        mov ax,b
        mov es,ax    ;es指向b段地址
        mov bx,0
        mov cx,8
s:      mov al,[bx]
        add al,es:[bx]
        mov dx,c
        mov ds,dx    ;ds指向c段地址
        mov [bx],al
        mov ax,a
        mov ds,ax    ;重新将ds指向a段（好像此处还能改进）
        inc bx
        loop s
        mov ax,4c00h
        int 21h
code ends
end start

```

=====华丽的分割线=====

程序三：程序二的改进版

```

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
    mov ds,ax
    mov ax,b
    mov es,ax
    mov ax,c
    mov ss,ax
    mov bx,0
    mov cx,8
s:    mov ax,[bx]
    mov ss:[bx],ax
    mov ax,es:[bx]
    add ss:[bx],ax
    inc bx
    loop s
    mov ax,4c00h
    int 21h
code ends
end start

```

---

#### 实验5 编写、调试具有多个段的程序

(6) 程序如下，编写code段中代码，用PUSH指令将A段中的前8个字型数据，逆序存储到B段中。

```

assume cs:code
a segment
    dw 1,2,3,4,5,6,7,8
a ends
b segment
    dw 0,0,0,0,0,0,0,0
b ends
code segment
start: mov ax,a
    mov ds,ax    ;ds指向a段
    mov ax,b
    mov bx,0     ;ds:bx指向a段的第1个单元
    mov ss,ax
    mov sp,16    ;设置栈顶指向b:16
    mov cx,8

```

```

        s: push [bx]
           add bx,2
           loop s           ;将a段中0~16个单元逆次入栈
code ends
end start

```

---

#### 实验6 实践课程中的程序

(1)将课程中所有讲解过的程序上机调试，用DEBUG跟踪其执行过程，并在过程中进一步理解所讲内容。

略

---

#### 实验6 实践课程中的程序

(2)编程，完成问题7.9中的程序。

(编程，将datasg段中每个单词的前4个字母改为大写字母。)

```
assume cs:codesg,ds:datasg,ss:stacksg
```

```
datasg segment
    db '1. display      '
    db '2. brows       '
    db '3. replace     '
    db '4. modify      '
datasg ends
```

```
stacksg segment
    dw 0,0,0,0,0,0,0,0
stacksg ends
```

```
codesg segment
start: mov ax,datasg
       mov ds,ax
       mov bx,0
       mov ax,stacksg
       mov ss,ax
       mov sp,16
       mov cx,4
s0:    push cx
       mov si,0
       mov cx,4
s:     mov al,[bx+3][si]
       and al,11011111b
       mov [bx+3][si],al
       inc si
       loop s
       add bx,16

```

```

        pop cx
        loop s0
        mov ax,4c00h
        int 21h
codesg ends
end start

```

C:\DOCUME~1\SNUSER>debug sy7-9.exe

-d0c4e:0 3f

```

0C4E:0000  31 2E 20 64 69 73 70 6C-61 79 20 20 20 20 20 20  1. display
0C4E:0010  32 2E 20 62 72 6F 77 73-20 20 20 20 20 20 20 20  2. brows
0C4E:0020  33 2E 20 72 65 70 6C 61-63 65 20 20 20 20 20 20  3. replace
0C4E:0030  34 2E 20 6D 6F 64 69 66-79 20 20 20 20 20 20 20  4. modify

```

-g

Program terminated normally

-d 0c4e:0 3f

```

0C4E:0000  31 2E 20 44 49 53 50 6C-61 79 20 20 20 20 20 20  1. DISPlay
0C4E:0010  32 2E 20 42 52 4F 57 73-20 20 20 20 20 20 20 20  2. BROWs
0C4E:0020  33 2E 20 52 45 50 4C 61-63 65 20 20 20 20 20 20  3. REPLace
0C4E:0030  34 2E 20 4D 4F 44 49 66-79 20 20 20 20 20 20 20  4. MODIfy

```

-

## 实验七 寻址方式在结构化访问中的应用（两个程序）

### 程序一：四个循环

```
assume cs:codesg,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'
```

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

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

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

```
    dw 11452,14430,15257,17800
```

```
data ends
```

```
table segment
```

```
    db 21 dup ('year summ ne ?? ')
```

```
table ends
```

```
codesg segment
```

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

```
        mov ds,ax
```

```
        mov ax,table
```

```
        mov ss,ax
```

```

        mov bx, 0
        mov si, 0
        mov bp, 0
        mov cx, 21
s0: mov ax, [bx+si]
    mov [bp+0], ax
    add si, 2
    mov ax, [bx+si]
    mov [bp+2], ax
    add si, 2
    add bp, 10h
    loop s0

        mov cx, 21
        mov bp, 0
        mov si, 0
s1: mov ax, [bx+si+84]
    mov [bp+5], ax
    add si, 2
    mov ax, [bx+si+84]
    mov [bp+7], ax
    add si, 2
    add bp, 10h
    loop s1

        mov cx, 21
        mov bp, 0
        mov si, 0
s2: mov ax, [bx+si+168]
    mov [bp+10], ax
    add si, 2
    add bp, 10h
    loop s2

        mov cx, 21
        mov bp, 0
s3: mov ax, [bp+5]
    mov dx, [bp+7]
    div word ptr [bp+10]
    mov [bp+13], ax
    add bp, 10h
    loop s3

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

```

-----华丽的分割线-----

程序二：一个循环

```
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'
    dd 16,22,382,1356,2390,8000,16000,24486,50065,97479,140417,197514
    dd 345980,590827,803530,1183000,1843000,2759000,3753000,4649000,5937000
    dw 3,7,9,13,28,38,130,220,476,778,1001,1442,2258,2793,4037,5635,8226
    dw 11452,14430,15257,17800
data ends
table segment
    db 21 dup ('year summ ne ?? ')
table ends
code segment
start: mov ax,data
        mov ds,ax
        mov ax,table
        mov es,ax
        mov bx,0
        mov si,0
        mov di,0
        mov cx,21
s:      mov ax,[bx]
        mov es:[si],ax
        mov ax,[bx].2
        mov es:[si].2,ax

        mov ax,[bx].84
        mov es:[si].5,ax
        mov dx,[bx].86
        mov es:[si].7,dx

        div word ptr ds:[di].168
        mov es:[si].13,ax

        mov ax,[di].168
        mov es:[si].10,ax

        add di,2
        add bx,4
        add si,16
        loop s

        mov ax,4c00h
        int 21h
```

code ends  
end start

-----华丽的分割线-----

程序执行前内存数据

0B80:0000	31 39 37 35 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1975	summ	ne	??
0B80:0010	31 39 37 36 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1976	summ	ne	??
0B80:0020	31 39 37 37 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1977	summ	ne	??
0B80:0030	31 39 37 38 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1978	summ	ne	??
0B80:0040	31 39 37 39 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1979	summ	ne	??
0B80:0050	31 39 38 30 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1980	summ	ne	??
0B80:0060	31 39 38 31 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1981	summ	ne	??
0B80:0070	31 39 38 32 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1982	summ	ne	??
0B80:0080	31 39 38 33 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1983	summ	ne	??
0B80:0090	31 39 38 34 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1984	summ	ne	??
0B80:00A0	31 39 38 35 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1985	summ	ne	??
0B80:00B0	31 39 38 36 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1986	summ	ne	??
0B80:00C0	31 39 38 37 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1987	summ	ne	??
0B80:00D0	31 39 38 38 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1988	summ	ne	??
0B80:00E0	31 39 38 39 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1989	summ	ne	??
0B80:00F0	31 39 39 30 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1990	summ	ne	??
0B80:0100	31 39 39 31 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1991	summ	ne	??
0B80:0110	31 39 39 32 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1992	summ	ne	??
0B80:0120	31 39 39 33 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1993	summ	ne	??
0B80:0130	31 39 39 34 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1994	summ	ne	??
0B80:0140	31 39 39 35 20 73 75 6D-6D 20 6E 65 20 3F 3F 20	1995	summ	ne	??

程序执行后内存数据

0B80:0000	31 39 37 35 20 10 00 00-00 20 03 00 20 05 00 20	1975	....	..	..
0B80:0010	31 39 37 36 20 16 00 00-00 20 07 00 20 03 00 20	1976	....	..	..
0B80:0020	31 39 37 37 20 7E 01 00-00 20 09 00 20 2A 00 20	1977	~...	..	*
0B80:0030	31 39 37 38 20 4C 05 00-00 20 0D 00 20 68 00 20	1978	L...	..	h.
0B80:0040	31 39 37 39 20 56 09 00-00 20 1C 00 20 55 00 20	1979	V...	..	U.
0B80:0050	31 39 38 30 20 40 1F 00-00 20 26 00 20 D2 00 20	1980	@...	&.	..
0B80:0060	31 39 38 31 20 80 3E 00-00 20 82 00 20 7B 00 20	1981	.>..	..	{.
0B80:0070	31 39 38 32 20 A6 5F 00-00 20 DC 00 20 6F 00 20	1982	._..	..	o.
0B80:0080	31 39 38 33 20 91 C3 00-00 20 DC 01 20 69 00 20	1983	....	..	i.
0B80:0090	31 39 38 34 20 C7 7C 01-00 20 0A 03 20 7D 00 20	1984	. ..	..	}.
0B80:00A0	31 39 38 35 20 81 24 02-00 20 E9 03 20 8C 00 20	1985	.\$..	..	..
0B80:00B0	31 39 38 36 20 8A 03 03-00 20 A2 05 20 88 00 20	1986	....	..	..
0B80:00C0	31 39 38 37 20 7C 47 05-00 20 D2 08 20 99 00 20	1987	G..	..	..
0B80:00D0	31 39 38 38 20 EB 03 09-00 20 E9 0A 20 D3 00 20	1988	....	..	..
0B80:00E0	31 39 38 39 20 CA 42 0C-00 20 C5 0F 20 C7 00 20	1989	.B..	..	..
0B80:00F0	31 39 39 30 20 18 0D 12-00 20 03 16 20 D1 00 20	1990	....	..	..
0B80:0100	31 39 39 31 20 38 1F 1C-00 20 22 20 20 E0 00 20	1991	8...	"	..
0B80:0110	31 39 39 32 20 58 19 2A-00 20 16 2D 20 EF 00 20	1992	X.*.	.-	..
0B80:0120	31 39 39 33 20 28 44 39-00 20 5E 38 20 04 01 20	1993	(D9.	^8	..
0B80:0130	31 39 39 34 20 28 F0 46-00 20 99 3B 20 30 01 20	1994	(.F.	.;	0.
0B80:0140	31 39 39 35 20 68 97 5A-00 20 88 45 20 4D 01 20	1995	h.Z.	.E	M.



---

### 检测点9.1

(1) 程序如下。

```
assume cs:code
data segment
    dw 2 dup (0)
data ends
code segment
    start: mov ax, data
           mov ds, ax
           mov bx, 0
           jmp word ptr [bx+1]
code ends
end start
```

若要使jmp指令执行后，CS:IP指向程序的第一条指令，在data段中应该定义哪些数据？

答案①db 3 dup (0)

答案②dw 2 dup (0)

答案③dd 0

jmp word ptr [bx+1]为段内转移，要CS:IP指向程序的第一条指令，应设置ds:[bx+1]的字单元(2个字节)存放数据应为0，则(ip)=ds:[bx+1]=0

简单来说就是，只要ds:[bx+1]起始地址的两个字节为0就可以了

---

### 检测点9.1

(1) 程序如下。

```
assume cs:code
data segment
    dd 12345678h
data ends
code segment
    start: mov ax, data
           mov ds, ax
           mov bx, 0
           mov [bx], __bx__ ;或mov [bx], word ptr 0 ;或mov [bx], offset start
           mov [bx+2], __cs__ ;或mov [bx+2], cs ;或mov [bx+2], seg code
           jmp dword ptr ds:[0]
code ends
end start
```

补全程序，使用jmp指令执行后，CS:IP指向程序的第一条指令。

第一格可填①mov [bx],bx                      ②mov [bx],word ptr 0                      ③mov [bx],offset start等。

第二格可填①mov [bx+2],cs    ②mov [bx+2],cs                      ③mov [bx+2],seg code等。

解析:

jmp dword ptr ds:[0]为段间转移, (cs)=(内存单元地址+2), (ip)=(内存单元地址), 要CS:IP指向程序的第一条指令, 第一条程序地址cs:0, 应设置CS:IP指向cs:0

程序中的mov [bx],bx这条指令, 是将ip设置为0

mov [bx+2],cs, 将cs这个段地址放入内存单元

执行后, cs应该不变, 只调整ip为0, (ip)=ds:[0]=0

C:\DOCUME~1\SNUSER>debug jc9-1.exe

-r

```
AX=0000 BX=0000 CX=0021 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3E ES=0C3E SS=0C4E CS=0C4F IP=0000 NV UP EI PL NZ NA PO NC
0C4F:0000 B84E0C MOV AX,0C4E
```

-t

```
AX=0C4E BX=0000 CX=0021 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3E ES=0C3E SS=0C4E CS=0C4F IP=0003 NV UP EI PL NZ NA PO NC
0C4F:0003 8ED8 MOV DS,AX
```

-t

```
AX=0C4E BX=0000 CX=0021 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C4E ES=0C3E SS=0C4E CS=0C4F IP=0005 NV UP EI PL NZ NA PO NC
0C4F:0005 BB0000 MOV BX,0000
```

-t

```
AX=0C4E BX=0000 CX=0021 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C4E ES=0C3E SS=0C4E CS=0C4F IP=0008 NV UP EI PL NZ NA PO NC
0C4F:0008 891F MOV [BX],BX DS:0000=5678
```

-t

```
AX=0C4E BX=0000 CX=0021 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C4E ES=0C3E SS=0C4E CS=0C4F IP=000A NV UP EI PL NZ NA PO NC
0C4F:000A 8C4F02 MOV [BX+02],CS DS:0002=1234
```

-t

```
AX=0C4E BX=0000 CX=0021 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C4E ES=0C3E SS=0C4E CS=0C4F IP=000D NV UP EI PL NZ NA PO NC
0C4F:000D FF2E0000 JMP FAR [0000] DS:0000=0000
```

-t

```
AX=0C4E BX=0000 CX=0021 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C4E ES=0C3E SS=0C4E CS=0C4F IP=0000 NV UP EI PL NZ NA PO NC
0C4F:0000 B84E0C MOV AX,0C4E
```

-q

---

### 检测点9.1

(3) 用Debug查看内存, 结果如下:

2000:1000 BE 00 06 00 00 00 .....

则此时，CPU执行指令：

```
mov ax,2000h
```

```
mov es,ax
```

```
jmp dword ptr es:[1000h]
```

后，(cs)=0006H，(ip)=00BEH

解析：

jmp dword ptr为段间转移，高位存放段地址，低位存放偏移地址

(cs)=(内存单元地址+2)，(ip)=(内存单元地址)

根据书P16，对于寄存器AX，AH为高位(前1字节为高位)，AL为低位(后1字节为低位)

推算出(内存单元地址)=00beh，(内存单元地址+2)=0006h

根据书P182，高位存放段地址(后2个字节为高位)，低位存放偏移地址(前2个字节为低位)

(cs)=(内存单元地址+2)，(ip)=(内存单元地址)

推算出(cs)=0006h，(ip)=00beh

用debug跟踪，可能会出现如下错误，debug给出的答案是(cs)不变，(ip)=1000h

```
C:\DOCUME~1\SNUSER>debug
```

```
-r es
```

```
ES 0BF9
```

```
:2000
```

```
-e 2000:1000 be 00 06 00 00 00
```

```
-a
```

```
0BF9:0100 mov ax,2000
```

```
0BF9:0103 mov es,ax
```

```
0BF9:0105 jmp dword ptr es:[1000]
```

```
^ Error
```

```
0BF9:0105 jmp dword ptr 2000:1000
```

```
0BF9:0108
```

```
-r
```

```
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
```

```
DS=0BF9 ES=2000 SS=0BF9 CS=0BF9 IP=0100 NV UP EI PL NZ NA PO NC
```

```
0BF9:0100 B80020 MOV AX,2000
```

```
-t
```

```
AX=2000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
```

```
DS=0BF9 ES=2000 SS=0BF9 CS=0BF9 IP=0103 NV UP EI PL NZ NA PO NC
```

```
0BF9:0103 8EC0 MOV ES,AX
```

```
-t
```

```
AX=2000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
```

```
DS=0BF9 ES=2000 SS=0BF9 CS=0BF9 IP=0105 NV UP EI PL NZ NA PO NC
```

```
0BF9:0105 E9F80E JMP 1000
```

```
-t
```

```
AX=2000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
```

```
DS=0BF9 ES=2000 SS=0BF9 CS=0BF9 IP=1000 NV UP EI PL NZ NA PO NC
```

```
0BF9:1000 E475 IN AL,75
```

出现错误的原因是：

jmp dword ptr es:[1000H]对应的debug下的指令并不是你给出的

jmp dword ptr 2000:[1000H]这样的形式，可以看出，当你写出上述指令后，运行的时候其指令仅仅变成了jmp 1000，缺少了一个指定段地址的指令。

我们可以写一个源程序模拟一下上面的这段程序

```
assume cs:codesg
data segment
    db 0BEH,0,6,0,0,0
data ends
codesg segment
start:
mov ax,data
mov es,ax
jmp dword ptr es:[0H]
codesg ends
end start
```

上面这个程序，数据地址是程序分配的，不是指定的那个地址，但是，对于我们理解程序运行的整个过程没有影响。下面是debug的信息

```
-t
AX=1438 BX=0000 CX=001A DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=1428 ES=1428 SS=1438 CS=1439 IP=0003 NV UP EI PL NZ NA PO NC
1439:0003 8EC0          MOV     ES,AX
-t
AX=1438 BX=0000 CX=001A DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=1428 ES=1438 SS=1438 CS=1439 IP=0005 NV UP EI PL NZ NA PO NC
1439:0005 26          ES:
1439:0006 FF2E0000     JMP     FAR [0000]          ES:0000=00BE
-d es:0 f
1438:0000 BE 00 06 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
-t
AX=1438 BX=0000 CX=001A DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=1428 ES=1438 SS=1438 CS=0006 IP=00BE NV UP EI PL NZ NA PO NC
0006:00BE 00F0          ADD     AL,DH
```

我们可以看到，源程序中jmp dword ptr es:[0H] 对应的debug下的汇编指令是

```
1439:0005 26          ES:
1439:0006 FF2E0000     JMP     FAR [0000]
```

而不是仅仅的一个（JMP 地址）那样的形式，所以，你在debug下的操作本身就是不行的。

另外，此题目的检测目的就是将内存中的数据作为跳转的CS和IP的值来进行跳转。对于给定的一个地址A，A开始的一个字单元是IP，A+2开始的一个字段元是CS。也就是以A为其实地址的内存中，低字单元是IP，高字单元是CS。

如非要在DEBUG中进行操作，可用以下方式：

```
-e 2000:1000 be 00 06 00 00 00
-a
139A:0100 mov ax,2000
139A:0103 mov es,ax
139A:0105 es:
139A:0106 jmp far [1000]
```

```

139A:010A
-u
139A:0100 B80020      MOV     AX, 2000
139A:0103 8EC0        MOV     ES, AX
139A:0105 26          ES:
139A:0106 FF2E0010    JMP     FAR [1000]
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=139A ES=139A SS=139A CS=139A IP=0100  NV UP EI PL NZ NA PO NC
139A:0100 B80020      MOV     AX, 2000
-t
AX=2000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=139A ES=139A SS=139A CS=139A IP=0103  NV UP EI PL NZ NA PO NC
139A:0103 8EC0        MOV     ES, AX
-t
AX=2000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=139A ES=2000 SS=139A CS=139A IP=0105  NV UP EI PL NZ NA PO NC
139A:0105 26          ES:
139A:0106 FF2E0010    JMP     FAR [1000]                      ES:1000=00BE
-t
AX=2000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=139A ES=2000 SS=139A CS=0006 IP=00BE  NV UP EI PL NZ NA PO NC
0006:00BE 00F0        ADD     AL, DH
-

```

---

## 检测点9.2

补全编程，利用jcxz指令，实现在内存2000H段中查找第一个值为0的字节，找到后，将它的偏移地址存储在dx中。

```

assume cs:code
code segment
    start: mov ax, 2000h
           mov ds, ax
           mov bx, 0
    s: mov ch, 0
       mov cl, [bx]
       jcxz ok ;当cx=0时，CS:IP指向OK
       inc bx
       jmp short s
    ok: mov dx, bx
       mov ax, 4c00h
       int 21h
code ends
end start

```

---

### 检测点9.3

补全编程，利用loop指令，实现在内存2000H段中查找第一个值为0的字节，找到后，将它的偏移地址存储在dx中。

```
assume cs:code
code segment
start: mov ax,2000h
        mov ds,ax
        mov bx,0
s:      mov cl,[bx]
        mov ch,0
        inc cx
        inc bx
        loop s
ok:     dec bx
        mov dx,bx
        mov ax,4c00h
        int 21h
code ends
end start
```

书P101，执行loop s时，首先要将(cx)减1。

“loop 标号”相当于

```
dec cx
if((cx)≠0) jmp short 标号
```

---

### 实验8

```
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
```

ax=0
占一字节, 机器码90
占一字节, 机器码90
(di)=s偏移地址
(si)=s2偏移地址
(ax)=jmp short s1指令对应的机器码EBF6
jmp short s1覆盖s处指令2条nop指令
执行s???? 未执行到这里, 直接跳回mov ax,4c00h了

```

        int 21h
        mov ax,0
s2: jmp short s1
        nop
codesg ends
end start

```

P180中关于 jmp 指令的位移内容

当指令执行到s0: jmp short s时，该指令得到执行，编译器算出的ip位移量为 $8-18h=-16$ (补码F0)， $(ip)=(ip)+位移量=18h+(-16)=8$ ，cs:8指向s；

当指令执行到s标段 jmp 命令时，第1个字节中的机器码为EBF6，给出的ip位移量为-10(补码F6)， $(ip)=(ip)+位移量=ah+(-10)=0$ ，cs:0指向第一条指令。

```
C:\DOCUME~1\SNUSER>debug sy8.exe
```

```
-u
```

```

0C4E:0005 B80000      MOV     AX,0000
0C4E:0008 90             NOP
0C4E:0009 90             NOP
0C4E:000A BF0800      MOV     DI,0008
0C4E:000D BE2000      MOV     SI,0020
0C4E:0010 2E             CS:
0C4E:0011 8B04          MOV     AX,[SI]
0C4E:0013 2E             CS:
0C4E:0014 8905          MOV     [DI],AX
0C4E:0016 EBF0          JMP     0008
0C4E:0018 B80000      MOV     AX,0000
0C4E:001B CD21          INT     21
0C4E:001D B80000      MOV     AX,0000
0C4E:0020 EBF6          JMP     0018
0C4E:0022 90             NOP
0C4E:0023 FEFE          ???     DH

```

---

实验9 显示三行welcome to masm (三个程序)

编程：在屏幕中间分别显示绿色，绿底红色，白色蓝底的字符串' welcome to masm!'

```
Command Prompt
Microsoft(R) Windows DOS
(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\SNUSER>masm sy9;
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.

50776 + 446488 Bytes symbol space free

0 Warning Errors
0 Severe Errors
welcome to masm!
welcome to masm!
C:\DOCUME~1\SNUSER>link sy9;
Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

LINK : warning L4021: no stack segment
C:\DOCUME~1\SNUSER>sy9
C:\DOCUME~1\SNUSER>
```

效果图

程序一：最保守的方法先实现实验要求（三个循环）

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

```
data segment
```

```
    db 'welcome to masm!'
```

```
data ends
```

```
table segment
```

```
    dw 4000 dup (0)
```

```
table ends
```

```
code segment
```

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

```
        mov ds,ax
```

```
        mov ax,0b800h
```

```
        mov es,ax
```

```
        mov bx,0
```

```
        mov si,0
```

```
        mov cx,16
```

```
s0:     mov ax,[bx]
```

```
        mov es:[bx+720h][si],ax
```

```
        mov al,2
```

```
        mov es:[bx+721h][si],al
```

```
        inc bx
```

```
        inc si
```

```
    loop s0
```

```
        mov bx,0
```

```
        mov cx,16
```

```
        mov si,160
```

```
s1:     mov ax,[bx]
```

```
        mov es:[bx+720h][si],ax
```

```
        mov al,36
```

```
        mov es:[bx+721h][si],al
```

```
        inc bx
```



```

        inc si
        loop s1

        mov bx,0
        mov cx,16
        mov si,320
s2:     mov ax,[bx]
        mov es:[bx+720h][si],ax
        mov al,113
        mov es:[bx+721h][si],al
        inc bx
        inc si
        loop s2

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

```

=====华丽的分割线1=====

程序二：一个循环

```

assume cs:code
data segment
        db 'welcome to masn!'
data ends
code segment
start:  mov ax,data
        mov ds,ax
        mov ax,0b800h
        mov es,ax
        mov bx,0720h           ;设置中间行中间列的首地址
        mov si,0
        mov cx,16
s:      mov ax,[si]
        mov ah,2h
        mov es:[bx],ax         ;设置绿色字体
        mov ah,24h
        mov es:[bx].0a0h,ax     ;设置绿底红色
        mov ah,71h
        mov es:[bx].0a0h.0a0h,ax ;设置白底蓝色
        inc si                  ;指向下一字符
        add bx,2                ;指向下一显存单元
        loop s
        mov ax,4c00h
        int 21h
code ends
end start

```

=====华丽的分割线2=====

程序三：一个循环

```
assume cs:code,ds:data
data segment
    db 'welcome to masm!'
data ends
code segment
start: mov ax,data
      mov ds,ax
      mov bx,0                ;ds:bx指向data字符串
      mov ax,0b800h
      mov es,ax
      mov si,0                ;es:si指向显存
      mov cx,16
s:     mov al,[bx]             ;字符赋值al
      mov ah,02h              ;绿色
      mov es:[si].720h,ax     ;写入第12行64列
      mov ah,14h              ;绿底红色
      mov es:[si].7c0h,ax     ;写入第13行64列
      mov ah,71h              ;白底蓝色
      mov es:[si].860h,ax     ;写入第14行64列
      inc bx                  ;指向下一字符
      add si,2                ;指向下一显存单元
      loop s
      mov ax,4c00h
      int 21h
code ends
end start
```

检测点10.1

补全程序，实现从内存1000: 0000处开始执行指令。

```
assume cs:code
stack segment
    db 16 dup (0)
stack ends
code segment
start: mov ax,stack
      mov ss,ax
      mov sp,16
      mov ax,1000h
      push ax
      mov ax,0
```

```

        push ax
        retf
code ends
end start

```

执行retf指令时，相当于进行：

```

pop ip
pop cs

```

根据栈先进后出原则，应先将段地址cs入栈，再将偏移地址ip入栈。

C:\DOCUME~1\SNUSER>debug jc10-1.exe

~u

```

0C50:0000 B84F0C      MOV     AX,0C4F
0C50:0003 8ED0          MOV     SS,AX
0C50:0005 BC1000      MOV     SP,0010
0C50:0008 B80010      MOV     AX,1000
0C50:000B 50            PUSH    AX
0C50:000C B80000      MOV     AX,0000
0C50:000F 50            PUSH    AX
0C50:0010 CB          RETF
0C50:0011 3986FEFE      CMP     [BP+FEFE],AX
0C50:0015 737D          JNB     0094

```

~g 0010

```

AX=0000 BX=0000 CX=0021 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C50 IP=0010 NV UP EI PL NZ NA PO NC
0C50:0010 CB          RETF

```

~t

```

AX=0000 BX=0000 CX=0021 DX=0000 SP=0010 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=1000 IP=0000 NV UP EI PL NZ NA PO NC
1000:0000 6E            DB      6E

```

-

## 检测点10.2

下面的程序执行后，ax中的数值为多少？

内存地址	机器码	汇编指令	执行后情况
1000:0	b8 00 00	mov ax,0	ax=0 ip指向1000:3
1000:3	e8 01 00	call s	pop ip ip指向1000:7
1000:6	40	inc ax	
1000:7	58	s:pop ax	ax=6

用debug进行跟踪确认，“call 标号”是将该指令后的第一个字节偏移地址入栈，再转到标号处执行指令。

```

assume cs:code
code segment
start: mov ax,0
        call s
        inc ax
s:      pop ax
        mov ax,4c00h
        int 21h
code ends
end start

```

C:\DOCUME~1\SNUSER>debug jc10-2.exe

~u

```

0C4F:0000 B80000      MOV     AX,0000
0C4F:0003 E80100      CALL    0007
0C4F:0006 40          INC     AX
0C4F:0007 58          POP     AX
0C4F:0008 B8004C      MOV     AX,4C00
0C4F:000B CD21      INT     21

```

~r

```

AX=0000 BX=0000 CX=000D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C4F IP=0000  NV UP EI PL NZ NA PO NC
0C4F:0000 B80000      MOV     AX,0000

```

~t

```

AX=0000 BX=0000 CX=000D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C4F IP=0003  NV UP EI PL NZ NA PO NC
0C4F:0003 E80100      CALL    0007

```

~t

```

AX=0000 BX=0000 CX=000D DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C4F IP=0007  NV UP EI PL NZ NA PO NC
0C4F:0007 58          POP     AX

```

~t

```

AX=0006 BX=0000 CX=000D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C4F IP=0008  NV UP EI PL NZ NA PO NC
0C4F:0008 B8004C      MOV     AX,4C00

```

~t

```

AX=4C00 BX=0000 CX=000D DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C4F IP=000B  NV UP EI PL NZ NA PO NC
0C4F:000B CD21      INT     21

```

~p

Program terminated normally

---

检测点10.3

下面的程序执行后，ax中的数值为多少？

内存地址	机器码	汇编指令	执行后情况
1000:0	b8 00 00	mov ax,0	ax=0, ip指向1000:3
1000:3	9a 09 00 00 10	call far ptr s	pop cs, pop ip, ip指向1000:9
1000:8	40	inc ax	
1000:9	58	s:pop ax	ax=8h
		add ax, ax	ax=10h
		pop bx	bx=1000h
		add ax, bx	ax=1010h

用debug进行跟踪确认，“call far ptr s”是先将该指令后的第一个字节段地址cs=1000h入栈，再将偏移地址ip=8h入栈，最后转到标号处执行指令。

出栈时，根据栈先进后出的原则，先出的为ip=8h，后出的为cs=1000h

---

#### 检测点10.4

下面的程序执行后，ax中的数值为多少？

内存地址	机器码	汇编指令	执行后情况
1000:0	b8 06 00	mov ax,6	ax=6, ip指向1000:3
1000:3	ff d0	call ax	pop ip, ip指向1000:6
1000:5	40	inc ax	
1000:6	58	mov bp, sp	bp=sp=ffffh
		add ax, [bp]	ax=[6+ds:(ffffh)]=6+5=0bh

用debug进行跟踪确认，“call ax(16位reg)”是先将该指令后的第一个字节偏移地址ip入栈，再转到偏移地址为ax(16位reg)处执行指令。

---

#### 检测点10.5

(1) 下面的程序执行后，ax中的数值为多少？

```
assume cs:code
stack segment
    dw 8 dup (0)
stack ends
code segment
start: mov ax, stack
        mov ss, ax
        mov sp, 16
        mov ds, ax
        mov ax, 0
        call word ptr ds:[0eh]
        inc ax
```

```

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

```

推算：

执行call word ptr ds:[0eh]指令时，先cs入栈，再ip=11入栈，最后ip转移到(ds:[0eh])。  
(ds:[0eh])=11h，执行inc ax……最终ax=3

题中特别关照别用debug跟踪，跟踪结果不一定正确，但还是忍不住去试试，看是什么结果。

根据单步跟踪发现，执行call word ptr ds:[0eh]指令时，显示ds:[0eh]=065D。

ds:0000~ds:0010不是已设置成stack数据段了嘛，不是应该全都是0的嘛。

于是进行了更详细的单步跟踪，发现初始数据段中数据确实为0，但执行完mov ss, ax; mov sp, 16  
这两条指令后，数据段中数据发生改变。这是为什么呢？中断呗~~~~

C:\DOCUME~1\SNUSER>debug jcl0-5.exe

~u

```

0C50:0000 B84F0C      MOV     AX,0C4F
0C50:0003 8ED0          MOV     SS,AX
0C50:0005 BC1000      MOV     SP,0010
0C50:0008 8ED8          MOV     DS,AX
0C50:000A B80000      MOV     AX,0000
0C50:000D FF160E00     CALL    [000E]
0C50:0011 40             INC     AX
0C50:0012 40             INC     AX
0C50:0013 40             INC     AX
0C50:0014 B8004C      MOV     AX,4C00
0C50:0017 CD21      INT     21

```

~r

```

AX=0000 BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C50 IP=0000  NV UP EI PL NZ NA PO NC
0C50:0000 B84F0C      MOV     AX,0C4F

```

-d 0c4f:0 f

```

0C4F:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....

```

~t

```

AX=0C4F BX=0000 CX=0029 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C50 IP=0003  NV UP EI PL NZ NA PO NC
0C50:0003 8ED0          MOV     SS,AX

```

-d 0c4f:0 f

```

0C4F:0000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....

```

~t

```

AX=0C4F BX=0000 CX=0029 DX=0000 SP=0010 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C50 IP=0008  NV UP EI PL NZ NA PO NC
0C50:0008 8ED8          MOV     DS,AX

```

-d 0c4f:0 f

```

0C4F:0000  00 00 00 00 00 00 4F 0C-00 00 08 00 50 0C 5D 06  ....0....P.].
-t
AX=0C4F  BX=0000  CX=0029  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=0C4F  ES=0C3F  SS=0C4F  CS=0C50  IP=000A  NV UP EI PL NZ NA PO NC
0C50:000A B80000          MOV     AX,0000
-d 0c4f:0 f
0C4F:0000  00 00 00 00 00 00 4F 0C-00 00 0A 00 50 0C 5D 06  ....0....P.].
-t
AX=0000  BX=0000  CX=0029  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=0C4F  ES=0C3F  SS=0C4F  CS=0C50  IP=000D  NV UP EI PL NZ NA PO NC
0C50:000D FF160E00      CALL    [000E]                DS:000E=065D
-d 0c4f:0 f
0C4F:0000  00 00 00 00 00 00 00 00-00 00 0D 00 50 0C 5D 06  .....P.].
-

```

#### 检测点10.5

(2) 下面的程序执行后，ax和bx中的数值为多少？

```

assume cs:codesg
stack segment
    dw 8 dup(0)
stack ends
codesg segment
start:
    mov ax,stack
    mov ss,ax
    mov sp,10h
    mov word ptr ss:[0],offset s ;(ss:[0])=1ah
    mov ss:[2],cs                ;(ss:[2])=cs
    call dword ptr ss:[0]        ;cs入栈,ip=19h入栈,转到cs:1ah处执行指令
                                ;(ss:[4])=cs,(ss:[6])=ip

    nop
s:   mov ax,offset s             ;ax=1ah
    sub ax,ss:[0ch]              ;ax=1ah-(ss:[0ch])=1ah-19h=1
    mov bx,cs                    ;bx=cs=0c5bh
    sub bx,ss:[0eh]              ;bx=cs-cs=0
    mov ax,4c00h
    int 21h
codesg ends
end start

```

C:\DOCUME~1\ADMINI~1>debug jc10-5.exe

```

-u
0C5B:0000 B85A0C          MOV     AX,0C5A
0C5B:0003 8ED0          MOV     SS,AX

```

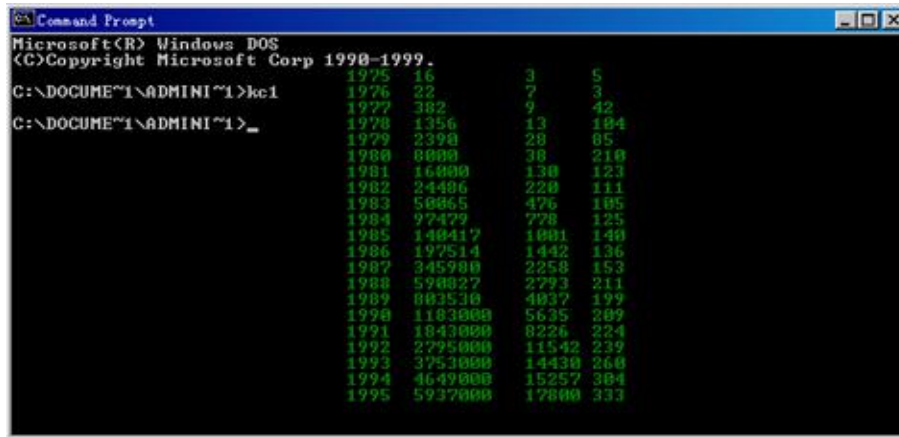
```

0C5B:0005 BC1000      MOV     SP, 0010
0C5B:0008 36          SS:
0C5B:0009 C70600001A00    MOV     WORD PTR [0000], 001A
0C5B:000F 36          SS:
0C5B:0010 8C0E0200    MOV     [0002], CS
0C5B:0014 36          SS:
0C5B:0015 FF1E0000    CALL    FAR [0000]
0C5B:0019 90          NOP
0C5B:001A B81A00      MOV     AX, 001A
0C5B:001D 36          SS:
0C5B:001E 2B060C00    SUB     AX, [000C]
-u
0C5B:0022 8CCB      MOV     BX, CS
0C5B:0024 36          SS:
0C5B:0025 2B1E0E00    SUB     BX, [000E]
0C5B:0029 B8004C      MOV     AX, 4C00

```

## 课程设计一

将实验7中的Power idea公司的数据按照图所示的格式在屏幕上显示现来。



效果图

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

table 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'

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

dd 345980, 590827, 803530, 1183000, 1843000, 2795000, 3753000, 4649000, 5937000

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

dw 11542, 14430, 15257, 17800

dw 5, 3, 42, 104, 85, 210, 123, 111, 105, 125, 140, 136, 153, 211, 199, 209, 224, 239

dw 260, 304, 333



```

table ends
data segment
    db 32 dup (0)
data ends
code segment
start: mov ax,data
        mov ds,ax
        mov ax,table
        mov es,ax
        mov bx,0
        mov si,0
        mov di,0
        mov cx,21
        mov dh,2
        mov dl,30
g:      push cx
        push dx
        mov ax,es:[bx]
        mov [si],ax
        mov ax,es:[bx].2
        mov [si].2,ax ;年份入ds:si
        add si,6
        mov ax,es:[bx].84
        mov dx,es:[bx].86
        call dtoc2    ;收入转成十进制字符入ds:si
        add si,10
        mov ax,es:[di].168
        mov dx,0
        call dtoc2    ;人数转成十进制字符入ds:si
        add si,6
        mov ax,es:[di].210
        mov dx,0
        call dtoc2    ;人均收入转成十进制字符入ds:si
        mov si,0      ;设置ds:si指向需显示字符首地址
b:      mov cx,29
c:      push cx
        mov cl,[si]
        jcxz f        ;(ds:si)=0转到f执行
d:      inc si
        pop cx
        loop c
        inc si
        mov al,0
        mov [si],al   ;设置结尾符0
        mov si,0      ;设置ds:si指向需显示字符首地址
        pop dx
        mov cl,2

```

```

        call show_str
        add bx,4           ;dword数据指向下一数据单元
        add di,2           ;word数据指向下一数据单元
        add dh,1           ;指向显存下一行
        pop cx
        loop g
        mov ax,4c00h
        int 21h
f:      mov al,20h
        mov [si],al       ;(ds:si)=0的数据改成空格
        jmp d

;名称: dtoc2
;功能: 将dword型数据转变为表示十进制的字符串，字符串以0为结尾符。
;参数: (ax)=dword型数据的低16位；
;       (dx)=dword型数据的高16位；
;       ds:si指向字符串首地址。
;返回: 无。
dtoc2:
        push ax
        push bx
        push cx
        push dx
        push si
        push di
        mov di,0
d20:    mov cx,10           ;除数为10
        call divdw
        add cx,30h         ;余数+30h，转为字符
        push cx            ;字符入栈
        inc di             ;记录字符个数
        mov cx,ax
        jcxz d21           ;低位商=0时，转到d21检测高位商
        jmp d20
d21:    mov cx,dx
        jcxz d22           ;高低位商全=0时，转到d22执行
        jmp d20
d22:    mov cx,di
d23:    pop ax             ;字符出栈
        mov [si],al
        inc si             ;ds:si指向下一单元
        loop d23
        mov al,0
        mov [si],al       ;设置结尾符0
        pop di
        pop si
        pop dx

```

```

pop cx
pop bx
pop ax
ret

```

;名称: divdw  
 ;功能: 进行不会产生溢出的除法运算, 被除数为dword型, 除数为word型, 结果为dword型。  
 ;参数: (ax)=dword型数据的低16位;  
 ;       (dx)=dword型数据的高16位;  
 ;       (cx)=除数。  
 ;返回: (dx)=结果的高16位;  
 ;       (ax)=结果的低16位;  
 ;       (cx)=余数。

```

divdw:
    push si
    push bx
    push ax
    mov ax, dx
    mov dx, 0
    div cx          ;被除数的高位/cx
    mov si, ax
    pop ax
    div cx          ;(被除数高位的商+低位)/cx
    mov cx, dx      ;余数入cx
    mov dx, si      ;高位的商入dx
    pop bx
    pop si
    ret

```

;名称: show\_str  
 ;功能: 在指定的位置, 用指定的颜色, 显示一个用0结束的字符串。  
 ;参数: (dh)=行号(取值范围0~24);  
 ;       (dl)=列号(取值范围0~79);  
 ;       (cl)=颜色;  
 ;       ds:si指向字符串的首地址。  
 ;返回: 无。

```

show_str:
    push ax
    push bx
    push es
    push si
    mov ax, 0b800h
    mov es, ax
    mov ax, 160
    mul dh
    mov bx, ax      ;bx=160*dh
    mov ax, 2

```

```

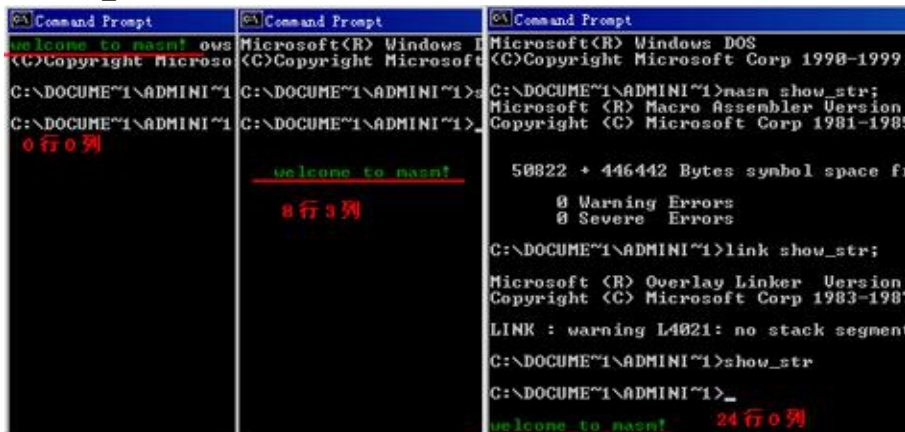
        mul dl          ;ax=dl*2
        add bx,ax        ;mov bx, (160*dh+dl*2) 设置es:bx指向显存首地址
        mov al,cl        ;把颜色cl赋值al
        mov cl,0
show0:
        mov ch,[si]
        jcxz show1      ;(ds:si)=0时, 转到show1执行
        mov es:[bx],ch
        mov es:[bx].1,al
        inc si          ;ds:si指向下一个字符地址
        add bx,2         ;es:bx指向下一个显存地址
        jmp show0
show1:
        pop si
        pop es
        pop bx
        pop ax
        ret

code ends
end start

```

## 实验10

### (1) show\_str子程序



效果图

;在屏幕的8行3列, 用绿色显示data段中的字符串。

assume cs:code,ds:data

data segment

db 'welcome to masn!',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

```

;名称: show\_str  
 ;功能: 在指定的位置, 用指定的颜色, 显示一个用0结束的字符串。  
 ;参数: (dh)=行号(取值范围0~24);  
 ; (dl)=列号(取值范围0~79);  
 ; (cl)=颜色;  
 ; ds:si指向字符串的首地址。  
 ;返回: 无。

```

show_str:
  push ax
  push bx
  push es
  push si
  mov ax,0b800h
  mov es,ax
  mov ax,160
  mul dh
  mov bx,ax      ;bx=160*dh
  mov ax,2
  mul dl         ;ax=dl*2
  add bx,ax      ;mov bx,(160*dh+dl*2) 设置es:bx指向显存首地址
  mov al,cl      ;把颜色cl赋值al
  mov cl,0

show0:
  mov ch,[si]
  jcxz show1     ;(ds:si)=0时, 转到show1执行
  mov es:[bx],ch
  mov es:[bx].1,al
  inc si        ;ds:si指向下一个字符地址
  add bx,2      ;es:bx指向下一个显存地址
  jmp show0

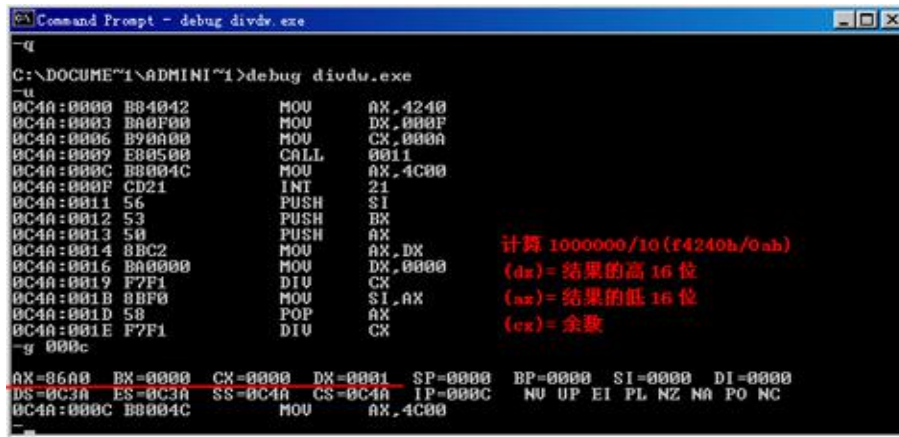
show1:
  pop si
  pop es
  pop bx
  pop ax
  ret

```

code ends  
end start

## 实验10

### (2) divdw子程序



```
Command Prompt - debug divdw.exe
-q
C:\DOCUMENTS\ADMINI~1>debug divdw.exe
-u
0C4A:0000 B84042      MOV     AX,4240
0C4A:0003 BA0F00      MOV     DX,000F
0C4A:0006 B90A00      MOV     CX,000A
0C4A:0009 EB0500      CALL   0011
0C4A:000C B8004C      MOV     AX,4C00
0C4A:000F CD21      INT     21
0C4A:0011 56      PUSH    SI
0C4A:0012 53      PUSH    BX
0C4A:0013 50      PUSH    AX
0C4A:0014 8BC2      MOV     AX,DX      计算 1000000/10 (f4240h/0ah)
0C4A:0016 BA0000      MOV     DX,0000    (dx)= 结果的高 16 位
0C4A:0019 F7F1      DIV     CX          (ax)= 结果的低 16 位
0C4A:001B 8BF0      MOV     SI,AX      (cx)= 余数
0C4A:001D 58      POP     AX
0C4A:001E F7F1      DIV     CX
-g 000c
AX=86A0  BX=0000  CX=0000  DX=0001  SP=0000  BP=0000  SI=0000  DI=0000
DS=0C3A  ES=0C3A  SS=0C4A  CS=0C4A  IP=000C  NU UP EI PL NZ NA PO NC
0C4A:000C B8004C      MOV     AX,4C00
```

效果图

;应用举例：计算1000000/10 (f4240h/0ah)

assume cs:code

code segment

start: mov ax,4240h

mov dx,000fh

mov cx,0ah

call divdw

mov ax,4c00h

int 21h

;名称：divdw

;功能：进行不会产生溢出的除法运算，被除数为dword型，除数为word型，结果为dword型。

;参数：(ax)=dword型数据的低16位；

; (dx)=dword型数据的高16位；

; (cx)=除数。

;返回：(dx)=结果的高16位；

; (ax)=结果的低16位；

; (cx)=余数。

divdw:

push si

push bx

push ax

mov ax,dx

```

        mov dx, 0
        div cx          ;被除数的高位/cx, 高位在ax, 余数在dx
        mov si, ax
        pop ax
        div cx          ;(被除数高位的商+低位)/cx, 高位在ax, 余数在dx
        mov cx, dx      ;余数入cx
        mov dx, si      ;高位的商入dx

        pop bx
        pop si
        ret

code ends
end start

```

公式 $x/n = \text{int}(h/n) * 65536 + [\text{rem}(h/n) * 65536 + 1] / n$ 解析:

把一个会溢出的除法 变成几个除法来做!

如果高位除法有商, 那么商就是结果的高位值, 如果会有余数, 那么余数自然不能丢弃, 余数就作为低位除法的dx (也就是高位的被除数, 因为他是从高位除法中余下的)

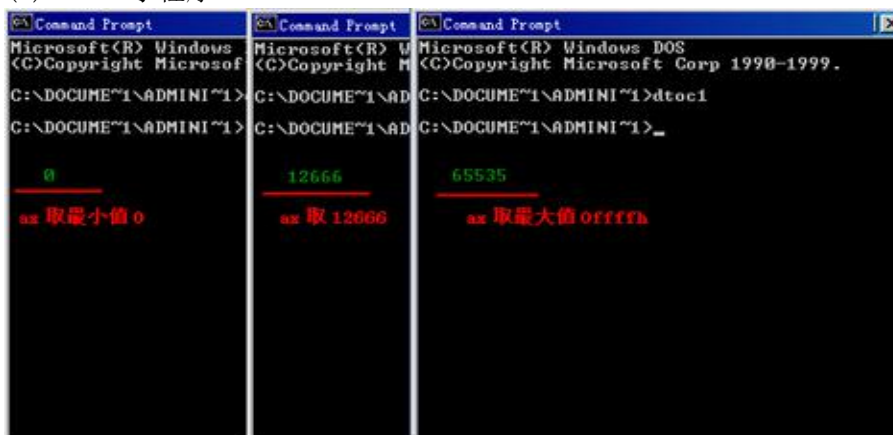
做低位除法的时候, 就拿余下的dx+低位数的ax除以除数, 会得到 一个低位的商 和 余数

高位的商+低位数的商+余数 就是结果

这个程序的理解和完成, 应该是这章比较重要的一个点。咱得多注意这种变通和分解的方式来处理问题的思维方式。

## 实验10

### (3) dtoc1子程序



效果图

;应用举例: 将数据12666以十进制的形式在屏幕的8行3列, 用绿色显示出来。

;在显示时调用子程序show\_str。

assume cs:code,ds:data

```

data segment
    db 10 dup (0)
data ends
code segment
start: mov ax, 12666
    mov bx, data
    mov ds, bx
    mov si, 0      ;ds:si指向data首地址
    call dtoc1

    mov dh, 8
    mov dl, 3
    mov cl, 2
    call show_str

    mov ax, 4c00h
    int 21h

;名称: dtoc1
;功能: 将word型数据转变为表示十进制的字符串，字符串以0为结尾符。
;参数: (ax)=word型数据;
;      ds:si指向字符串首地址。
;返回: 无。
dtoc1: push ax
    push bx
    push cx
    push dx
    push si
    push di
    mov di, 0
d10:  mov dx, 0      ;设置被除数高位为0
    mov bx, 10      ;除数为10
    div bx
    add dx, 30h     ;ax/10的余数+30h，转为字符
    push dx         ;字符入栈
    inc di          ;记录字符个数
    mov cx, ax
    jcxz d11        ;当ax/10的商=0时，转到d11执行
    jmp d10
d11:  mov cx, di
d12:  pop dx         ;字符出栈
    mov [si], dl
    inc si          ;ds:si指向下一单元
    loop d12
    mov dl, 0
    mov [si], dl    ;设置结尾符0
    pop si

```



```

        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret

;名称: show_str
;功能: 在指定的位置, 用指定的颜色, 显示一个用0结束的字符串。
;参数: (dh)=行号(取值范围0~24);
;       (dl)=列号(取值范围0~79);
;       (cl)=颜色;
;       ds:si指向字符串的首地址。
;返回: 无。
show_str:
        push ax
        push bx
        mov ax, 0b800h
        mov es, ax
        mov ax, 160
        mul dh
        mov bx, ax           ;bx=160*dh
        mov ax, 2
        mul dl               ;ax=dl*2
        add bx, ax           ;mov bx, (160*dh+dl*2) 设置es:bx指向显存首地址
        mov al, cl           ;把颜色cl赋值al
        mov cl, 0

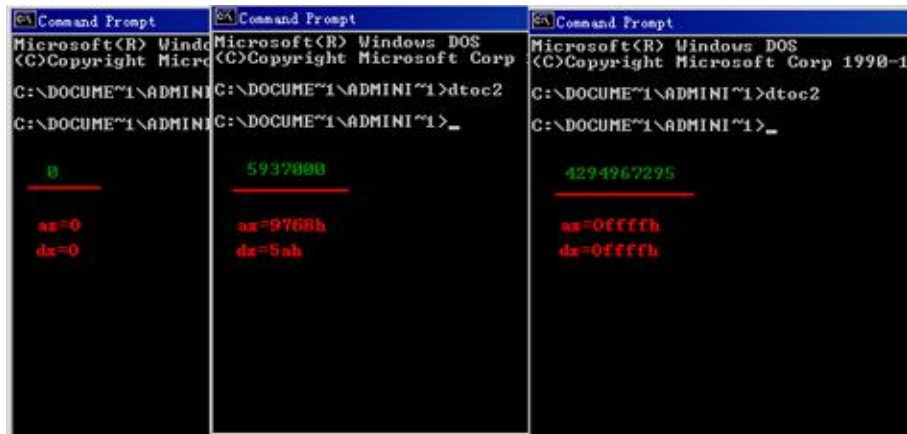
show0:
        mov ch, [si]
        jcxz show1          ;(ds:si)=0时, 转到show1执行
        mov es:[bx], ch
        mov es:[bx].1, al
        inc si               ;ds:si指向下一个字符地址
        add bx, 2            ;es:bx指向下一个显存地址
        jmp show0

show1:
        pop bx
        pop ax
        ret

code ends
end start

```

#### (4) dtoc2子程序



效果图

;应用举例：将数据2494967295以十进制的形式在屏幕的8行3列，用绿色显示出来。

assume cs:code,ds:data

data segment

db 10 dup (0)

data ends

code segment

start: mov ax,0ffffh

mov dx,0ffffh

mov bx,data

mov ds,bx

mov si,0 ;ds:si指向字符串首地址

call dtoc2

mov dh,8

mov dl,3

mov cl,2

call show\_str

mov ax,4c00h

int 21h

;名称: dtoc2

;功能: 将dword型数据转变为表示十进制的字符串，字符串以0为结尾符。

;参数: (ax)=dword型数据的低16位;

; (dx)=dword型数据的高16位;

; ds:si指向字符串首地址。

;返回: 无。

dtoc2:

push ax

push bx

push cx

push dx

```

        push si
        push di
        mov di,0
d20:    mov cx,10          ;除数为10
        call divdw
        add cx,30h        ;余数+30h, 转为字符
        push cx           ;字符入栈
        inc di            ;记录字符个数
        mov cx,ax
        jcxz d21          ;低位商=0时, 转到d21检测高位商
        jmp d20
d21:    mov cx,dx
        jcxz d22          ;高低位商全=0时, 转到d22执行
        jmp d20
d22:    mov cx,di
d23:    pop ax            ;字符出栈
        mov [si],al
        inc si            ;ds:si指向下一单元
        loop d23
        mov al,0
        mov [si],al      ;设置结尾符0
        pop di
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        ret

```

;名称: show\_str

;功能: 在指定的位置, 用指定的颜色, 显示一个用0结束的字符串。

;参数: (dh)=行号(取值范围0~24);

; (dl)=列号(取值范围0~79);

; (cl)=颜色;

; ds:si指向字符串的首地址。

;返回: 无。

show\_str:

```

        push ax
        push bx
        mov ax,0b800h
        mov es,ax
        mov ax,160
        mul dh
        mov bx,ax        ;bx=160*dh
        mov ax,2
        mul dl            ;ax=dl*2
        add bx,ax         ;mov bx, (160*dh+dl*2) 设置es:bx指向显存首地址

```

```

        mov al, c1          ;把颜色c1赋值al
        mov cl, 0
show0:
        mov ch, [si]
        jcxz show1         ;(ds:si)=0时, 转到show1执行
        mov es:[bx], ch
        mov es:[bx].1, al
        inc si              ;ds:si指向下一个字符地址
        add bx, 2           ;es:bx指向下一个显存地址
        jmp show0
show1:
        pop bx
        pop ax
        ret

;名称: divdw
;功能: 进行不会产生溢出的除法运算, 被除数为dword型, 除数为word型, 结果为dword型。
;参数: (ax)=dword型数据的低16位;
;       (dx)=dword型数据的高16位;
;       (cx)=除数。
;返回: (dx)=结果的高16位;
;       (ax)=结果的低16位;
;       (cx)=余数。
divdw:
        push si
        push bx
        push ax
        mov ax, dx
        mov dx, 0
        div cx              ;被除数的高位/cx
        mov si, ax
        pop ax
        div cx              ;(被除数高位的商+低位)/cx
        mov cx, dx          ;余数入cx
        mov dx, si          ;高位的商入dx
        pop bx
        pop si
        ret
code ends
end start

```

---

#### 检测点11.1

写出下面每条指令执行后, ZF、PF、SF、等标志位的值。

sub al, al      al=0h      ZF=1      PF=1      SF=0

mov al,1	al=1h	ZF=1	PF=1	SF=0
push ax	ax=1h	ZF=1	PF=1	SF=0
pop bx	bx=1h	ZF=1	PF=1	SF=0
add al,b1	al=2h	ZF=0	PF=0	SF=0
add al,10	al=12h	ZF=0	PF=1	SF=0
mul al	ax=144h	ZF=0	PF=1	SF=0

检测点涉及的相关内容：

ZF是flag的第6位，零标志位，记录指令执行后结果是否为0，结果为0时，ZF=1

PF是flag的第2位，奇偶标志位，记录指令执行后结果二进制中1的个数是否为偶数，结果为偶数时，PF=1

SF是flag的第7位，符号标志位，记录有符号运算结果是否为负数，结果为负数时，SF=1

add、sub、mul、div、inc、or、and等运算指令影响标志寄存器

mov、push、pop等传送指令对标志寄存器没影响。

C:\DOCUME~1\ADMINI~1>debug

-a

0C1C:0100 sub al,al

0C1C:0102 mov al,1

0C1C:0104 push ax

0C1C:0105 pop bx

0C1C:0106 add al,b1

0C1C:0108 add al,10

0C1C:010A mul al

0C1C:010C

-t

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0102 NV UP EI PL ZR NA PE NC

0C1C:0102 B001 MOV AL,01

-t

AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0104 NV UP EI PL ZR NA PE NC

0C1C:0104 50 PUSH AX

-t

AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEC BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0105 NV UP EI PL ZR NA PE NC

0C1C:0105 5B POP BX

-t

AX=0001 BX=0001 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0106 NV UP EI PL ZR NA PE NC

0C1C:0106 00D8 ADD AL,BL

-t

AX=0002 BX=0001 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=0108 NV UP EI PL NZ NA PO NC

0C1C:0108 0410 ADD AL,10

-t

AX=0012 BX=0001 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

```

DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=010A NV UP EI PL NZ NA PE NC
OC1C:010A F6E0 MUL AL
-t
AX=0144 BX=0001 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C1C ES=0C1C SS=0C1C CS=0C1C IP=010C OV UP EI PL NZ NA PE CY
OC1C:010C 1599CD ADC AX,CD99
-

```

## 检测点11.2

写出下面每条指令执行后，ZF、PF、SF、CF、OF等标志位的值。

	al	CF	OF	SF	ZF	PF
sub al, al	0h/0000 0000b	0	0	0	1	1
mov al, 10h	10h/0010 0000b	0	0	0	1	1
add al, 90h	a0h/1010 0000b	0	0	1	0	1
mov al, 80h	80h/1000 0000b	0	0	1	0	1
add al, 80h	0h/0000 0000b	1	1	0	1	1
mov al, 0fch	0fch/1111 1100b	1	1	0	1	1
add al, 05h	1h/0000 0001b	1	0	0	0	0
mov al, 7dh	7dh/1111 1101b	1	0	0	0	0
add al, 0bh	88h/1000 1000b	0	1	1	0	1

检测点涉及的相关内容：

ZF是flag的第6位，零标志位，记录指令执行后结果是否为0，结果为0时，ZF=1

PF是flag的第2位，奇偶标志位，记录指令执行后结果二进制数中1的个数是否为偶数，结果为偶数时，PF=1

SF是flag的第7位，符号标志位，记录有符号运算结果是否为负数，结果为负数时，SF=1

CF是flag的第0位，进位标志位，记录无符号运算结果是否有进/借位，结果有进/借位时，SF=1

OF是flag的第11位，溢出标志位，记录有符号运算结果是否溢出，结果溢出时，OF=1

add、sub、mul、div、inc、or、and等运算指令影响flag

mov、push、pop等传送指令对flag没影响

Microsoft(R) Windows DOS

(C)Copyright Microsoft Corp 1990-2001.

C:\DOCUME~1\SNUSER>debug

-a

0BF9:0100 sub al,al

0BF9:0102 mov al,10

0BF9:0104 add al,90

0BF9:0106 mov al,80

0BF9:0108 mov al,fc

0BF9:010A add al,5

0BF9:010C mov al,7d

0BF9:010E add al,b

0BF9:0110

```

-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0100 NV UP EI PL NZ NA PO NC
0BF9:0100 28C0 SUB AL, AL
-t
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0102 NV UP EI PL ZR NA PE NC
0BF9:0102 B010 MOV AL, 10
-t
AX=0010 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0104 NV UP EI PL ZR NA PE NC
0BF9:0104 0490 ADD AL, 90
-t
AX=00A0 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0106 NV UP EI NG NZ NA PE NC
0BF9:0106 B080 MOV AL, 80
-t
AX=0080 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0108 NV UP EI NG NZ NA PE NC
0BF9:0108 B0FC MOV AL, FC
-t
AX=00FC BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=010A NV UP EI NG NZ NA PE NC
0BF9:010A 0405 ADD AL, 05
-t
AX=0001 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=010C NV UP EI PL NZ AC PO CY
0BF9:010C B07D MOV AL, 7D
-t
AX=007D BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=010E NV UP EI PL NZ AC PO CY
0BF9:010E 040B ADD AL, 0B
-t
AX=0088 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0110 OV UP EI NG NZ AC PE NC
0BF9:0110 C6BF1F9903 MOV BYTE PTR [BX+991F], 03 DS:991F=00
-

```

### 检测点11.3

(1) 补全下面的程序，统计F000:0处32个字节中，大小在[32, 128]的数据个数。

```

mov ax, 0f000h
mov ds, ax
mov bx, 0          ;ds:bx指向第一个字节
mov dx, 0          ;初始化累加器

```

```

        mov cx, 32
s:      mov al, [bx]
        cmp al, 32          ;和32进行比较
        jnb s0              ;如果低于a1转到s0,继续循环
        cmp al, 128         ;和128进行比较
        ja s0               ;如果高于a1转到s0,继续循环
        inc dx
s0:     inc bx
        loop s

```

[32, 128]是闭区间, 包括两端点的值  
(32, 128)是开区间, 不包括两端点的值

### 检测点11.3

(2) 补全下面的程序, 统计F000:0处32个字节中, 大小在(32, 128)的数据个数。

```

        mov ax, 0f000h
        mov ds, ax
        mov bx, 0           ;ds:bx指向第一个字节
        mov dx, 0           ;初始化累加器
        mov cx, 32
s:      mov al, [bx]
        cmp al, 32          ;和32进行比较
        jna s0              ;如果不高于a1转到s0,继续循环
        cmp al, 128         ;和128进行比较
        jnb s0              ;如果不低于a1转到s0,继续循环
        inc dx
s0:     inc bx
        loop s

```

[32, 128]是闭区间, 包括两端点的值  
(32, 128)是开区间, 不包括两端点的值

### 检测点11.4

下面指令执行后, (ax)= 45h

```

mov ax, 0
push ax
popf
mov ax, 0fff0h
add ax, 0010h
pushf

```



```

pop ax
and al,11000101B
and ah,00001000B

```

推算过程:

popf后,标志寄存器中,本章节介绍的那些标志位都为0(但是此时标志寄存器并不是所有位置都为0,这个不用关心,没学过的位置用\*先代替),向下进行,那么pushf将计算后的当时状态的标志寄存器入栈,然后pop给ax,这是ax是寄存器的值(这个值中包含了我们的\*号),接下来就是对那些没有学过的标志位的屏蔽操作,这就是最后两条指令的意义所在,将不确定的位置都归0,那么只剩下我们能够确定的位置了,所以,结果就可以推理出来了。

```

mov ax,0
push ax
popf
mov ax,0fff0h
add ax,0010h
pushf
pop ax

```

	0	0	0	0	of	df	if	tf	sf	zf	0	af	0	pf	0	cf
	0	0	0	0	0	0	*	*	0	1	0	*	0	1	0	1

ax=flag=000000\*\* 010\*0101b

```

and al,11000101B    al=01000101b=45h
and ah,00001000B    ah=00000000b=0h

```

C:\DOCUME~1\SNUSER>debug

-a

```

0BF9:0100 mov ax,0
0BF9:0103 push ax
0BF9:0104 popf
0BF9:0105 mov ax,fff0
0BF9:0108 add ax,10
0BF9:010B pushf
0BF9:010C pop ax
0BF9:010D and al,c5
0BF9:010F and ah,8
0BF9:0112

```

-r

```

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0100 NV UP EI PL NZ NA PO NC
0BF9:0100 B80000 MOV AX,0000

```

-t

```

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0103 NV UP EI PL NZ NA PO NC
0BF9:0103 50 PUSH AX

```

-t

```

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEC BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0104 NV UP EI PL NZ NA PO NC
0BF9:0104 9D POPF

```

```

-t
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0105 NV UP DI PL NZ NA PO NC
0BF9:0105 B8F0FF MOV AX, FFF0
-t
AX=FFF0 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0108 NV UP DI PL NZ NA PO NC
0BF9:0108 051000 ADD AX, 0010
-t
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=010B NV UP DI PL ZR NA PE CY
0BF9:010B 9C PUSHF
-t
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEC BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=010C NV UP DI PL ZR NA PE CY
0BF9:010C 58 POP AX
-t
AX=3047 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=010D NV UP DI PL ZR NA PE CY
0BF9:010D 24C5 AND AL, C5
-t
AX=3045 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=010F NV UP DI PL NZ NA PO NC
0BF9:010F 80E408 AND AH, 08
-t
AX=0045 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0BF9 ES=0BF9 SS=0BF9 CS=0BF9 IP=0112 NV UP DI PL ZR NA PE NC
0BF9:0112 4C DEC SP

```

## 实验11（letterc子程序）小写改成大写

编写一个子程序，将包含任意字符，以0结尾的字符串的小写字母转变成大写字母。

```

C:\>debug syll-2.exe
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C53 IP=0000 NV UP EI PL NZ NA PO NC
0C53:0000 B04F0C MOV AX, 0C4F
-d 0c4f:0 3f
0C4F:0000 53 65 67 69 6E 6E 65 72-27 73 20 41 6C 6C 2D 70 Seginner's All-p
0C4F:0010 75 72 70 6F 73 65 20 53-79 6D 62 6F 6C 69 63 20 urpose Symbolic
0C4F:0020 49 6E 73 74 72 75 63 74-69 6F 6E 20 43 6F 64 65 Instruction Code
0C4F:0030 2E 30 00 00 00 00 00 00-00 00 00 00 00 00 00 .0.....
-g
Program terminated normally
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C53 IP=0000 NV UP EI PL NZ NA PO NC
0C53:0000 B04F0C MOV AX, 0C4F
-d 0c4f:0 3f
0C4F:0000 53 45 47 49 4E 4E 45 52-27 53 20 41 4C 4C 2D 50 BEGINNER'S ALL-P
0C4F:0010 55 52 50 4F 53 45 20 53-59 4D 42 4F 4C 49 43 20 URPOSE SYMBOLIC
0C4F:0020 49 4E 53 54 52 55 43 54-49 4F 4E 20 43 4F 44 45 INSTRUCTION CODE
0C4F:0030 2E 30 00 00 00 00 00 00-00 00 00 00 00 00 00 .0.....

```

效果图

程序一：此题为小写改成大写，根据书P141页介绍，小写字母'a'-'z'对应ASCII码为61h-86h，只要[61,86]这段区间里的ASCII减去20h，就改成了大写字母。

```
assume cs:codesg
datasg segment
    db "Seginner's All-purpose Symbolic Instruction Code.", '0'
datasg ends
codesg segment
begin:
    mov ax,datasg
    mov ds,ax
    mov si,0                ;ds:si指向第一个字节
    call letterc

    mov ax,4c00h
    int 21h

;名称: letterc
;功能: 将以0结尾的字符中的小写字母转变成大写字母
;参数: ds:si指向字符串首地址
letterc:push ax
        push si
let:    cmp byte ptr [si],0 ;和0进行比较
        je let0            ;如果等于0则转到let0，结束
        cmp byte ptr [si],61h ;和61h进行比较
        jb let1            ;如果低于60h则转到let1，继续循环
        cmp byte ptr [si],86h ;和86h进行比较
        ja let1            ;如果高于86h则转到let1，继续循环
        mov al,[si]
        sub al,20h          ;转为大写字母
        mov [si],al
let1:
        inc si
        jmp let
let0:
        pop si
        pop ax
        ret
codesg ends
end begin
```

-----华丽的分割线-----

程序二：参考书中P143页内容，有更好的办法，无需用到寄存器。  
可以用and直接修改内存，将ASCII码的第5位置为0，变为大写字母。

```
assume cs:codesg
```

```

datasg segment
    db "Seginner's All-purpose Symbolic Instruction Code.", '0'
datasg ends
codesg segment
begin: mov ax,datasg
        mov ds,ax
        mov si,0      ;ds:si指向第一个字节
        call letterc

        mov ax,4c00h
        int 21h

;名称: letterc
;功能: 将以0结尾的字符中的小写字母转变成大写字母
;参数: ds:si指向字符串首地址
letterc:push si
let:   cmp byte ptr [si],0 ;和0进行比较
        je let0           ;如果等于0则转到let0, 结束
        cmp byte ptr [si],61h ;和61h进行比较
        jb let1           ;如果低于60h则转到let1, 继续循环
        cmp byte ptr [si],86h ;和86h进行比较
        ja let1           ;如果高于86h则转到let1, 继续循环
        and byte ptr [si],11011111b ;ASCII码的第5位置为0, 转为大写
let1:
        inc si
        jmp let
let0:
        pop si
        ret
codesg ends
end begin

```

---

### 检测点12.1

(1)用debug查看内存, 情况如下:

0000:0000 68 10 A7 00 8B 01 70 00-16 00 9D 03 8B 01 70 00

则3号中断源对应的中断处理程序入口的偏移地址的内存单位的地址为: 0070:018b

检测点涉及相关内容:

一个表项存放一个中断向量, 也就是一个中断处理程序的入口地址, 这个入口地址包括段地址和偏移地址, 一个表项占两个字, 高地址存放段地址, 低地址存放偏移地址

---

## 检测点12.1

(2)

存储N号中断源对应的中断处理程序入口的偏移地址的内存单元的地址为: 4N

存储N号中断源对应的中断处理程序入口的段地址的内存单元的地址为: 4N+2

检测点涉及相关内容:

一个表项存放一个中断向量, 也就是一个中断处理程序的入口地址, 这个入口地址包括段地址和偏移地址, 一个表项占两个字, 高地址存放段地址, 低地址存放偏移地址

---

## 实验12 编写0号中断处理程序

编写0号中断处理程序, 使得在除法溢出发生时, 在屏幕中间显示字符串“divide error!”, 然后返回DOS

;名称: 0号中断处理程序

;功能: 使得除法溢出发生时, 在屏幕中间显示字符串'divide error!', 然后返回DOS

```
assume cs:code
```

```
code segment
```

```
start:  mov ax,cs
```

```
        mov ds,ax
```

```
        mov si,offset do0                ;设置ds:si指向源地址
```

```
        mov ax,0
```

```
        mov es,ax
```

```
        mov di,200h                    ;设置es:di指向目标地址
```

```
        mov cx,offset do0end-offset do0 ;设置cx为传输长度
```

```
        cld                            ;设置传输方向为正
```

```
        rep movsb
```

```
        mov ax,0
```

```
        mov es,ax
```

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

```
        mov word ptr es:[0*4+2],0      ;设置中断向量表
```

```
        mov ax,4c00h
```

```
        int 21h
```

```
do0:    jmp short do0start
```

```
        db 'divide error!'
```

```
do0start:mov ax,cs
```

```

        mov ds, ax
        mov si, 202h                ;设置ds:si指向字符串

        mov ax, 0b800h
        mov es, ax
        mov di, 12*160+34*2        ;设置es:di指向显存空间的中间位置

        mov cx, 13                  ;设置cx为字符串长度
s:      mov al, [si]
        mov es:[di], al
        inc si
        add di, 2
        loop s

        mov ax, 4c00h
        int 21h

do0end: nop
code ends
end start

```

---

### 检测点13.1

7ch中断例程如下：

```

lp:      push bp
        mov bp, sp
        dec cx
        jcxz lpret
        add [bp+2], bx
lpret:   pop bp
        iret

```

(1) 在上面的内容中，我们用7ch中断例程实现loop的功能，则上面的7ch中断例程所能进行的最大转移位移是多少？

最大位移是FFFFH

---

### 检测点13.1

(2) 用7ch中断例程完成jmp near ptr s指令功能，用bx向中断例程传送转移位移。

应用举例：在屏幕的第12行，显示data段中以0结尾的字符串。

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
s:    cmp byte ptr [si],0
        je ok
    mov al,[si]
    mov es:[di],al
    inc si
    add di,2
    mov bx,offset s-offset ok
    int 7ch
ok:    mov ax,4c00h
        int 21h
code ends
end start

```

jmp near ptr s指令的功能为:  $(ip)=(ip)+16$ 位移, 实现段内近转移

```

assume cs:code
code segment
start:
    mov ax,cs
    mov ds,ax
    mov si,offset do0                                ;设置ds:si指向源地址
    mov ax,0
    mov es,ax
    mov di,200h                                       ;设置es:di指向目标地址
    mov cx,offset do0end-offset do0                 ;设置cx为传输长度
    cld                                              ;设置传输方向为正
    rep movsb
    mov ax,0
    mov es,ax
    mov word ptr es:[7ch*4],200h
    mov word ptr es:[7ch*4+2],0                     ;设置中断向量表
    mov ax,4c00h
    int 21h
do0:
    push bp
    mov bp,sp

```

```
        add [bp+2],bx                ;ok的偏移地址+bx得到s的偏移地址
    pop bp
    iret
    mov ax,4c00h
    int 21h
do0end:
    nop
code ends
end start
```

---

### 检测点13.2

判断下面说法的正误:

(1) 我们可以编程改变FFFF:0处的指令,使得CPU不去执行BIOS中的硬件系统检测和初始化程序。

答: 错误, FFFF:0处的内容无法改变。

---

### 检测点13.2

判断下面说法的正误:

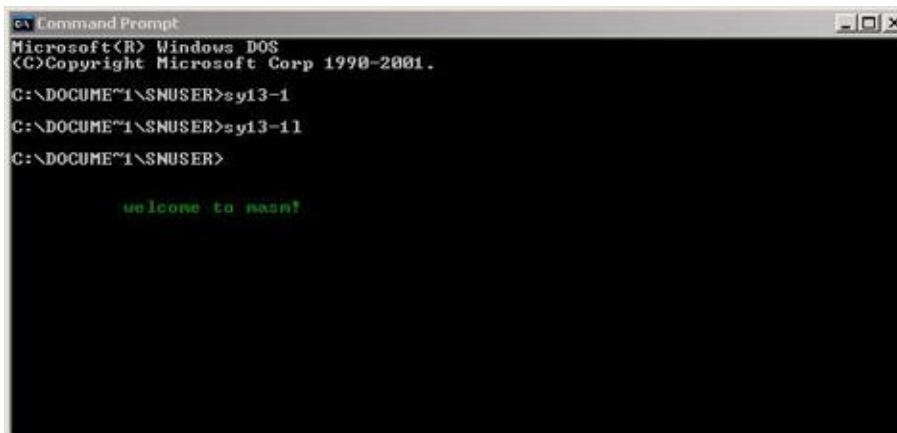
(2) int 19h中断例程,可以由DOS提供。

答: 错误, 先调用int 19h, 后启动DOS。

---

### 实验13

(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。





效果图

```
;名称: int 7ch中断例程
;功能: 显示一个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
assume cs:code
code segment
start:  mov ax,cs
        mov ds,ax
        mov si,offset show_str          ;设置ds:si指向源地址
        mov ax,0
        mov es,ax
        mov di,200h                     ;设置es:di指向目标地址
        mov cx,offset show_strend-offset show_str ;设置cx为传输长度
        cld                             ;设置传输方向为正
        rep movsb
        mov ax,0
        mov es,ax
        mov word ptr es:[7ch*4],200h
        mov word ptr es:[7ch*4+2],0      ;设置中断向量表
        mov ax,4c00h
        int 21h
;名称: show_str
;功能: 在指定的位置, 用指定的颜色, 显示一个用0结束的字符串。
;参数: (dh)=行号(取值范围0~24);
;       (dl)=列号(取值范围0~79);
;       (cl)=颜色;
;       ds:si指向字符串的首地址。
;返回: 无。
show_str:push ax
        push bx
        push es
        push si
        mov ax,0b800h
        mov es,ax
        mov ax,160
        mul dh
        mov bx,ax          ;bx=160*dh
        mov ax,2
        mul dl              ;ax=dl*2
        add bx,ax           ;mov bx, (160*dh+dl*2) 设置es:bx指向显存首地址
        mov al,cl           ;把颜色cl赋值al
        mov cl,0
show0:  mov ch,[si]
        jcxz show1         ;(ds:si)=0时, 转到show1执行
        mov es:[bx],ch
        mov es:[bx].1,al
```

```

        inc si          ;ds:si指向下一个字符地址
        add bx,2        ;es:bx指向下一个显存地址
        jmp show0
show1:  pop si
        pop es
        pop bx
        pop ax
        iret
        mov ax,4c00h
        int 21h
show_strend:nop
code ends
end start

```

;实验13(1)应用举例

```

assume cs:code
data segment
    db 'welcome to masm!',0
data ends
code segment
start: mov dh,10
        mov dl,10
        mov cl,2
        mov ax,data
        mov ds,ax
        mov si,0
        int 7ch
        mov ax,4c00h
        int 21h
code ends
end start

```

---

### 实验13

(2) 编写并安装int 7ch中断例程，功能为完成loop指令功能



;实验13(2)应用举例

```
assume cs:code
```

```
code segment
```

```
start:  mov ax,0b800h
```

```
        mov es,ax
```

```
        mov di,160*12
```

```
        mov bx,offset s-offset se ;设置从标号se到标号s的转移位移
```

```
        mov cx,80
```

```
s:      mov byte ptr es:[di],','
```

```
        add di,2
```

```
        int 7ch ;(cx)≠0, 转移到标号s处
```

```
se:     nop
```

```
        mov ax,4c00h
```

```
        int 21h
```

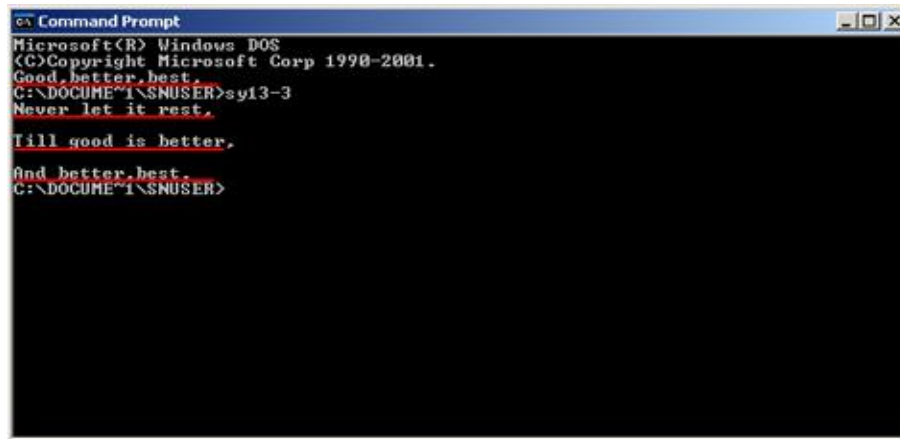
```
code ends
```

```
end start
```

---

### 实验13

(3)下面的程序，分别在屏幕的第2、4、6、8行显示4句英文诗，补全代码。



效果图

```
assume cs:code
```

```
code segment
```

```
s1:  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          ;第0页
        mov dh, [si]       ;dh中放行号
        mov dl, 0          ;dl中放列号
        mov ah, 2          ;置光标
        int 10h

        mov dx, [bx]       ;ds:dx指向字符串首地址
        mov ah, 9          ;在光标位置显示字符
        int 21h
        inc si              ;行号递增
        add bx, 2           ;指向下一字符串
        loop ok

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

```

---

#### 检测点14.1 读取写入CMOS RAM单元内容

(1) 编程，读取CMOS RAM的2号单元内容。

```

assume cs:code
code segment
start:  mov al, 2          ;赋值al
        out 70h, al        ;将al送入端口70h
        in al, 71h         ;从端口71h处读出单元内容
        mov ax, 4c00h
        int 21h
code ends
end start

```

---

#### 检测点14.1

(2) 编程，向CMOS RAM的2号单元写入0。

```

assume cs:code
code segment
start:  mov al, 2          ;赋值al

```

```

    out 70h, al      ;将al送入端口70h
    mov al, 0        ;赋值al
    out 71h, al      ;向端口71h写入数据al
    mov ax, 4c00h
    int 21h
code ends
end start

```

## 检测点14.2 用加法和移位指令计算

```

LINK : warning L4021: no stack segment
C:\DOCUMENT1\SNUSER>debug jc14-2.exe
-u
0C4F:0000 B8FF00      MOV     AX,00FF
0C4F:0003 8BD0          MOV     BX,AX
0C4F:0005 D1E0          SHL     AX,1
0C4F:0007 B103          MOV     CL,03
0C4F:0009 D3E3          SHL     BX,CL
0C4F:000B 03C3          ADD     AX,BX
0C4F:000D B8004C      MOV     AX,4C00
0C4F:0010 CD21          INT     21
0C4F:0012 86FE          XCHG    BH,DH
0C4F:0014 FE00          INC     BYTE PTR [BX+SI]
0C4F:0016 00EB          ADD     BL,CH
0C4F:0018 0590FF      ADD     AX,FF90
0C4F:001B 86FE          XCHG    BH,DH
0C4F:001D FEA15607     JMP     [BX+DI+0756]
-g000d
ax= (0ffh)+10=9f0h
AX=09F6 BX=07FB CX=0003 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3F ES=0C3F SS=0C4F CS=0C4F IP=000D  NU UP EI PL NZ AC PE NC
0C4F:000D B8004C      MOV     AX,4C00

```

效果图

编程，用加法和移位指令计算  $(ax) = (ax) * 10$

提示：  $(ax) * 10 = (ax) * 2 + (ax) * 8$

```

assume cs:code
code segment
start: mov bx, ax
      shl ax, 1      ;左移1位 (ax)=(ax)*2
      mov cl, 3
      shl bx, cl     ;左移3位 (bx)=(ax)*8
      add ax, bx      ;(ax)=(ax)*2+(ax)*8
      mov ax, 4c00h
      int 21h
code ends
end start

```

;应用举例：计算  $ffh * 10$

```

assume cs:code
code segment
start: mov ax, 0ffh
      mov bx, ax
      shl ax, 1      ;左移1位 (ax)=(ax)*2

```

```

        mov cl,3
        shl bx,cl      ;左移3位 (bx)=(ax)*8
        add ax,bx      ;(ax)=(ax)*2+(ax)*8
        mov ax,4c00h
        int 21h
code ends
end start

```

PS:

左移1位,  $N=(N)*2$   
 左移2位,  $N=(N)*4$   
 左移3位,  $N=(N)*8$   
 左移4位,  $N=(N)*16$   
 左移5位,  $N=(N)*32$

#### 实验14 访问CMOS RAM

编程,以“年/月/日 时:分:秒”的格式,显示当前的日期、时间。(两个程序)

```

C:\DOCUME~1\ADMINI~1>nasm cs;
Microsoft (R) Macro Assembler Version 5.08
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.

50698 + 445718 Bytes symbol space free

0 Warning Errors
0 Severe Errors

C:\DOCUME~1\ADMINI~1>link cs; 08/07/03 17:36:39
C:\DOCUME~1\ADMINI~1>
Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

LINK : warning L4021: no stack segment
C:\DOCUME~1\ADMINI~1>cs

```

效果图

程序一:

```

assume cs:code
code segment
start: mov ax,0
        mov al,9
        mov si,0
        mov cx,6
s:      push cx
        push ax
        out 70h,al      ;将al送入端口70h
        in al,71h       ;从端口71h处读出单元内容
        mov ah,al
        mov cl,4

```

```

        shr ah, cl
        and al, 00001111b    ;a1分成两个表示BCD码值的数据
        add ah, 30h
        add al, 30h          ;BCD码+30h=10进制数对应的ASCII码
        mov bx, 0b800h
        mov es, bx
        mov byte ptr es:[160*12+40*2][si], ah    ;显示十位数码
        mov byte ptr es:[160*12+40*2+2][si], al   ;显示个位数码
        pop ax
        dec ax                ;指向前一数据单元
        jmp s1
s0:     pop cx
        add si, 6
        loop s
        mov ax, 4c00h
        int 21h
s1:     cmp ax, 10
        ja s0
        cmp ax, 0
        je s0
        cmp ax, 6
        ja s2                ;ax>6, 为年/月/日
        je s3                ;ax=6, 为日结尾
        jb s4                ;ax<6, 为时:分:秒
s2:     mov byte ptr es:[160*12+40*2+4][si], '/'   ;添加 '/'
        jmp s0
s3:     sub ax, 2
        jmp s0
s4:     sub ax, 1
        mov byte ptr es:[160*12+40*2+4][si], ':'   ;添加 ':'
        jmp s0
code ends
end start

```

=====华丽的分割线=====

程序二:

```

assume cs:code
code segment
time    db 'yy/mm/dd hh:mm:ss', '$'
cmos    db 9, 8, 7, 4, 2, 0
start:  mov ax, cs
        mov ds, ax
        mov bx, 0
        mov si, 0
        mov cx, 6
a:      push cx

```



```

    mov al, cmos[bx]
    out 70h, al    ;将al送入地址端口70h
    in al, 71h     ;从数据端口71h处读出单元内容
    mov ah, al
    mov cl, 4
    shr al, cl     ;右移4位
    and ah, 0fh    ;al分成两个表示BCD码值的数据
    add ax, 3030h  ;BCD码+30h=10进制数对应的ASCII码
    mov cs:[si], ax    ;ASCII码写入time段
    inc bx
    add si, 3
    pop cx
    loop a
;名称: BIOS中断(int 10h)
;功能: (ah)=2置光标到屏幕指定位置、(ah)=9在光标位置显示字符
;参数: (al)=字符、(bh)=页数、(dh)=行号、(dl)=例号
;      (bl)=颜色属性、(cx)=字符重复个数
    mov ah, 2      ;置光标
    mov bh, 0      ;第0页
    mov dh, 13     ;dh中放行号
    mov dl, 32     ;dl中放例号
    int 10h
;名称: DOS中断(int 21h)
;功能: (ah)=9显示用'$'结束的字符串、(ah)=4ch程序返回
;参数: ds:dx指向字符串、(al)=返回值
    mov dx, 0
    mov ah, 9
    int 21h
;结束
    mov ax, 4c00h
    int 21h
code ends
end start

```

#### 检测点15.1

(1) 仔细分析一下书中的in9中断例程，看看是否可以精简一下？

其实在我们的int 9中断例程中，模拟int指令调用原int 9中断例程的程序段是可以精简的，因为在进入中断例程后，IF和TF都已置0，没有必要再进行设置了，对于程序段：

```

    pushf          ;标志寄存器入栈
    pushf
    pop bx
    and bh, 11111100b    ;IF和TF为flag的第9位和第8位
    push bx

```

```

    popf      ;TF=0, IF=0
    call dword ptr ds:[0]      ;CS、IP入栈; (IP)=ds:[0], (CS)=ds:[2]

```

可以精简为:

```

    pushf      ;标志寄存器入栈
    call dword ptr ds:[0] ;CS、IP入栈; (IP)=ds:[0], (CS)=ds:[2]

```

两条指令。

#### 检测点15.1

(2) 仔细分析程序中的主程序，看看有什么潜在的问题？

在主程序中，如果在设置执行设置int 9中断例程的段地址和偏移地址的指令之间发生了键盘中断，则CPU将转去一个错误的地址执行，将发生错误。

找出这样的程序段，改写他们，排除潜在的问题。

```

;在中断向量表中设置新的int 9中断例程的入口地址
cli      ;设置IF=0屏蔽中断
mov word ptr es:[9*4],offset int9
mov es:[9*4+2],cs
sti      ;设置IF=1不屏蔽中断

```

=====更改后的int 9中断例程=====

;功能：在屏幕中间依次显示'a'~'z'，并让人看清。在显示过程中按下Esc键后，改变显示的颜色。

```

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

```

;将原来的int 9中断例程的入口地址保存在ds:0、ds:2单元中

```

    mov ax,data
    mov ds,ax

    mov ax,0
    mov es,ax

    push es:[9*4]

```

```

    pop ds:[0]
    push es:[9*4+2]
    pop ds:[2]

;在中断向量表中设置新的int 9中断例程的入口地址
    cli                    ;设置IF=0屏蔽中断
    mov word ptr es:[9*4],offset int9
    mov word ptr es:[9*4+2],cs
    sti                    ;设置IF=1不屏蔽中断

;依次显示'a'~'z'
    mov ax,0b800h
    mov es,ax
    mov ah,'a'
s:    mov es:[160*12+40*2],ah    ;第12行第40列
    inc ah
    cmp ah,'z'
    jnb s

;将中断向量表中int 9中断例程的入口恢复为原来的地址
    mov ax,0
    mov es,ax

    push ds:[0]
    pop ss:[9*4]
    push ds:[2]
    pop es:[9*4+2]

;结束
    mov ax,4c00h
    int 21h

;循环延时，循环100000h次
delay: push ax
    push dx
    mov dx,1000h
    mov ax,0
delay1: sub ax,1
    sbb dx,0                ;(dx)=(dx)-0-CF
    cmp ax,0
    jne delay1
    cmp dx,0
    jne delay1
    pop dx
    pop ax
    ret

```

```

;以下为新的int 9中断例程
int9:  push ax
        push bx
        push es

        in  al,60h          ;从端口60h读出键盘输入

;对int指令进行模拟，调用原来的int 9中断例程
        pushf               ;标志寄存器入栈
        call dword ptr ds:[0] ;CS、IP入栈;(IP)=ds:[0], (CS)=ds:[2]

;如果是ESC扫描码，改变显示颜色
        cmp al,1            ;和esc的扫描码01比较
        jne int9ret         ;不等于esc时转移

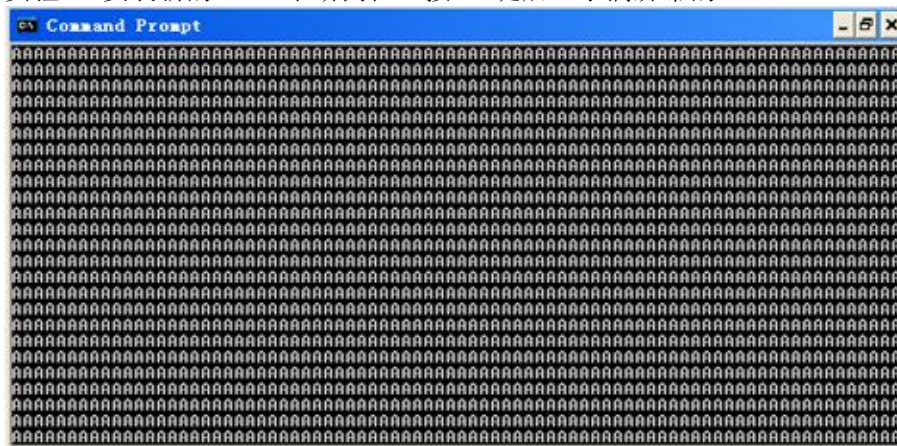
        mov ax,0b800h
        mov es,ax
        inc byte ptr es:[160*12+40*2+1] ;将属性值+1，改变颜色

int9ret:pop es
        pop bx
        pop ax
        iret

code ends
end start

```

实验15 安装新的int 9中断例程（按'A'键后显示满屏幕的'A'）



效果图

;任务：安装一个新的int 9中断例程  
 ;功能：在DOS下，按'A'键后除非不再松开，如果松开，就显示满屏幕的'A'，其他键照常处理

```

assume cs:code
stack segment
    db 128 dup (0)
stack ends
code segment
start: mov ax,stack
        mov ss,ax
        mov sp,128
        push cs
        pop ds
        mov ax,0
        mov es,ax
        mov si,offset int9          ;设置ds:si指向源地址
        mov di,204h                 ;设置es:di指向目标地址
        mov cx,offset int9end-offset int9 ;设置cx为传输长度
        cld                         ;设置传输方向为正
        rep movsb
;将原来的int 9中断例程的入口地址保存在ds:200、ds:202单元中
        push es:[9*4]
        pop es:[200h]
        push es:[9*4+2]
        pop es:[202h]
;在中断向量表中设置新的int 9中断例程的入口地址
        cli                         ;设置IF=0屏蔽中断
        mov word ptr es:[9*4],204h
        mov word ptr es:[9*4+2],0
        sti                         ;设置IF=1不屏蔽中断
;结束
        mov ax,4c00h
        int 21h
;新的int 9中断例程
int9:  push ax
        push bx
        push cx
        push es
        in al,60h                   ;从端口60h读出键盘输入
;对int指令进行模拟，调用原来的int 9中断例程
        pushf                       ;标志寄存器入栈
        call dword ptr cs:[200h]     ;CS, IP入栈, (IP)=cs:[200h], (CS)=0
;如果是A断码，改变当前屏幕的显示字符
        cmp al,9eh                  ;和A的断码(1eh+80h)比较
        jne int9ret                  ;不等于A时转移
        mov ax,0b800h
        mov es,ax
        mov bx,0
        mov cx,2000
s:      mov byte ptr es:[bx],41h     ;将A的ASCII码写入，改变字符

```

```

        add bx,2
        loop s
int9ret:pop es
        pop cx
        pop bx
        pop ax
        iret
int9end:nop
code ends
end start

```

#### 检测点16.1（两个程序）

下面的程序将code段中a处的8个数值累加，结果存储到b处的双字节中，补全程序。

程序一：

```

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 a[16],ax
        adc a[18],0
        add si,2
        loop s
        mov ax,4c00h
        int 21h
code ends
end start

```

C:\DOCUME~1\SNUSER>debug 16-1.exe

~u

0C4E:0014	BE0000	MOV	SI,0000
0C4E:0017	B90800	MOV	CX,0008
0C4E:001A	2E	CS:	
0C4E:001B	8B840000	MOV	AX,[SI+0000]
0C4E:001F	2E	CS:	
0C4E:0020	01061000	ADD	[0010],AX
0C4E:0024	2E	CS:	
0C4E:0025	8316120000	ADC	WORD PTR [0012],+00
0C4E:002A	83C602	ADD	SI,+02
0C4E:002D	E2EB	LOOP	001A
0C4E:002F	B8004C	MOV	AX,4C00
0C4E:0032	CD21	INT	21

```

-r
AX=0000 BX=0000 CX=0034 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3E ES=0C3E SS=0C4E CS=0C4E IP=0014 NV UP EI PL NZ NA PO NC
0C4E:0014 BE0000 MOV SI,0000
-d 0c4e:0 1f
0C4E:0000 01 00 02 00 03 00 04 00-05 00 06 00 07 00 08 00 .....
0C4E:0010 00 00 00 00 BE 00 00 B9-08 00 2E 8B 84 00 00 2E .....
-g002f
AX=0008 BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=0010 DI=0000
DS=0C3E ES=0C3E SS=0C4E CS=0C4E IP=002F NV UP EI PL NZ AC PO NC
0C4E:002F B8004C MOV AX,4C00
-d 0c4e:0 1f
0C4E:0000 01 00 02 00 03 00 04 00-05 00 06 00 07 00 08 00 .....
0C4E:0010 24 00 00 00 BE 00 00 B9-08 00 2E 8B 84 00 00 2E $......
-

```

程序二:

```

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

下面的程序将data段中a处的8个数值累加，结果存储到b处的双字节中，补全程序。

```

assume cs:code,es:data

data segment
    a db 1,2,3,4,5,6,7,8
    b dw 0
data ends

```

```

code segment
start: mov ax,data
      mov es,ax
      mov si,0
      mov cx,8
s:     mov al,a[si]
      mov ah,0
      add b,ax
      inc si
      loop s

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

```

C:\DOCUME~1\SNUSER>debug 16-2.exe

~u

```

0C4F:0000 B84E0C      MOV     AX,0C4E
0C4F:0003 8EC0        MOV     ES,AX
0C4F:0005 BE0000      MOV     SI,0000
0C4F:0008 B90800      MOV     CX,0008
0C4F:000B 26           ES:
0C4F:000C 8A840000     MOV     AL,[SI+0000]
0C4F:0010 B400        MOV     AH,00
0C4F:0012 26           ES:
0C4F:0013 01060800     ADD     [0008],AX
0C4F:0017 46           INC     SI
0C4F:0018 E2F1        LOOP    000B
0C4F:001A B8004C      MOV     AX,4C00
0C4F:001D CD21      INT     21
0C4F:001F 56           PUSH    SI

```

~r

```

AX=0000 BX=0000 CX=002F DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0C3E ES=0C3E SS=0C4E CS=0C4F IP=0000 NV UP EI PL NZ NA PO NC
0C4F:0000 B84E0C      MOV     AX,0C4E

```

-d 0c4e:0 f

```

0C4E:0000 01 02 03 04 05 06 07 08-00 00 00 00 00 00 00 00 .....

```

-g001a

```

AX=0008 BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=0008 DI=0000
DS=0C3E ES=0C4E SS=0C4E CS=0C4F IP=001A NV UP EI PL NZ NA PO NC
0C4F:001A B8004C      MOV     AX,4C00

```

-d 0c4e:0 f

```

0C4E:0000 01 02 03 04 05 06 07 08-24 00 00 00 00 00 00 00 .....$.

```

-



---

实验16 包含多个功能子程序的中断例程（两个程序）

安装一个新的int 7ch中断例程，为显示输出提供功能子程序。

- (1) 清屏；
- (2) 设置前景色；
- (3) 设置背景色；
- (4) 向上滚动一行；

入口参数说明如下。

(1) 用ah寄存器传递功能号：0表示清屏，1表示设置前景色，2表示设置背景色，3表示向上滚动一行；

(2) 对于2、3号功能，用al传送颜色值， $(al) \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ 。

程序一：

；任务：安装一个新的int 7ch中断例程

；功能：为显示输出提供功能子程序

；参数：ah传递功能号；0表示清屏，1表示设置前景色，2表示设置背景色，3表示向上滚动一行，对于2、3号功能，用al传送颜色值， $(al) \in \{0, 1, 2, 3, 4, 5, 6, 7\}$

assume cs:code

code segment

start: mov ax, cs

mov ds, ax

mov si, offset s ;设置ds:si指向源地址

mov ax, 0

mov es, ax

mov di, 200h ;设置es:di指向目标地址

mov cx, offset send-offset s ;设置cx为传输长度

cld ;设置传输方向为正

rep movsb

；在中断向量表中设置新的int 7ch中断例程的入口地址

cli ;设置IF=0屏蔽中断

mov word ptr es:[7ch\*4], 200h

mov word ptr es:[7ch\*4+2], 0

sti ;设置IF=1不屏蔽中断

；结束

mov ax, 4c00h

int 21h

；新的int 7ch中断例程

s: jmp short set

table dw sub1-s+200h, sub2-s+200h, sub3-s+200h, sub4-s+200h

set: push bx

cmp ah, 3 ;判断ah中的功能号是否大于3

ja sret

mov bl, ah

mov bh, 0

```

        add bx, bx                ;计算对应子程序在table表中的偏移
        call word ptr table[bx]  ;调用对应的功能子程序
sret:   pop bx
        iret
;功能: 清屏
sub1:   push bx
        push cx
        push es
        mov bx, 0b800h
        mov es, bx
        mov bx, 0
        mov cx, 2000
subles:      mov byte ptr es:[bx], ' '
        add bx, 2
        loop subles
        pop es
        pop cx
        pop bx
        ret
;设置前景色, a1传送颜色值, (a1) ∈ {0, 1, 2, 3, 4, 5, 6, 7}
sub2:   push bx
        push cx
        push es
        mov bx, 0b800h
        mov es, bx
        mov bx, 1
        mov cx, 2000
sub2s:  and byte ptr es:[bx], 11111000b
        or es:[bx], a1
        add bx, 2
        loop sub2s
        pop es
        pop cx
        pop bx
        ret
;设置背景色, a1传送颜色值, (a1) ∈ {0, 1, 2, 3, 4, 5, 6, 7}
sub3:   push ax
        push bx
        push cx
        push es
        mov cl, 4
        shl a1, cl
        mov bx, 0b800h
        mov es, bx
        mov bx, 1
        mov cx, 2000
sub3s:  and byte ptr es:[bx], 10001111b

```

```

        or es:[bx],al
        add bx,2
        loop sub3s
        pop es
        pop cx
        pop bx
        pop ax
        ret
;向上滚动一行
sub4:  push cx
        push si
        push di
        push es
        push ds
        mov si,0b800h
        mov es,si
        mov ds,si
        mov si,160                ;ds:si指向第N+1行
        mov di,0                  ;es:di指向第N行
        cld
        mov cx,24                ;共复制24行
sub4s: push cx
        mov cx,160
        rep movsb                ;复制
        pop cx
        loop sub4s
        mov cx,80
sub4s1:      mov byte ptr [160*24+si],'' ;最后一行清空
        add bx,2
        loop sub4s1
        pop ds
        pop es
        pop di
        pop si
        pop cx
        ret
send:  nop
code ends
end start

```

=====

程序二：

说明：

在进行这个实验时，往往会参考16.4给出的代码(四个子程序和直接定址表功能选择程序)如果安装16.4给出的功能子程序的安装程序习惯采用下面图1安装结构会出问题，问题有两个：

1: table dw sub1……中SUB1存放的应该是子程序的偏移地址，这个偏移地址是相对于中断程序入口的，而此时Sub1中存放的是相对于安装程序入口的地址。所以应该将被安装代码放到code段标号Start前边。

2: 中断程序被调用时，table[bx]默认的段寄存器为cs，所以在安装程序中设置中断向量表时，应设：cs=200h，ip=0，而不是cs=0，ip=200h

```

    assume cs:code
    code segment
        start:
            安装程序

            ; -----
            被安装代码:
            ; -----
    code ends
end start

```

== 图1 ==

---

;任务：安装一个新的int 7ch中断例程

;功能：为显示输出提供功能子程序

;参数：ah传递功能号；0表示清屏，1表示设置前景色，2表示设置背景色，3表示向上滚动一行，对于2、3号功能，用al传送颜色值，(al) ∈ {0, 1, 2, 3, 4, 5, 6, 7}

```

assume cs:code
code segment
;新的int 7ch中断例程
s:    jmp short set
table dw sub1, sub2, sub3, sub4
set:  push bx
      cmp ah, 3                ;判断ah中的功能号是否大于3
      ja sret
      mov bl, ah
      mov bh, 0
      add bx, bx                ;计算对应子程序在table表中的偏移
      call word ptr table[bx]  ;调用对应的功能子程序
sret: pop bx
      iret
;功能：清屏
sub1: push bx
      push cx
      push es
      mov bx, 0b800h
      mov es, bx
      mov bx, 0
      mov cx, 2000
subles: mov byte ptr es:[bx], ' '
        add bx, 2

```

```

        loop subles
        pop es
        pop cx
        pop bx
        ret
;设置前景色, a1传送颜色值, (a1) ∈ {0, 1, 2, 3, 4, 5, 6, 7}
sub2:  push bx
        push cx
        push es
        mov bx, 0b800h
        mov es, bx
        mov bx, 1
        mov cx, 2000
sub2s: and byte ptr es:[bx], 11111000b
        or es:[bx], a1
        add bx, 2
        loop sub2s
        pop es
        pop cx
        pop bx
        ret
;设置背景色, a1传送颜色值, (a1) ∈ {0, 1, 2, 3, 4, 5, 6, 7}
sub3:  push ax
        push bx
        push cx
        push es
        mov cl, 4
        shl al, cl
        mov bx, 0b800h
        mov es, bx
        mov bx, 1
        mov cx, 2000
sub3s: and byte ptr es:[bx], 10001111b
        or es:[bx], a1
        add bx, 2
        loop sub3s
        pop es
        pop cx
        pop bx
        pop ax
        ret
;向上滚动一行
sub4:  push cx
        push si
        push di
        push es
        push ds

```

```

        mov si,0b800h
        mov es,si
        mov ds,si
        mov si,160                ;ds:si指向第N+1行
        mov di,0                  ;es:di指向第N行
        cld
        mov cx,24                  ;共复制24行
sub4s:  push cx
        mov cx,160
        rep movsb                  ;复制
        pop cx
        loop sub4s
        mov cx,80
sub4s1:  mov byte ptr [160*24+si],'' ;最后一行清空
        add bx,2
        loop sub4s1
        pop ds
        pop es
        pop di
        pop si
        pop cx
        ret
send:    nop
start:   mov ax,cs
        mov ds,ax
        mov si,offset s            ;设置ds:si指向源地址
        mov ax,0
        mov es,ax
        mov di,200h                ;设置es:di指向目标地址
        mov cx,offset send-offset s ;设置cx为传输长度
        cld                        ;设置传输方向为正
        rep movsb
;在中断向量表中设置新的int 7ch中断例程的入口地址
        cli                        ;设置IF=0屏蔽中断
        mov word ptr es:[7ch*4],0
        mov word ptr es:[7ch*4+2],20h
        sti                        ;设置IF=1不屏蔽中断
;结束
        mov ax,4c00h
        int 21h

code ends
end start

```

=====

应用举例测试程序

```

assume cs:code
code segment
start: mov ax,303h
        int 7ch
        mov ax,4c00h
        int 21h
code ends
end start

```

---

### 17.3 字符串的输入（非课后习题，仅为书中程序注释，实验用途）

```

assume cs:code
code segment
start: mov ax,2000h
        mov ds,ax
        mov si,0
        mov dl,0
        mov dh,0
        call getstr
        mov ax,4c00h
        int 21h

```

;子程序：接收字符串输入。

```
getstr:      push ax
```

```

getstrs:mov ah,0
        int 16h
        cmp al,20h
        jb nochar          ;ASCII码小于20h，说明不是字符
        mov ah,0
        call charstack      ;字符入栈
        mov ah,2
        call charstack      ;显示栈中字符
        jmp getstrs

```

```

nochar:      cmp ah,0eh      ;退格键的扫描码
        je backspace
        cmp ah,1ch          ;Enter键的扫描码
        je enter
        jmp getstrs

```

```

backspace:mov ah,1
        call charstack      ;字符出栈
        mov ah,2

```

```

        call charstack          ;显示栈中字符
        jmp getstrs

enter:  mov al,0
        mov ah,0
        call charstack          ;0入栈
        mov ah,2
        call charstack          ;显示栈中字符
        pop ax
        ret

;子程序：字符栈的入栈、出栈和显示。
;参数说明：(ah)=功能号，0表示入栈，1表示出栈，2表示显示；
;           ds:si指向字符栈空间；
;           对于0号功能：(al)=入栈字符；
;           对于1号功能：(al)=返回的字符；
;           对于2号功能：(dh)、(dl)=字符串在屏幕上显示的行、列位置。
charstack: jmp short charstart

table  dw charpush, charpop, charshow
top     dw 0                    ;栈顶(字符地址、个数记录器)

charstart: push bx
          push dx
          push di
          push es

          cmp ah,2              ;判断ah中的功能号是否大于2
          ja sret                ;功能号>2，结束
          mov bl,ah
          mov bh,0
          add bx,bx              ;计算对应子程序在table表中的偏移
          jmp word ptr table[bx] ;调用对应的功能子程序

charpush: mov bx,top
          mov [si][bx],al
          inc top
          jmp sret

charpop:  cmp top,0
          je sret                ;栈顶为0(无字符)，结束
          dec top
          mov bx,top            ;//保存数据，其它作用不详
          mov al,[si][bx]       ;//保存数据，其它作用不详
          jmp sret

charshow: mov bx,0b800h

```



```

    mov es, bx
    mov al, 160
    mov ah, 0
    mul dh          ; dh*160
    mov di, ax
    add dl, dl      ; dl*2
    mov dh, 0
    add di, dx      ; di=dh*160+dl*2, es:di指向显存

    mov bx, 0      ; ds:[si+bx]指向字符串首地址

charshows: cmp bx, top      ; 判断字符栈中字符是否全部显示完毕
            jne noempty     ; top≠bx, 有未显示字符, 执行显示
            mov byte ptr es:[di], ' ' ; 显示完毕, 字符串末加空格
            jmp sret
noempty: mov al, [si][bx]    ; 字符ASCII码赋值al
            mov es:[di], al  ; 显示字符
            mov byte ptr es:[di+2], ' ' ; 字符串末加空格
            inc bx          ; 指向下一个字符
            add di, 2        ; 指向下一显存单元
            jmp charshows

sret:  pop es
       pop di
       pop dx
       pop bx
       ret

code ends
end start

```

---

### 检测点17.1

“在int 16h中断例程中，一定有设置IF=1的指令。”这种说法对吗？

正确，当键盘缓冲区为空时，如果设置IF=0，int 9中断无法执行，循环等待会死锁。

相关内容：

IF=1，CPU响应中断，引发中断过程

IF=0，不响应可屏蔽中断

几乎所有由外设引发的外中断，都是可屏蔽中断（int 9是可屏蔽中断）

CPU对外设输入的通常处理方法:

- (1) 外设的输入端口
- (2) 向CPU发出外中断(可屏蔽中断)信息
- (3) CPU检测到可屏蔽中断信息, 如果IF=1, cpu在执行完当前指令后响应中断, 执行相应的中断例程
- (4) 可在中断例程中实现对外设输入的处理

---

#### 实验17 通过逻辑扇区号对软盘进行读写(未调试, 留底稿)

;功能: 安装新的int 7ch中断例程, 实现通过逻辑扇区号对软盘进行读写

;入口参数:

; (dx)=读写扇区的逻辑扇区号  
; (ah)=0表示读扇区, (ah)=1表示写扇区  
; es:bx指向存储读出数据或写入数据的内存区

;返回参数:

; (ah)=int 13h的功能号(2表示读扇区, 3表示写扇区)  
; (ch)=磁道号  
; (cl)=扇区号  
; (dh)=磁头号(面号)

assume cs:code

code segment

```
start: mov ax, cs
      mov ds, ax
      mov si, offset s                ;设置ds:si指向源地址
      mov ax, 0
      mov es, ax
      mov di, 200h                   ;设置es:di指向目标地址
      mov cx, offset send-offset s ;设置cx为传输长度
      cld                            ;设置传输方向为正
      rep movsb
```

;在中断向量表中设置新的int 7ch中断例程的入口地址

```
cli                                ;设置IF=0屏蔽中断
mov word ptr es:[7ch*4], 200h
mov word ptr es:[7ch*4+2], 0
sti                                ;设置IF=1不屏蔽中断
```

;结束

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

;新的int 7ch中断例程

```
s:   cmp ah, 0
      je s1
      cmp ah, 1
      je s2
      ja sret                ;ah大于1, 结束
```

```

        cmp dx, 1443
        ja sret          ;逻辑扇区号大于1443，结束

s3:     push bx
        mov ax, dx       ;逻辑扇区号赋值ax
        mov dx, 0
        mov cx, 1440
        div cx           ;商在ax；余在dx
        push ax          ;ax=面号=int(逻辑扇区号/1440)，入栈

        mov ax, dx
        mov cx, 18
        div cx           ;商在ax；余在dx
        push ax          ;ax=磁道号=int(rem(逻辑扇区号/1440)/18)，入栈

        add dx, 1        ;dx=扇区号=rem(rem(逻辑扇区号/1440)/18)+1

        mov cl, dl       ;扇区号赋值cl
        pop bx           ;磁道号出栈
        mov ch, bl
        pop bx           ;面号出栈
        mov dh, bl
        pop bx           ;功能号出栈
        mov ah, bl

sret:   pop bx
        iret

s1:     mov ah, 2         ;int 13h的功能号(ah=2表示读扇区)
        push ax          ;功能号，入栈
        jmp s3

s2:     mov ah, 3         ;int 13h的功能号(ah=3表示写扇区)
        push ax          ;功能号，入栈
        jmp s3

send:   nop
code ends
end start

```

---

## 课程设计2

阅读下面的材料：

开机后，CPU自动进入到FFFF:0单元处执行，此处有一条跳转指令。CPU执行该指令后，转去执行BIOS中的硬件系统检测和初始化程序。

初始化程序将建立BIOS所支持的中断向量，即将BIOS提供的中断例程的入口地址登记在中断向量表中。

硬件系统检测和初始化完成后，调用int 19h进行操作系统的引导。

如果设为从软盘启动操作系统,则int 19h将主要完成以下工作。

(1)控制0号软驱,读取软盘0道0面1扇区的内容到0:7c00;

(2)将CS:IP指向0:7c00。

软盘的0道0面1扇区中装有操作系统引导程序。int 19h将其装到0:7c00处后,设置CPU从0:7c00开始执行此处的引导程序,操作系统被激活,控制计算机。

如果在0号软驱中没有软盘,或发生软盘I/O错误,则int 19h将主要完成以下工作。

(1)读取硬盘C的0道0面1扇区的内容到0:7c00;

(2)将CS:IP指向0:7c00。

这次课程设计的任务是编写一个可以自行启动的计算机,不需要在现有操作系统环境中运行的程序。

该程序的功能如下:

(1)列出功能选项,让用户通过键盘进行选择,界面如下

- 1) reset pc ;重新启动计算机
- 2) start system ;引导现有的操作系统
- 3) clock ;进入时钟程序
- 4) set clock ;设置时间

(2)用户输入"1"后重新启动计算机(提示:考虑ffff:0单元)

(3)用户输入"2"后引导现有的操作系统(提示:考虑硬盘C的0道0面1扇区)。

(4)用户输入"3"后,执行动态显示当前日期、时间的程序。

显示格式如下:年/月/日 时:分:秒

进入此项功能后,一直动态显示当前的时间,在屏幕上将出现时间按秒变化的效果(提示:循环读取CMOS)。

当按下F1键后,改变显示颜色;按下Esc键后,返回到主选单(提示:利用键盘中断)。

(5)用户输入"4"后可更改当前的日期、时间,更改后返回到主选单(提示:输入字符串)。

```
assume cs:code
code segment
;=====
;功能: 将代码写入0面0道1扇区
;入口参数:
;    (ah)=int 13h的功能号(2表示读扇区, 3表示写扇区)
;    (al)=写入的扇区数
;    (ch)=磁道号
;    (cl)=扇区号
;    (dh)=磁头号(面号)
;    (dl)=驱动器号    软区从0开始, 0: 软驱A, 1: 软驱B
;                    硬盘从80h开始, 80h: 硬盘C, 81h: 硬盘D
;    es:bx指向将写入磁盘的数据或指向接收从扇区读入数据的内存区
;返回参数:
;    操作成功: (ah)=0, (al)=写入的扇区数
;    操作失败: (ah)=出错代码
;=====
start:
    mov ax, floppyend-floppy
    mov dx, 0
    mov bx, 512
```

```

div bx      ;商ax为所需的扇区数
inc al      ;写入的扇区数(为余数加扇区)

push cs
pop es
mov bx,offset floppy ;es:bx指向要被写入的内存单元

mov ch,0    ;磁道号
mov cl,1    ;扇区号
mov dl,0    ;驱动器号, 软盘A
mov dh,0    ;磁头号(面号)
mov ah,3    ;int 13h的功能号(3表示写扇区)
int 13h     ;将代码写入0面0道1扇区

mov ax,4c00h
int 21h

floppy:
    jmp read
;=====
;直接定址表
;=====
table dw function1-floppy,function2-floppy
      dw function3-floppy,function4-floppy

menu  db '***main Menu***',0
      db '1) reset pc      ',0
      db '2) start system',0
      db '3) clock        ',0
      db '4) set clock    ',0
      db 'Please Choose: ',0

time  db 'YY/MM/DD hh:mm:ss',0
cmos  db 9,8,7,4,2,0
hint  db 'press F1 to change the color, press Esc to return',0

hint1 db 'Please input: YY/MM/DD hh:mm:ss',0
char  db ' / /      : : ',0

;=====
;功能: 将0面0道2扇区的内容写入0:7e00h
;入口参数:
;    (ah)=int 13h的功能号(2表示读扇区)
;    (al)=写入的扇区数
;    (ch)=磁道号
;    (cl)=扇区号
;    (dh)=磁头号(面号)

```

```

;      (dl)=驱动器号      软区从0开始, 0: 软驱A, 1: 软区B
;
;      硬盘从80h开始, 80h: 硬盘C, 81h: 硬盘D
;      es:bx指向将写入磁盘的数据
;返回参数:
;      操作成功: (ah)=0, (al)=写入的扇区数
;      操作失败: (ah)=出错代码
;=====
read:
    mov ax,floppyend-floppy
    mov dx,0
    mov bx,512
    div bx      ;商ax为所需的扇区数
    inc al

    mov bx,0
    mov es,bx
    mov bx,7e00h ;es:bx指向要读入的内存单元

    mov ch,0    ;磁道号
    mov cl,2    ;扇区号
    mov dl,0    ;驱动器号
    mov dh,0    ;磁头号(面号)
    mov ah,2    ;int 13h的功能号(2表示读扇区)
    int 13h     ;读取0面0道2扇区的内容到0:7e00h处
                ;(第二扇区从512/200h开始)

    mov ax,7c0h
    push ax     ;push cs
    mov ax,showmenu-floppy
    push ax     ;push ip
    retf       ;jmp showmenu

;*****
;显示主菜单, 调用show_str、clean子程序
;*****
showmenu:
    call clean  ;清屏
    push cs
    pop ds
    mov si,menu-floppy
    mov dh,8
    mov dl,30
    mov cx,6
showmenu0:
    push cx
    mov cl,2
    call show_str

```

```

        add si,16
        inc dh
        pop cx
        loop showmenu0

;=====
;接收键盘输入，跳转相应功能程序段
;调用BIOS用来提供读取键盘缓冲区功能的中断例程int 16h，
;将读取的扫描码送入ah，ASCII码送入al
;=====
go:      mov ax,0
        int 16h
        cmp al,'1'
        jb showmenu
        cmp al,'4'
        ja showmenu

        sub al,31h
        mov bl,al
        mov bh,0
        add bx,bx
        add bx,3      ;计算相应子程序在table中的位移
        call word ptr cs:[bx]

        jmp showmenu

;*****
;功能1：重新启动计算机
;*****
function1:
        mov ax,0ffffh
        push ax
        mov ax,0
        push ax
        retf          ;jmp ffff:0

;*****
;功能3：进入时钟程序
;*****
function3:
        push ax
        push bx
        push cx
        push dx
        push si
        push ds
        push es

```

```

        call clean    ;清屏
        mov dh,0      ;行号
        mov dl,0      ;列号
        mov cl,2
        mov si,offset hint-floppy
        call show_str ;显示提示信息
;=====
;名称: clock
;功能: 动态显示当前日期、时间
;=====
        mov cx,2      ;显示颜色
clock:  mov bx,offset cmos-floppy
        mov si,offset time-floppy
        push cx
        mov cx,6
clock0: push cx
        mov al,[bx]
        out 70h,al    ;将al送入地址端口70h
        in al,71h     ;从数据端口71h处读出单元内容
        mov ah,al
        mov cl,4
        shr al,cl     ;右移4位
        and ah,0fh    ;al分成两个表示BCD码值的数据
        add ax,3030h  ;BCD码+30h=10进制数对应的ASCII码
        mov [si],ax   ;ASCII码写入time段
        inc bx
        add si,3
        pop cx
        loop clock0
;-----
;按下F1键后, 改变显示颜色
;按下Esc键后, 返回主菜单, 其他键照常处理
;-----
        mov al,0
        in al,60h
        pop cx        ;显示颜色
        cmp al,3bh    ;和F1的扫描码3bh比较
        je colour
        cmp al,1      ;和esc的扫描码01h比较
        je clockend
        jmp show_clock
col_1:  mov cx,1       ;cl∈[1,7], 0为黑色
        jmp show_clock
colour: cmp cx,7
        je col_1
        inc cx
show_clock:

```



```

        mov dh,12      ;行号
        mov dl,30      ;列号
        mov si,offset time-floppy
        call show_str
        jmp clock      ;循环显示CMOS
clockend:
        pop es
        pop ds
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        ret

;*****
;功能4: 设置时间
;*****
function4:
        push ax
        push bx
        push cx
        push dx
        push si
        call clean

        mov dh,8      ;行号
        mov dl,30      ;列号
        mov cl,2
        mov si,offset hint1-floppy
        call show_str ;显示提示信息

        add dh,1      ;行号
        add dl,14      ;列号
        mov si,offset char-floppy ;ds:si指向字符栈空间
        call show_str ;显示输入格式
        mov di,0
        call getstrs
        call witein
        call cleanchar
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        ret

```

```

;=====
;清除char内输入数据，还原环境
;=====
cleanchar:
    push cx
cleanchar1:
    mov cx, di
    jcxz cleanchar2
    call charpop
    jmp cleanchar1
cleanchar2:
    pop cx
    ret

;=====
;ASCII=>BCD，写入CMOS
;=====
witein:
    push si
    mov cx, 6
    mov bx, offset cmos-floppy
wite: push cx
    mov al, [bx]
    out 70h, al    ;将a1送入地址端口70h
    mov ax, [si]
    sub ah, 30h    ;10进制数对应的ASCII码-30h=BCD码
    sub al, 30h
    mov cl, 4
    shl al, cl     ;左移4位
    add al, ah     ;a1为8位BCD码
    out 71h, al    ;从数据端口71h处写入单元内容
    add si, 3
    inc bx
    pop cx
    loop wite
    pop si
    ret

;=====
;子程序：接收数字输入
;参数说明：di=char栈顶(字符地址、个数记录器)
;=====
getstrs:
    push ax
    push bx
getstr: mov ax, 0
    int 16h
    cmp ah, 0eh    ;退格键的扫描码

```

```

        je backspace
        cmp ah,1ch          ;Enter键的扫描码
        je enter1
        cmp al,'0'
        jb getstr
        cmp al,'9'
        ja getstr
        cmp di,16          ;限制输入个数
        ja enter1
        call charpush      ;字符入栈
        call show_str      ;显示栈中字符
        jmp getstr
backspace:
        call charpop       ;字符出栈
        call show_str      ;显示栈中字符
        jmp getstr
enter1:  call show_str      ;显示栈中字符
        pop bx
        pop ax
        ret

;=====
;子程序：数字的入栈。
;参数说明：ds:si指向char栈空间；(al)=入栈字符；
;=====
charpush:
        mov bx,di
        mov [si][bx],al
        inc di
        cmp di,2
        je adds
        cmp di,5
        je adds
        cmp di,8
        je adds
        cmp di,11
        je adds
        cmp di,14
        je adds
        ret
adds:   inc di
        ret

;=====
;子程序：数字的出栈。
;参数说明：ds:si指向char栈空间；(al)=入栈字符；
;=====

```

```

charpop:cmp di,0
        je sret                                ;栈顶为0(无字符)，结束
        cmp di,3
        je subs
        cmp di,6
        je subs
        cmp di,9
        je subs
        cmp di,12
        je subs
        cmp di,15
        je subs
        dec di
        mov bx,di
        mov al,' '
        mov [si][bx],al
        ret
subs:    sub di,2
        mov bx,di
        mov al,' '
        mov [si][bx],al
        ret
sret:    ret

```

```

;=====
;名称: show_str子程序
;功能: 在指定的位置，用指定的颜色，显示一个用0结束的字符串。
;参数: (dh)=行号(取值范围0~24);
;       (dl)=列号(取值范围0~79);
;       (cl)=颜色;
;       ds:si指向字符串的首地址。
;返回: 无。
;=====

```

```

show_str:
        push ax
        push bx
        push cx
        push dx
        push si
        push es
        mov ax,0b800h
        mov es,ax
        mov ax,160
        mul dh
        mov bx,ax      ;bx=160*dh
        mov ax,2
        mul dl          ;ax=dl*2

```

```

        add bx,ax      ;mov bx, (160*dh+d1*2) 设置es:bx指向显存首地址
        mov al,c1      ;把颜色c1赋值al
        mov cl,0
show0:  mov ch,[si]
        jcxz show1     ;(ds:si)=0时, 转到show1执行
        mov es:[bx],ch
        mov es:[bx].1,al
        inc si         ;ds:si指向下一个字符地址
        add bx,2       ;es:bx指向下一个显存地址
        jmp show0
show1:  pop es
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        ret

;=====
;名称: clean子程序
;功能: 清屏
;=====
clean:  push bx
        push cx
        push es
        mov bx,0b800h
        mov es,bx
        mov bx,0
        mov cx,2000
clean0: mov byte ptr es:[bx], ' '
        add bx,2
        loop clean0
        pop bx
        pop cx
        pop es
        ret

;*****
;功能2: 引导现有的操作系统
;*****
;=====
;功能2实现引导现有的操作系统, 代码需要将硬盘的0面0道1扇区读入0:7c00,
;会覆盖从软盘读到0:7c00的第一个扇区, 所以功能2代码不能写在第一个扇区
;=====
function2:
        call clean
        mov ax,0

```

```

        mov es, ax
        mov bx, 7c00h

        mov al, 1      ;读取的扇区数
        mov ch, 0      ;磁道号
        mov cl, 1      ;扇区号
        mov dl, 80h    ;驱动器号
        mov dh, 0      ;磁头号(面号)
        mov ah, 2      ;int 13h的功能号(2表示读扇区)
        int 13h        ;读取0面0道1扇区的内容到0:7c00h处

        mov ax, 0
        push ax         ;push cs
        mov ax, 7c00h
        push ax         ;push ip
        retf           ;jmp 0:7c00h

floppyend:nop

code ends
end start

```

---

### 研究试验1 搭建一个精简的C语言开发环境

除了TC.EXE外，必须用到的相关文件有5个，分别是：

COS.OBJ  
 EMU.LIB  
 MATHS.LIB  
 GRAPHICS.LIB  
 CS.LIB

---

### 研究实验2 使用寄存器

(2)用debug加载url.exe,用u命令查看url.c编译后的机器码和汇编代码。

思考:main函数的代码在什么段?用debug怎样找到url.exe中main函数的代码?

main函数的代码应该在code段中

用Debug加载程序后用反汇编命令U找到main函数汇编代码。

(3)用下面的方法打印出url.exe被加载运行时,main函数在代码段中的偏移地址:

```

main()
{

```

```
    printf("%x\n",main);
}
```

“%x”指的是按照十六进制格式打印。

思考:为什么这个程序能够打印出main函数在代码段中的偏移地址?

main类似于汇编中子程序的标号,其入口偏移地址传递给printf函数被打印出来。

(4)url.c源程序

```
main()
{
    _AX=1;
    _BX=1;
    _CX=2;
    _AX=_BX+_CX;
    _AH=_BL+_CL;
    _AL=_BH+_CH;

    printf("%x\n",main);
}
```

Debug反汇编查看

C:\>debug url.exe

~u

```
0C71:0000 BAA90D      MOV     DX,0DA9
0C71:0003 2E             CS:
0C71:0004 8916F801      MOV     [01F8],DX
0C71:0008 B430          MOV     AH,30
0C71:000A CD21          INT     21
0C71:000C 8B2E0200      MOV     BP,[0002]
0C71:0010 8B1E2C00      MOV     BX,[002C]
0C71:0014 8EDA          MOV     DS,DX
0C71:0016 A39200        MOV     [0092],AX
0C71:0019 8C069000      MOV     [0090],ES
0C71:001D 891E8C00      MOV     [008C],BX
```

~u

.

.

.

~u

```
0C71:01F8 0000      ADD     [BX+SI],AL
0C71:01FA 55        PUSH    BP           ;main函数入口
0C71:01FB 8BEC      MOV     BP,SP
0C71:01FD B80100    MOV     AX,0001
0C71:0200 BB0100    MOV     BX,0001
0C71:0203 B90200    MOV     CX,0002
0C71:0206 8BC3      MOV     AX,BX
0C71:0208 03C1      ADD     AX,CX
```

0C71:020A 8AE3	MOV	AH, BL	
0C71:020C 02E1	ADD	AH, CL	
0C71:020E 8AC7	MOV	AL, BH	
0C71:0210 02C5	ADD	AL, CH	
0C71:0212 B8FA01	MOV	AX, 01FA	;main函数偏移地址赋值AX
0C71:0215 50	PUSH	AX	
0C71:0216 B89401	MOV	AX, 0194	
-u			
0C71:0219 50	PUSH	AX	
0C71:021A E8B808	CALL	0AD5	;调用显示地址子程序1
0C71:021D 59	POP	CX	
0C71:021E 59	POP	CX	
0C71:021F 5D	POP	BP	
0C71:0220 C3	RET		;main函数结束
0C71:0221 55	PUSH	BP	
0C71:0222 8BEC	MOV	BP, SP	
0C71:0224 56	PUSH	SI	
0C71:0225 8B7604	MOV	SI, [BP+04]	
0C71:0228 0BF6	OR	SI, SI	
0C71:022A 7C14	JL	0240	
0C71:022C 83FE58	CMP	SI, +58	
0C71:022F 7603	JBE	0234	
0C71:0231 BE5700	MOV	SI, 0057	
0C71:0234 89369801	MOV	[0198], SI	
0C71:0238 8A849A01	MOV	AL, [SI+019A]	
-u 0C71:0AD5			
0C71:0AD5 55	PUSH	BP	;显示地址子程序1入口
0C71:0AD6 8BEC	MOV	BP, SP	
0C71:0AD8 B84D0C	MOV	AX, 0C4D	
0C71:0ADB 50	PUSH	AX	
0C71:0ADC B81002	MOV	AX, 0210	
0C71:0ADF 50	PUSH	AX	
0C71:0AE0 FF7604	PUSH	[BP+04]	
0C71:0AE3 8D4606	LEA	AX, [BP+06]	
0C71:0AE6 50	PUSH	AX	
0C71:0AE7 E84C02	CALL	0D36	;调用子程序2
0C71:0AEA EB00	JMP	0AEC	
0C71:0AEC 5D	POP	BP	
0C71:0AED C3	RET		;子程序1结束返回
0C71:0AEE 55	PUSH	BP	
0C71:0AEF 8BEC	MOV	BP, SP	
0C71:0AF1 8B5E06	MOV	BX, [BP+06]	
0C71:0AF4 FF0F	DEC	WORD PTR [BX]	
-u 0C71:0D36			
0C71:0D36 55	PUSH	BP	;调用子程序2入口
0C71:0D37 8BEC	MOV	BP, SP	
0C71:0D39 81EC9600	SUB	SP, 0096	



```

0C71:0D3D 56          PUSH    SI
0C71:0D3E 57          PUSH    DI
0C71:0D3F C746AA0000    MOV     WORD PTR [BP-56], 0000
0C71:0D44 C646AD50    MOV     BYTE PTR [BP-53], 50
0C71:0D48 EB38          JMP     0D82
0C71:0D4A 57          PUSH    DI
0C71:0D4B B9FFFF    MOV     CX, FFFF
0C71:0D4E 32C0        XOR     AL, AL
0C71:0D50 F2          REPNZ
0C71:0D51 AE          SCASB
0C71:0D52 F7D1        NOT     CX
0C71:0D54 49          DEC     CX
0C71:0D55 5F          POP     DI
-u
0C71:0D56 C3          RET     ;子程序2结束返回
0C71:0D57 8805        MOV     [DI], AL
0C71:0D59 47          INC     DI
0C71:0D5A FE4EAD    DEC     BYTE PTR [BP-53]
0C71:0D5D 7E22        JLE     0D81
0C71:0D5F 53          PUSH    BX
0C71:0D60 51          PUSH    CX
0C71:0D61 52          PUSH    DX
0C71:0D62 06          PUSH    ES
0C71:0D63 8D46AE    LEA     AX, [BP-52]
0C71:0D66 2BF8        SUB     DI, AX
0C71:0D68 8D46AE    LEA     AX, [BP-52]
0C71:0D6B 50          PUSH    AX
0C71:0D6C 57          PUSH    DI
0C71:0D6D FF7608    PUSH    [BP+08]
0C71:0D70 FF560A    CALL    [BP+0A]
0C71:0D73 C646AD50    MOV     BYTE PTR [BP-53], 50
-

```

(5)通过main函数后面有ret指令,我们可以设想:C语言将函数实现为汇编语言中的子程序。研究下面程序的汇编代码,验证这个设想。

ur2.c源程序

```
void f(void);
```

```
main()
```

```
{
    _AX=1; _BX=1; _CX=2;
```

```
    f();
}
```

```
void f(void)
```

```
{
```

```

    _AX=_BX+_CX;
}

```

Debug反汇编查看

C:\>debug ur2.exe

-U 0C71:1FA

```

0C71:01FA 55          PUSH    BP           ;main函数入口
0C71:01FB 8BEC        MOV     BP, SP
0C71:01FD B80100      MOV     AX, 0001
0C71:0200 BB0100      MOV     BX, 0001
0C71:0203 B90200      MOV     CX, 0002
0C71:0206 E80200      CALL    020B         ;调用f函数
0C71:0209 5D          POP     BP
0C71:020A C3          RET
0C71:020B 55          PUSH    BP
0C71:020C 8BEC        MOV     BP, SP
0C71:020E 8BC3        MOV     AX, BX
0C71:0210 03C1        ADD     AX, CX
0C71:0212 5D          POP     BP
0C71:0213 C3          RET                 ;main函数结束返回
0C71:0214 C3          RET
0C71:0215 55          PUSH    BP
0C71:0216 8BEC        MOV     BP, SP
0C71:0218 EBOA        JMP     0224

```

-U 0C71:020B

```

0C71:020B 55          PUSH    BP           ;f函数入口
0C71:020C 8BEC        MOV     BP, SP
0C71:020E 8BC3        MOV     AX, BX
0C71:0210 03C1        ADD     AX, CX
0C71:0212 5D          POP     BP
0C71:0213 C3          RET                 ;f函数结束返回
0C71:0214 C3          RET
0C71:0215 55          PUSH    BP
0C71:0216 8BEC        MOV     BP, SP
0C71:0218 EBOA        JMP     0224
0C71:021A 8B1E9E01    MOV     BX, [019E]
0C71:021E D1E3        SHL     BX, 1
0C71:0220 FF97A601    CALL    [BX+01A6]
0C71:0224 A19E01      MOV     AX, [019E]
0C71:0227 FF0E9E01    DEC     WORD PTR [019E]

```

-

当main函数在调用void f(void)这个函数时，其汇编代码为call地址  
验证了C语言将函数实现为汇编语言中的子程序，C函数名相当于子程序的标号。

---

### 研究试验3 使用内存空间

#### (1) um1.c源程序

```
main()
{
    *(char *)0x2000='a';
    *(int *)0x2000=0xf;
    *(char far *)0x20001000='a';

    _AX=0x2000;
    *(char *)_AX='b';

    _BX=0x1000;
    *(char *)(_BX+_BX)='a';

    *(char far *) (0x20001000+_BX)=*(char *)_AX;
}
```

#### Debug反汇编查看

C:\>debug um1.exe

-u0c71:1fa

0C71:01FA	55	PUSH	BP	;main函数入口
0C71:01FB	8BEC	MOV	BP, SP	
0C71:01FD	C606002061	MOV	BYTE PTR [2000], 61	; 'a' 写入ds:2000
0C71:0202	C70600200F00	MOV	WORD PTR [2000], 000F	; 0fh写入ds:2000
0C71:0208	BB0020	MOV	BX, 2000	
0C71:020B	8EC3	MOV	ES, BX	
0C71:020D	BB0010	MOV	BX, 1000	
0C71:0210	26	ES:		
0C71:0211	C60761	MOV	BYTE PTR [BX], 61	; 'a' 写入2000:1000
0C71:0214	B80020	MOV	AX, 2000	
0C71:0217	8BD8	MOV	BX, AX	
0C71:0219	C60762	MOV	BYTE PTR [BX], 62	; 'b' 写入ds:2000
-u				
0C71:021C	BB0010	MOV	BX, 1000	
0C71:021F	03DB	ADD	BX, BX	
0C71:0221	C60761	MOV	BYTE PTR [BX], 61	; 'a' 写入ds:2000
0C71:0224	8BD8	MOV	BX, AX	
0C71:0226	8A07	MOV	AL, [BX]	
0C71:0228	33C9	XOR	CX, CX	
0C71:022A	81C30010	ADD	BX, 1000	
0C71:022E	81D10020	ADC	CX, 2000	
0C71:0232	8EC1	MOV	ES, CX	
0C71:0234	26	ES:		
0C71:0235	8807	MOV	[BX], AL	; 'a' 写入2000:3000
0C71:0237	5D	POP	BP	
0C71:0238	C3	RET		;main函数结束返回

0C71:0239 C3 RET

#### 单步跟踪执行情况

-g01fa

AX=0000 BX=059A CX=0015 DX=F21F SP=FFE8 BP=FFF2 SI=0366 DI=0555

DS=0CCA ES=0CCA SS=0CCA CS=0C71 IP=01FA NV UP EI PL ZR NA PE NC

0C71:01FA 55 PUSH BP

-t

AX=0000 BX=059A CX=0015 DX=F21F SP=FFE6 BP=FFF2 SI=0366 DI=0555

DS=0CCA ES=0CCA SS=0CCA CS=0C71 IP=01FB NV UP EI PL ZR NA PE NC

0C71:01FB 8BEC MOV BP, SP

-t

AX=0000 BX=059A CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555

DS=0CCA ES=0CCA SS=0CCA CS=0C71 IP=01FD NV UP EI PL ZR NA PE NC

0C71:01FD C606002061 MOV BYTE PTR [2000], 61 DS:2000=69

-d 0cca:2000 200f

OCCA:2000 69 63 65 20 79 6F 75 20-77 61 6E 74 20 74 6F 20 ice you want to

-t

AX=0000 BX=059A CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555

DS=0CCA ES=0CCA SS=0CCA CS=0C71 IP=0202 NV UP EI PL ZR NA PE NC

0C71:0202 C70600200F00 MOV WORD PTR [2000], 000F DS:2000=6361

-d 0cca:2000 200f

OCCA:2000 61 63 65 20 79 6F 75 20-77 61 6E 74 20 74 6F 20 ace you want to

-t

AX=0000 BX=059A CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555

DS=0CCA ES=0CCA SS=0CCA CS=0C71 IP=0208 NV UP EI PL ZR NA PE NC

0C71:0208 BB0020 MOV BX, 2000

-d 0cca:2000 200f

OCCA:2000 0F 00 65 20 79 6F 75 20-77 61 6E 74 20 74 6F 20 ..e you want to

-t

AX=0000 BX=2000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555

DS=0CCA ES=0CCA SS=0CCA CS=0C71 IP=020B NV UP EI PL ZR NA PE NC

0C71:020B 8EC3 MOV ES, BX

-t

AX=0000 BX=2000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555

DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=020D NV UP EI PL ZR NA PE NC

0C71:020D BB0010 MOV BX, 1000

-t

AX=0000 BX=1000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555

DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0210 NV UP EI PL ZR NA PE NC

0C71:0210 26 ES:

0C71:0211 C60761 MOV BYTE PTR [BX], 61 ES:1000=00

-d 2000:1000 100f

2000:1000 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....

-t

AX=0000 BX=1000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555

DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0214 NV UP EI PL ZR NA PE NC

```

0C71:0214 B80020      MOV      AX,2000
-d 2000:1000 100f
2000:1000 61 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  a.....
-t
AX=2000 BX=1000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0217 NV UP EI PL ZR NA PE NC
0C71:0217 8BD8      MOV      BX,AX
-t
AX=2000 BX=2000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0219 NV UP EI PL ZR NA PE NC
0C71:0219 C60762      MOV      BYTE PTR [BX],62          DS:2000=0F
-t
AX=2000 BX=2000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=021C NV UP EI PL ZR NA PE NC
0C71:021C BB0010      MOV      BX,1000
-d 0cca:2000 200f
0CCA:2000 62 00 65 20 79 6F 75 20-77 61 6E 74 20 74 6F 20  b.e you want to
-t
AX=2000 BX=1000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=021F NV UP EI PL ZR NA PE NC
0C71:021F 03DB      ADD      BX,BX
-t
AX=2000 BX=2000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0221 NV UP EI PL NZ NA PE NC
0C71:0221 C60761      MOV      BYTE PTR [BX],61          DS:2000=62
-t
AX=2000 BX=2000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0224 NV UP EI PL NZ NA PE NC
0C71:0224 8BD8      MOV      BX,AX
-d 0cca:2000 200f
0CCA:2000 61 00 65 20 79 6F 75 20-77 61 6E 74 20 74 6F 20  a.e you want to
-t
AX=2000 BX=2000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0226 NV UP EI PL NZ NA PE NC
0C71:0226 8A07      MOV      AL,[BX]          DS:2000=61
-t
AX=2061 BX=2000 CX=0015 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0228 NV UP EI PL NZ NA PE NC
0C71:0228 33C9      XOR      CX,CX
-t
AX=2061 BX=2000 CX=0000 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=022A NV UP EI PL ZR NA PE NC
0C71:022A 81C30010      ADD      BX,1000
-t
AX=2061 BX=3000 CX=0000 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=022E NV UP EI PL NZ NA PE NC
0C71:022E 81D10020      ADC      CX,2000

```

```

-t
AX=2061 BX=3000 CX=2000 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0232 NV UP EI PL NZ NA PE NC
0C71:0232 8EC1 MOV ES,CX
-t
AX=2061 BX=3000 CX=2000 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0234 NV UP EI PL NZ NA PE NC
0C71:0234 26 ES:
0C71:0235 8807 MOV [BX],AL ES:3000=00
-d 2000:3000 300f
2000:3000 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-t
AX=2061 BX=3000 CX=2000 DX=F21F SP=FFE6 BP=FFE6 SI=0366 DI=0555
DS=0CCA ES=2000 SS=0CCA CS=0C71 IP=0237 NV UP EI PL NZ NA PE NC
0C71:0237 5D POP BP
-d 2000:3000 300f
2000:3000 61 00 00 00 00 00 00 00-00 00 00 00 00 00 00 a.....
-

```

(2)

char型数据为字节型操作，占1个内存单元

int型数据为字型操作，占2个内存单元

一条指令，需将颜色和字符同时写入，使用int型

main()

```

{
    *(int *)0xb80007ce=0x261;
}

```

(3)分析下面程序中所有函数的汇编代码，思考相关的问题。

```
int a1,a2,a3;
```

```
void f(void);
```

```
main()
```

```

{
    int b1,b2,b3;
    a1=0xa1;a2=0xa2;a3=0xa3;
    b1=0xb1;b2=0xb2;b3=0xb3;
    printf("%x\n",main);
}
void f(void)
{
    int c1,c2,c3;
    a1=0x0fa1;a2=0xfa2;a3=0x0fa3;
    c1=0xc1;c2=0xc2;c3=0xc3;
}

```

问题：C语言将全局变量存放在哪里？将局部变量存放在哪里？每个函数开关的“push bp mov bp, sp”有何含义？

Debug反汇编查看

```
1412:01FA 55          PUSH    BP
1412:01FB 8BEC        MOV     BP, SP
1412:01FD 83EC06      SUB     SP, +06
1412:0200 C7062604A100 MOV     WORD PTR [0426], 00A1
1412:0206 C7062804A200 MOV     WORD PTR [0428], 00A2
1412:020C C7062A04A300 MOV     WORD PTR [042A], 00A3
1412:0212 C746FAB100  MOV     WORD PTR [BP-06], 00B1
1412:0217 C746FCB200  MOV     WORD PTR [BP-04], 00B2
1412:021C C746FEB300  MOV     WORD PTR [BP-02], 00B3
1412:0221 B8FA01      MOV     AX, 01FA
1412:0224 50          PUSH    AX          ; main函数入口
1412:0225 B89401      MOV     AX, 0194
1412:0228 50          PUSH    AX
1412:0229 E8E508      CALL    0B11
1412:022C 59          POP     CX
1412:022D 59          POP     CX
1412:022E 8BE5        MOV     SP, BP
1412:0230 5D          POP     BP
1412:0231 C3          RET
```

全局变量存放在DS段中，局部变量存放在SS段中。

PUSH BP          MOV BP, SP作用为用BP保存堆栈指针，从而能够在使用局部变量时用[BP+idata]来定位。

(4) 分析下面程序的汇编代码，思考相关的问题。

```
int f(void);
int a,b,ab;
main()
{
    int c;
    c=f();
}
int f(void)
{
    ab=a+b;
    return ab;
}
```

问题：C语言将函数的返回值存放在哪里？

Debug反汇编查看

```
1412:01FA 55          PUSH    BP
1412:01FB 8BEC        MOV     BP, SP
1412:01FD 83EC02      SUB     SP, +02
1412:0200 E80700      CALL    020A
1412:0203 8946FE      MOV     [BP-02], AX
1412:0206 8BE5        MOV     SP, BP
1412:0208 5D          POP     BP
```

```

1412:0209 C3          RET
1412:020A 55          PUSH    BP
1412:020B 8BEC        MOV     BP, SP
1412:020D A1A601      MOV     AX, [01A6]
1412:0210 0306A801    ADD     AX, [01A8]
1412:0214 A3AA01      MOV     [01AA], AX
1412:0217 A1AA01      MOV     AX, [01AA]
1412:021A EB00        JMP     021C
1412:021C 5D          POP     BP
1412:021D C3          RET

```

C语言将函数的返回值存放在AX中。

(5) 下面的程序向安全的内存空间写入从“a”到“h”的8个字符，理解程序的含义，深入理解相关的知识。

```

#define Buffer ((char *)*(int far *)0x02000000)
main()
{
    Buffer=(char *)malloc(20);
    Buffer[10]=0;
    while(Buffer[10]!=8)
    {
        Buffer[Buffer[10]]='a'+Buffer[10];
        Buffer[10]++;
    }
}

```

---

标志寄存器flag

```

15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0  0  0  0 of df if tf sf zf  0 af  0 pf  0 cf

```

ZF是flag的第6位，零标志位，判断结果是否为0，结果为0，ZF=1

PF是flag的第2位，奇偶标志位，运算结果二进制数中1的个数为偶数，PF=1

SF是flag的第7位，符号标志位，有符号数 运算结果为负数，SF=1

CF是flag的第0位，进位标志位，无符号数 运算结果有进/借位，CF=1

OF是flag的第11位，溢出标志位，有符号数 运算结果溢出，OF=1

DF是flag的第10位，方向标志位，DF=0 每次操作后 si, di递增

DF=1 每次操作后 si, di递减

TF是flag的第8位，TF=1，产生单步中断，引发中断过程

IF是flag的第9位，IF=1，CPU响应中断，引发中断过程

IF=0，不响应可屏蔽中断



add、sub、mul、div、inc、or、and等运算指令影响flag  
mov、push、pop等传送指令对flag没影响

abc 带位加法指令，利用CF位上记录的进位值  
abc ax, bx 实现功能  $(ax) = (ax) + (bx) + CF$

sbb 带位减法指令，利用CF位上记录的借位值  
sbb ax, bx 实现功能  $(ax) = (ax) - (bx) - CF$

cmp 比较指令，相当于减法指令，不保存结果，只影响flag相关各位  
cmp ax, bx 如果  $(ax) = (bx)$ ，则  $zf=1$   
如果  $(ax) \neq (bx)$ ，则  $zf=0$   
如果  $(ax) < (bx)$ ，则  $cf=1$   
如果  $(ax) \geq (bx)$ ，则  $cf=0$   
如果  $(ax) > (bx)$ ，则  $cf=0$ 且 $zf=0$   
如果  $(ax) \leq (bx)$ ，则  $cf=1$ 或 $zf=1$

如果因为溢出导致了实际结果为负，那么逻辑上真正的结果必然为正  
如果因为溢出导致了实际结果为正，那么逻辑上真正的结果必然为负

cmp指令和条件转移指令配合使用

指令	含义	检测的相关标志位
je	等于则转移	$zf=1$
jne	不等于则转移	$zf=0$
ja	高于则转移	$cf=0$ 且 $zf=0$
jb	低于则转移	$cf=1$
jna	不高于则转移	$cf=1$ 或 $zf=1$
jnb	不低于则转移	$cf=0$

[...]是闭区间，包括两端点的值  
(...)是开区间，不包括两端点的值

DF标志和串传送指令

DF 方向标志位，在串处理指令中，控制每次操作后si、di的增减  
df=0 每次操作后si、di递增  
df=1 每次操作后si、di递减

movsb:

相当于mov es:[di],byte ptr ds:[si]  
如果 df=0 如果 df=1  
inc di dec di  
inc si dec si

movsw:

相当于mov es:[di],word ptr ds:[si]  
如果 df=0 如果 df=1  
add si,2 sub si,2  
add di,2 sub di,2

movsb或movsw和rep配合使用，格式如下：

rep movsb

相当于

s: movsb

loop s

rep的作用是根据cx的值，重复执行rep后面的串传送指令

cld 将标志寄存器中的DF位置0，（配合movsb,则是正向传送）

std 将标志寄存器中的DF位置1，（配合movsb,则是反向传送）

pushf 标志寄存器的值压栈

popf 标志寄存器的值出栈

标志位在debug中的表示

AX=0000	BX=0000	CX=0000	DX=0000	SP=FFEE	BP=0000	SI=0000	DI=0000								
DS=****	ES=****	SS=****	CS=****	IP=0100	NV	UP	EI	PL	NZ	NA	PO	NC			
flag					of	df		sf	zf		pf	cf			
值为1的标记					OV	DN		NG	ZR		PE	CY			
值为0的标记					NV	UP		PL	NZ		PO	NC			

确定补码:

Xh为8位,  $X \in (80h, ffh)$ , 则Xh是 $(Xh-256)$ 的补码

Xh为16位,  $X \in (8000h, fffffh)$ , 则Xh是 $(Xh-65535)$ 的补码

---

## 内中断

4种情况将产生相应的中断信息

- (1) 除法错误
- (2) 单步执行
- (3) 执行into指令
- (4) 执行int指令

从内存0:0到0:3fff的1024个单元中存放着中断向量表

一个表项存放一个中断向量，也就是一个中断处理程序的入口地址，这个入口地址包括段地址和偏移地址，一个表项占两个字，高地址存放段地址，低地址存放偏移地址

中断过程:

- (1) 取得中断类型码N
- (2) pushf
- (3) TF=0, IF=0
- (4) push CS
- (5) push IP
- (6)  $(IP) = (N*4)$ ,  $(CS) = (N*4+2)$

iret指令, 相当于:

pop IP

ret指令, 相当于:

pop IP

retf指令, 相当于:

pop IP

```
pop CS
popf
pop CS
```

iret与int指令配合使用，call和ret指令配合使用。

编写N号中断处理程序步骤：

- (1)编写N号中断处理程序：do0
- (2)将do0送入内存0:200处
- (3)将do0的入口地址0:200存储在中断向量表N号表项中

```
assume cs:code
code segment
start:  mov ax,cs
        mov ds,ax
        mov si,offset do0           ;设置ds:si指向源地址
        mov ax,0
        mov es,ax
        mov di,200h                 ;设置es:di指向目标地址
        mov cx,offset do0end-offset do0 ;设置cx为传输长度
        cld                         ;设置传输方向为正
        rep movsb
        mov ax,0
        mov es,ax
        mov word ptr es:[N*4],200h
        mov word ptr es:[N*4+2],0   ;设置中断向量表
        mov ax,4c00h
        int 21h
do0:    .....
        mov ax,4c00h
        int 21h
do0end: nop
code ends
end start
```

TF是flag的第8位，TF=1，产生单步中断，引发中断过程  
IF是flag的第9位

CPU执行完设置SS指令后，不响应中断

---

端口

CPU可以直接读写三个地方的数据：

- (1) CPU内部的寄存器
- (2) 内存单元
- (3) 端口

端口 $N \in (0, 255)$

从端口 $N$ 读入一字节: `in al, N`

从端口 $N$ 读入一字: `in ax, N`

往端口 $N$ 写入一字节: `out N, al`

往端口 $N$ 写入一字: `out N, ax`

端口 $N \in (256, 65535)$

从端口 $N$ 读入一字节: `mov dx, N`

`in al, dx`

从端口 $N$ 读入一字: `mov dx, N`

`in ax, dx`

往端口 $N$ 写入一字节: `mov dx, N`

`out N, ax`

往端口 $N$ 写入一字: `mov dx, N`

`out N, ax`

CMOS RAM芯片, 简称CMOS, 有两个端口:

70h为地址端口, 存放要访问的CMOS单元的地址

71h为数据端口, 存放从选定的CMOS单元中读取的数据, 或要写入的数据

CPU对CMOS RAM的读写分两步进行:

1、将 $N$ 送入端口70h

2、从端口71h处读出 $N$ 号单元内容

`shl`

逻辑左移指令, 功能:

(1) 将一个寄存器或内存单元中的数据向左移位

(2) 将最后移出的一位写入CF

(3) 最低位用0补充

将 $X$ 逻辑左移一位, 相当于执行 $X = X * 2$

`shr`

逻辑右移指令, 功能:

(1) 将一个寄存器或内存单元中的数据向右移位

(2) 将最后移出的一位写入CF

(3) 最高位用0补充

将 $X$ 逻辑右移一位, 相当于执行 $X = X / 2$

移动位数 $>1$ 时, 移动位数放在 $c1$ 中

`mov al, 01010001b`

`mov cl, 3`

`shl al, cl`

`shr al, cl`

左移1位,  $N = N * 2$

右移1位,  $N = N / 2$

左移2位,  $N = N * 4$

右移2位,  $N = N / 4$

左移3位,  $N = N * 8$

右移3位,  $N = N / 8$

左移4位, $N=N*16$	右移4位, $N=N/16$
左移5位, $N=N*32$	右移5位, $N=N/32$
左移6位, $N=N*64$	右移6位, $N=N/64$

BCD码——4位2进制表示10进制数码, 一个字节表示两个BCD码, 两个BCD码表示两位的10进制数  
BCD码+30h=10进制数对应的ASCII码

CMOS RAM存放着当前时间: 每个信息长度1字节, 以BCD码方式存放  
时间存放单元:

秒	分	时	日	月	年
0	2	4	7	8	9

1字节al分成两个表示BCD码值的数据, 相当于al=12345678转成ah=5678, al=1234

```
mov ah, al
mov cl, 4
shr ah, cl      ;右移4位
and al, 00001111b
```

---

## 外中断

CPU通过总线和端口来与外部设备进行联系, 外部可屏蔽中断的中断类型码是通过数据总线送入CPU的。

IF=1, CPU响应中断, 引发中断过程

IF=0, 不响应可屏蔽中断

sti指令, 设置IF=1

cli指令, 设置IF=0

中断过程:

- (1) 取得中断类型码N
- (2) pushf
- (3) TF=0, IF=0
- (4) push CS
- (5) push IP
- (6)  $(IP)=(N*4)$ ,  $(CS)=(N*4+2)$

不可屏蔽中断的中断过程:

(不可屏蔽中断的中断类型码固定为2, 不用获取)

- (1) pushf
- (2) TF=0, IF=0
- (3) push CS
- (4) push IP
- (5)  $(IP)=(8)$ ,  $(CS)=(0AH)$

几乎所有由外设引发的外中断，都是可屏蔽中断

键盘按下一个键产生的扫描码为通码

键盘松开一个键产生的扫描码为断码

扫描码长度为一字节，通码第7位为0，断码第7位为1

断码=通码+80h

p274页——键盘上部分键的扫描码

键盘输入到达60H端口时，引发9号中断

9号中断是可屏蔽中断，由BIOS提供的，用来进行基本的键盘输入处理

执行int 9中断：

(1) 读出60H端口中的扫描码

(2) 如果是字符扫描码，将扫描码和对应的字符码(ASCII码)送入内存BIOS键盘缓冲区

如果是控制键，则将其转为状态字节写入存储状态字的单元

(3) 对键盘系统进行相关的控制

在BIOS键盘缓冲区中可存储15个键盘输入，一个键盘输入用一个字单元存放，高位扫描码，低位字符码

0040:17单元存放键盘状态字节，记录了控制键和切换键的状态

p275页——键盘状态字节各位记录信息

键盘输入的处理过程：

1、键盘产生扫描码

2、扫描码送入60H端口

3、引发9号中断

4、CPU执行int 9中断例程处理键盘输入

int指令执行中断过程：

(1) 取得中断类型码N

(2) 标志寄存器入栈

(3) TF=0, IF=0

(4) CS、IP入栈

(5) (IP)=(N\*4), (CS)=(N\*4+2)

模拟

(1) 不需要定位入口地址

(2) pushf

(3) pushf

pop ax

and ah, 11111100b ;IF为flag的第9位

;TF为flag的第8位

push ax

popf

(4) call dword ptr ds:[0]

因为在进入中断例程后，IF和TF都已置0，没有必要再进行设置了，对于模拟int指令可以精简为：

pushf

call dword ptr ds:[0]

;标志寄存器入栈

;CS、IP入栈; (IP)=ds:[0], (CS)=ds:[2]

CPU对外设输入的通常处理方法:

- (1) 外设的输入端口
- (2) 向CPU发出外中断(可屏蔽中断)信息
- (3) CPU检测到可屏蔽中断信息, 如果IF=1, cpu在执行完当前指令后响应中断, 执行相应的中断例程
- (4) 可在中断例程中实现对外设输入的处理

---

## 直接定址表

```
assume cs:code
code segment
    a dw 1, 2, 3, 4, 5, 6, 7, 8
    b dd 0
start: .
    .
    .
    mov ax, 4c00h
    int 21h
code ends
end start
```

a、b后面没有“:”, 它们是同时描写内存单元地址和单元长度的标号  
在后面加“:”的地址标号, 只能在代码段中使用, 不能在其他段中使用

数据标号和地址标号唯一的区别就是, 数据标号既表示内存单元的地址, 还表示内存单元的长度, 而地址标号只表示内存单元的地址

在代码段中直接使用数据标号访问数据, 则需要用伪指令assume将标号所在的段和一个段寄存器联系起来。否则编译器在编译的时候, 无法确定标号的段地址在哪个寄存器中。这种联系是编译器需要的, 但绝对不是说, 因为编译器的工作需要, 用assume指令将段寄存器和某个段相联系, 段寄存器中就会真的存放该段的地址, 在程序中还要使用指令对段寄存器进行设置。

seg操作符, 功能为取得某一段的段地址

通过依据数据, 直接计算出所要找的元素的位置的表为直接定址表  
直接定址表包含了数据长度信息, 可以方便的编写一些查表类的程序, 直接定址表的数据可以定义在代码段

---

## 使用BIOS进行键盘输入和磁盘读写

int 9h是把键盘的扫描码读入并将其转化成ASC II 码或状态信息, 存储在内存的指定位置的中断

## 例程

int 9h中断例程对键盘输入的处理

(一) 当有字符键按下时:

- 1、从60h端口读出该键通码
- 2、检测状态字节 (shift、ctrl等状态键是否按下)
- 3、没有状态键按下, 将该键扫描码和对应ASCII码写入键盘缓冲区  
(缓冲区字单元中, 高位存储扫描码, 低位存储ASCII码)

(二) 当有状态键按下时:

- 1、接收该键通码
- 2、设置40:17处状态字节第1位为1

int 16h是BIOS用来提供读取键盘缓冲区功能的中断例程

int 16h中断例程的0号功能: 读取键盘缓冲区

- 1、检测键盘缓冲区是否有数据
- 2、没有则继续第一步
- 3、读取缓冲区第一个字单元中的键盘输入
- 4、将读取的扫描码送入ah, ASCII码送入al
- 5、将已读取的键盘输入从缓冲区中删除

BIOS提供的访问磁盘的中断例程为int 13h。

3.5寸软盘分上下两面, 每面80个磁道, 每个磁道18个扇区, 每个扇区大小为512个字节

3.5寸软盘=2面\*80磁道\*18扇区\*512字节=1440KB≈1.44MB

面号和磁道号从0开始, 而扇区号从1开始

;功能: 读取0面0道1扇区的内容到0:200

;入口参数:

```
;      (ah)=int 13h的功能号 (2表示读扇区)
;      (al)=读取的扇区数
;      (ch)=磁道号
;      (cl)=扇区号
;      (dh)=磁头号 (对于软盘即面号, 一个面用一个磁头来读写)
;      (dl)=驱动器号    软区从0开始, 0: 软驱A, 1: 软区B
;                        硬盘从80h开始, 80h: 硬盘C, 81h: 硬盘D
;      es:bx指向接收从扇区读入数据的内存区
```

;返回参数:

```
;      操作成功: (ah)=0
;      操作失败: (ah)=出错代码
      mov ax, 0
      mov es, ax
      mov bx, 200h    ;es:bx指向0:200

      mov al, 1        ;读取的扇区数
      mov ch, 0        ;磁道号
      mov cl, 1        ;扇区号
```



```

        mov dl,0      ;驱动器号
        mov dh,0      ;磁头号(面号)
        mov ah,2      ;int 13h的功能号(2表示读扇区)
        int 13h

;功能: 将0:200的内容写入0面0道1扇区
;入口参数:
;      (ah)=int 13h的功能号(3表示写扇区)
;      (al)=写入的扇区数
;      (ch)=磁道号
;      (cl)=扇区号
;      (dh)=磁头号(面号)
;      (dl)=驱动器号    软区从0开始, 0: 软驱A, 1: 软区B
;                      硬盘从80h开始, 80h: 硬盘C, 81h: 硬盘D
;      es:bx指向将写入磁盘的数据
;返回参数:
;      操作成功: (ah)=0, (al)=写入的扇区数
;      操作失败: (ah)=出错代码
        mov ax,0
        mov es,ax
        mov bx,200h   ;es:bx指向0:200

        mov al,1      ;写入的扇区数
        mov ch,0      ;磁道号
        mov cl,1      ;扇区号
        mov dl,0      ;驱动器号
        mov dh,0      ;磁头号(面号)
        mov ah,3      ;int 13h的功能号(3表示写扇区)
        int 13h

```

;功能: 将当前屏幕内容保存在磁盘上  
;1屏的内容占4000个字节, 需要8个扇区, 用0面0道的1-8扇区存储显示中的内容

```

assume cs:code
code segment
start: mov ax,0b800h
        mov es,ax
        mov bx,0      ;es:bx指向0b800:0

        mov al,8      ;写入的扇区数
        mov ch,0      ;磁道号
        mov cl,1      ;扇区号
        mov dl,0      ;驱动器号
        mov dh,0      ;磁头号(面号)
        mov ah,3      ;int 13h的功能号(3表示写扇区)
        int 13h

```

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

---

## 《汇编语言(第2版)》勘误

《汇编语言（第2版）》 第6页，第11行。

原文为：

机器码：10100000 00000011 00000000

修改为：

机器码：10100001 00000011 00000000

《汇编语言（第2版）》第330页第11行。

“用补码的特性，-20的绝对值是20，00010100b，将其取反加1后为：11101100b。可知-20H的补码为：11101100b。”中的“-20H”应改为“-20”。

《汇编语言（第2版）》目录（第VII页）

“第12章 内中断” 中的子目录中

原文：12.5 中断处理程序和ire指令

应为：12.5 中断处理程序和iret指令

《汇编语言（第2版）》第140页，第10行有印刷错误。

原文为：“db 'fork'”相当于“db 66H、6FH、52H、4BH”

修改为：“db 'fork'”相当于“db 66H, 6FH, 52H,4BH”

《汇编语言（第2版）》第194页。

检测点10.4中的“1000: 2 ff d0 call ax”一行应修改为“1000: 3 ff d0 call ax”。

在masm5.0的实验环境下，《汇编语言（第2版）》中第180页中图9.3中的内容

“jmp s0”应修改为“jmp short s0”。

《汇编语言(第2版)》 第22页倒数第2行。

“比直”应修改为“笔直”。

---

摘录的汇编网在线测试题目（部分）

\*\*\*\*\*

第1章 基础知识

(1) 一个CPU的寻址能力为8KB, 那么它的地址总线的宽度为\_\_\_\_\_。

- 1、 8
- 2、 10
- 3、 12
- 4、 13

(2) 总线从逻辑上分为3类, 下列选项中不在其中的是: \_\_\_\_\_。

- 1、 数据总线
- 2、 并行总线
- 3、 地址总线
- 4、 控制总线

(3) 1个CPU的寻址能力为32KB, 那么它的地址总线宽度为\_\_\_\_\_。

- 1、 13
- 2、 15
- 3、 18
- 4、 32k

(4) 1个CPU读取1024字节的数据至少读取了512次, 数据总线的宽度\_\_\_\_\_。

- 1、 8
- 2、 10
- 3、 16
- 4、 32

(5) 1个CPU访问的最大内存地址是1023, 地址总线的宽度\_\_\_\_\_。

- 1、 8
- 2、 10
- 3、 13
- 4、 14

\*\*\*\*\*

第2章 寄存器 (CPU工作原理)

(1) 下列地址信息与0020H:03EFH确定的内存地址不同的是\_\_\_\_\_。

- 1、 5EFH
- 2、 203H:00EFH
- 3、 005EH:000FH
- 4、 0002H:05CFH

(2) 指令执行后AX中的数据是: \_\_\_\_\_。

```
mov ax, 936aH
mov bx, 79b8H
add al, bl
```

- 1、 1c22H

- 2、 9322H
- 3、 9422H
- 4、 1d22H

(3) CPU从1000:0处开始执行指令当执行完1000:10处的指令后CPU几次修改IP\_\_\_\_\_。

```
1000: 0 mov ax,8
1000:3 jmp ax
1000:5 mov ax,0
1000:8 mov bx,ax
1000:10 jmp bx
```

- 1、 4
- 2、 5
- 3、 6
- 4、 7

(4) 在DEBUG中，\_\_\_\_\_选项中的命令可以修改内存单元的内容

- 1、 a
- 2、 d
- 3、 t
- 4、 u

(5) 下列说法中正确的是：\_\_\_\_\_。

- 1、 一条指令被执行后，IP的值进行改变。
- 2、 当CPU执行完当前指令返回debug后CPU就闲下来不再进行工作。
- 3、 e命令可将所有内存单元中的数据进行改变。
- 4、 CPU将CS:IP所指向的内存单元中的数据当作指令来执行。

(6) 下列关于8086CPU的工作原理的描述错误的是\_\_\_\_\_。

- 1、 汇编程序员可以通过对各种寄存器中内容的修改实现对CPU的控制。
- 2、 CPU在访问内存时，采用“段地址\*16+偏移地址”的形式给出要访问的内存单元的物理地址。
- 3、 任意时刻，CS:IP指向的内容即是此刻CPU正在执行的指令。
- 4、 传送指令能够更改所有通用寄存器的内容。

\*\*\*\*\*

### 第3章 寄存器(内存访问)

(1) 下列说法正确的是：\_\_\_\_\_。

- 1、 数据段和代码段的段地址不能相同。
- 2、 指令mov ax,bx执行完后bx中的值为零。
- 3、 一个栈段的大小可以设为任意值。
- 4、 当SP=0时，再次压栈将发生栈顶超界，但压栈操作有效。

(2) 能够将ax中的内容送到内存0000:0200H处的指令序列是\_\_\_\_\_。

- 1、 mov ds,0 ;  
mov bx,200h  
mov [bx],ax
- 2、 mov ax,200h

```

mov ds, ax
mov bx, 0
mov [bx], ax
3、 mov ax, 20h
mov ds, ax
mov bx, 0
mov [bx], ax
4、 mov bx, 20h
mov ds, bx
mov bx, 0
mov [bx], ax

```

(3) 在8086CPU系统中一个栈段的容量最大为\_\_\_\_\_。

- 1、 1KB
- 2、 512KB
- 3、 64KB
- 4、 32GB

(4) 能够只将al中的内容压入栈的指令序列是\_\_\_\_\_。

- 1、 push al
- 2、 pop ax
- 3、 mov ah, 0  
push ax
- 4、 mov ax, 0  
push ax

(5) 若将以1000H为段地址的整个段空间当作栈使用，那么寄存器SP的初始值最合理的设置是\_\_\_\_\_。

- 1、 0000H
- 2、 0001H
- 3、 FFFFH
- 4、 FFFEh

\*\*\*\*\*

#### 第4章 第一个程序

(1) 关于伪指令相关描述错误的是\_\_\_\_\_。

- 1、 伪指令没有对应的机器码，只用来指导汇编过程的。
- 2、 伪指令由编译器处理，在程序中可有可无。
- 3、 编译器要通过执行伪指令才能对源程序进行相应的处理操作，完成编译工作。
- 4、 伪指令是汇编语言源程序不可缺少的组成部分。

(2) 下列程序中，出现逻辑错误的是：\_\_\_\_\_。

- 1、 assume cs:code  
code segment  
mov ax, 2  
add ax, ax  
mov ax, 4c00h

```

    int 21h
    code
end
2、 assume cs:code
    code segment
    mov ax,2
    add ax,ax
    code ends
end
3、 aume cs:code
    code segment
    mov ax,2
    add ax,ax
    mov ax,4c00h
    int 21h
    code ends
end
4、 assume cs:code
    code segment
    mov ax,2
    add ax,ax
    mov ax,4c00h
    int 21h
    code ends

```

\*\*\*\*\*

#### 第5章 [bx]和loop指令

(1)在Intel8086环境下，对指令mov ax,[bx]描述错误的是\_\_\_\_\_。

- 1、 指令中，源操作数的段地址默认为ds中的内容
- 2、 指令中，源操作数的偏移地址为bx中的内容
- 3、 指令执行后，(a1)=((ds)\*16+(bx))，(ah)=((ds)\*16+(bx)+1)
- 4、 指令执行后，((ds)\*16+(bx))=(a1)，((ds)\*16+(bx)+1)=(ah)

```

(2)  mov cx,5
      s: mov ax,[bx]
        add bx,2
      loop s

```

上面指令序列，当指令第一次执行到add bx,2时cx的值\_\_\_\_\_。

- 1、 5
- 2、 4
- 3、 3
- 4、 2

(3)已知21000h处字单元的内容为 BE00H，对于如下程序：

```

    mov ax,2000h
    mov ds,ax
    mov bx,1000h

```

```

mov ax, [bx]
inc bx
inc bx
mov [bx], ax
inc bx
inc bx
mov [bx], ax
inc bx
mov [bx], al
inc bx
mov [bx], al

```

程序执行后，内存中字单元2000:1005中的内容为\_\_\_\_\_。

- 1、 00
- 2、 BE
- 3、 00BE
- 4、 0000

(4) 要计算123与456的乘积，应填在空白处的指令序列是\_\_\_\_\_。

```

assume cs:code
code segment

```

```

_____  

mov ax, 4c00h  

int 21h  

code ends  

end

```

- 1、  

```

mov ax, 1
mov cx, 123
s: add ax, 123
loop s

```
- 2、  

```

mov ax, 0
mov cx, 456
s: add ax, 456
loop s

```
- 3、  

```

mov ax, 1
mov cx, 456
s: add ax, 123
loop s

```
- 4、  

```

mov ax, 0
mov cx, 456
s: add ax, 123
loop s

```

(5) 对于如下程序

```

assume cs:code
code segment
start: mov ax, code
mov ds, ax

```

```

mov ax, 0020h
mov es, ax
mov bx, 0
s: mov al, [bx]
mov es: [bx], al
inc bx
loop s
mov ax, 4c00h
int 21h
code ends
end

```

下列说法正确的是\_\_\_\_\_。

- 1、 指令mov ax, code改为mov ax, start对程序要实现的功能没有任何影响。
- 2、 程序不能通过编译，因为在loop指令之前，没有对寄存器cx进行设定。
- 3、 程序实现的功能是将程序的所有指令复制到内存中以0:200h为起始地址的一段内存空间中。
- 4、 程序实现的功能是用内存中以0:200h为起始地址的一段内存空间中的数据将程序的所有指令覆盖。

\*\*\*\*\*

#### 第6章 包含多个段的程序

(1)在Intel8086环境下，下列说法合理的是\_\_\_\_\_。

- 1、 汇编语言程序载入内存后处于64K空间以外的数据和指令将无法使用和执行。
- 2、 一个数据段命名为data，此标号代表这个数据段在内存中的起始地址。
- 3、 如果载入的程序没有返回语句，那么当程序代码执行完毕，将继续读取后续内存空间存储的指令到CPU内部作为指令执行下去，直到遇到返回指令为止。
- 4、 用DW和DD定义进行的数据，只能够以字为单位访问。

(2)在某程序中，定义了262B的数据段data，那么程序载入内存，该段实际占用的内存空间是\_\_B。

- 1、 256
- 2、 262
- 3、 272
- 4、 512

(3)

```

assume cs:code
code segment
start: mov ax, 3
      jmp s1
s: mov bx, 0
   mov ax, bx
s1: mov cx, 3
s2: add ax, 2
   loop s2
s3: mov ax, 4c00h
   int 21h
code ends

```



end s

上面代码执行s3处的指令之前，ax的值是\_\_\_\_\_。

- 1、 9
- 2、 6
- 3、 5
- 4、 2

(4) 如下程序：

```
assume cs:codesg
codesg segment
dw 0123h, 0456h, 0789h, 0abch, 0defh, 0fedh, 0cbah, 0987h
start: mov ax, 0
mov ds, ax
mov bx, 0
mov cx, 8
s: _____
add bx, 2
loop s
mov ax, 4c00h
int 21h
codesg ends
end start
```

要实现一次用内存0:0--0:15单元中的数据改写程序中定义的数据，添加到空白处的指令序列不能是\_\_\_\_\_。

- 1、 mov cs:[bx], ds:[bx]
- 2、 mov dx, [bx]  
mov cs:[bx], dx
- 3、 push [bx]  
pop cs:[bx]
- 4、 mov ax, [bx]  
mov cs:[bx], ax

(5) 某程序有数据段、栈段和代码段三部分，如果加载后代码段的段地址为X，那么下列说法正确的是\_\_\_\_\_。

- 1、 可以断定数据段的段地址是X-2。
- 2、 可以断定栈段的段地址是X-1。
- 3、 可以断定程序PSP区的段地址是X-10。
- 4、 确定数据段和栈段的段地址与X的关系，要视其大小和在其在源程序中定义的位置关系。

\*\*\*\*\*

## 第7章 更灵活的定位内存地址的方法

(1) 生成EXE之后用Debug加载后，查看寄存器内容如下：

ds=0b2d es=0b2d ss=0b3d cs=0b3e ip=0000

程序的起始地址的段地址是\_\_\_\_\_。

- 1、 0b3e
- 2、 0b2d
- 3、 0b3d

4、0

(2) 下列指令不能执行的是\_\_\_\_\_。

- 1、           mov ax, 10h[bx]
- 2、           mov ax, 10h[di]
- 3、           mov ax, [di+si]
- 4、           mov ax, 10h[bx][si]

(3) 对如下程序要实现将datasg段中的字符串"welcome to masm!"复制到它后续的数据区中，

```
assume cs:codesg,ds:datasg
datasg segment
db 'welcome to masm!'
db '.....'
datasg ends
codesg segment
start: mov ax,datasg
mov ds,ax
mov si,0
```

\_\_\_\_\_

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

在空白区域添加的指令序列合理的是\_\_\_\_\_。

- 1、mov di,10h  
   mov cx,10h  
   s: mov ax,[si]  
   mov [di],ax  
   add si,2  
   add di,2  
   loop s
- 2、mov di,10h  
   mov cx,8  
   s: mov ax,[si]  
   mov [di],ax  
   inc si  
   inc di  
   loop s
- 3、mov ss,ax  
   mov sp,32  
   mov cx,8  
   s: mov ax,[si]  
   push ax  
   add si,2  
   loop s
- 4、mov cx,8

```
s: mov ax, [si]
mov [si+10h], ax
add si, 2
loop s
```

\*\*\*\*\*

## 第8章 数据处理的两个基本问题

(1) 下列指令序列不能够实现把内存地址0:202h中的字节数据送入al功能的是\_\_\_\_\_。

- 1、  
mov ax, 0  
mov ds, ax  
mov bx, 202h  
mov al, [bx]
- 2、  
mov ax, 0  
mov ds, ax  
mov bx, 200h  
mov al, [bx+2]
- 3、  
mov ax, 0  
mov ds, ax  
mov bp, 202h  
mov al, [bp]
- 4、  
mov ax, 0  
mov ds, ax  
mov bp, 200h  
mov al, ds:[bp+2]

(2) 下列指令不合理的是\_\_\_\_\_。

- 1、  
mov ds:[0ffh], al
- 2、  
mov ds:[0ffh], ax
- 3、  
mov ds:[0ffh], 0ffh
- 4、  
push ds:[0ffh]

(3)   
mov dx, 0  
mov ax, 1001  
mov bx, 100  
div bl  
以上四条指令执行完后，ah值是\_\_\_\_\_。

- 1、  
1
- 2、  
10
- 3、  
1001
- 4、  
0

(4) 对如下程序：

```
assume cs:codesg, ds:datasg
datasg segment
dd 123456h
dw 789h, 0h
datasg ends
```

```

codesg segment
start: mov ax,datasg
      mov ds,ax
      mov ax,ds:[0]
      mov dx,ds:[2]
      div word ptr ds:[4]
      mov ds:[6],ax
      mov ax,4c00h
      int 21h
codesg ends
end start

```

下列说法正确的是\_\_\_\_\_。

- 1、 该程序实现的功能是计算123456h与78900h相除。
- 2、 该程序由于数据定义非法，无法通过编译。
- 3、 指令div word ptr ds:[4]可改为div near ptr ds:[4],不影响程序功能。
- 4、 指令div word ptr ds:[4]采用了直接寻址方式。

\*\*\*\*\*

## 第9章 转移指令的原理

(1)对于如下程序：

```

mov ax,2
mov cx,3
s: add ax,2
  sl:loop s
mov di,offset s1
mov si,offset s3
mov ax,cs:[di]
mov cs:[si],ax
mov ax,1
mov cx,3
s2: add ax,2
s3: nop
nop

```

所有指令执行完后ax的值为\_\_\_\_\_。

- 1、 15
- 2、 7
- 3、 11
- 4、 3

(2)对于如下程序：

```

assume cs:code
data segment
  ?
data ends
code segment
start:mov ax,data
      mov ds,ax

```

```

mov bx, 0
jmp dword ptr [bx+2]
code ends
end start

```

若在指令 `jmp dword ptr [bx+2]` 执行后, 要使程序再次从第一条指令开始执行, 下列对 `data` 段中的数据的定义更合理的是\_\_\_\_\_。

- 1、 `dd 0, 0, 0`
- 2、 `dw 0, 0, 0`
- 3、 `dw 0, 0, seg code`
- 4、 `dw 0, 0, offset code`

(3) 对于指令 `jmp dword ptr [bx+0dh]` 的说法错误的是\_\_\_\_\_。

- 1、 该指令能够实现段间转移。
- 2、 该指令转移的目的地址是在指令明确给出的。
- 3、 该指令转移的目的地址被存放在内存空间中。
- 4、 该指令中运用的寻址方式是寄存器相对寻址。

(4) 有如下程序段, 填写2条指令, 使程序在运行中将 `s` 处的一条指令复制到 `s0` 处。

```

assume segment
code segment
s: mov ax, bx
   mov si, offset s
   mov di, offset s0

```

```

_____
s0: nop

```

```

nop

```

```

code ends

```

```

end s

```

- 1、 `mov ax, cs:[si]`  
`mov cs:[di], ax`
- 2、 `mov ax, cs:[di]`  
`mov cs:[si], ax`
- 3、 `mov ax, [si]`  
`mov [di], ax`
- 4、 `mov ax, ds:[di]`  
`mov ds:[si], ax`

(5) 下列能够改变 `CS: IP` 所指位置并能通过编译的指令是\_\_\_\_\_。

- 1、 `jmp short [bx]`
- 2、 `jne [bx]`
- 3、 `loop byte ptr [bx]`
- 4、 `jmp [bx]`

(6) `mov dx, 0`  
`mov ax, 1001`  
`mov bx, 100`

div bl

以上四条指令执行完后，ah值是\_\_\_\_\_。

- 1、 1
- 2、 10
- 3、 1001
- 4、 0

\*\*\*\*\*

## 第10章 CALL和RET指令

(1) 某程序中定义了如下数据：

```
data segment
db "Hello"
db 'world'
db "!"
db 'W', 'elcome'
db "to", "asm!"
db "Bye", 'b', 'ye!'
dw 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0ah, 0bh, 0ch, 0dh, 0eh, 0fh
dd 16 dup (0)
data ends
```

下列说法正确的是\_\_\_\_\_。

- 1、 该程序加载后其中的字符数据和数字数据在内存中都是以ASCII码的形式存放的。
- 2、 该程序加载后这组数据在内存中所占用实际空间是90HB。
- 3、 该数据段定义可放在该程序中end伪指令之前的任何位置。
- 4、 在编译阶段，该程序因如此定义数据而报错。

(2) 补全程序，实现从内存1000:0处开始执行指令\_\_\_\_\_。

```
assume cs:code
stack segment
db 16 dup(0)
stack ends
code segment
start: mov ax, stack
      mov ss, ax
      mov sp, 16
      mov ax, _____
      push ax
      mov ax, _____
      push ax
      retf
code ends
end start
```

- 1、 cs, ip
- 2、 0, 1000h
- 3、 1000h, 0
- 4、 0, 1000

(3) 对下列程序说法正确的是\_\_\_\_\_。

```
assume cs:codesg
stack segment
dw 10 dup (0)
stack ends
codesg segment
mov ax, 4c00h
int 21h
start: mov ax, stack
mov ss, ax
mov sp, 20
mov ax, 0
push cs
push ax
mov bx, 0
retf
codesg ends
end start
```

- 1、 程序中start之后的汇编指令不能得到执行。
- 2、 程序加载后stack段在内存中实际占用的内存空间是20b。
- 3、 程序的每一条指令都能获得执行，但不能正常返回。
- 4、 指令retf可替换为ret，并能够正常返回。

(4) 下面的程序执行完add ax, 5时，ax中的数值为多少\_\_\_\_\_。

```
assume cs:code
stack segment
dw 8 dup (0)
stack ends
code segment
start: mov ax, stack
mov ss, ax
mov sp, 16
mov ds, ax
mov ax, 0
call word ptr ds:[0EH]
add ax, 2
inc ax
add ax, 5
mov ax, 4c00H
int 21H
code ends
end start
```

- 1、 3
- 2、 8
- 3、 0
- 4、 4c00H

(5) 下面指令执行后，ax中的数值为多少\_\_\_\_\_。

内存地址	机器码	汇编指令
1000:0	b8 00 00	mov ax, 0
1000:3	9a 09 00 00 10	call far ptr s
1000:8	40	inc ax
1000:9	58	s:pop ax
		add ax, ax
		pop bx
		add ax, bx

- 1、 1006h
- 2、 1010h
- 3、 0020h
- 4、 000ch

\*\*\*\*\*

## 第11章 标志寄存器

(1) 下列指令对标志寄存器内容产生影响，但不会改变参与其执行过程的其它寄存器内容的是\_\_\_\_\_。

- 1、 loop s
- 2、 cmp cx, 0
- 3、 jcxz s
- 4、 div cx

(2) 对于Intel8086cpu标志寄存器的说法正确的是\_\_\_\_\_。

- 1、 cpu执行每一条指令都有可能改变标志寄存器的内容。
- 2、 无符号数运算的进位或借位情况由cf标志位记录，有符号数运算的溢出情况由of标志位记录。
- 3、 当前指令对标志寄存器的影响为此指令的执行提供了所需的依据，对后续指令没有任何指导作用。
- 4、 所有的条件转移指令都要参考标志寄存器中的相关标志位。

(3) sub ax, ax

mov ax, 5

add ax, -3

以上程序执行完后，CF, OF的值是\_\_\_\_\_。

- 1、 0, 0
- 2、 0, 1
- 3、 1, 0
- 4、 1, 1

(4) mov ax, 8

mov bx, 3

cmp ax, bx

上面指令执行前ZF, PF的值是0, 0, 执行后ZF, PF的值是\_\_\_\_\_。

- 1、 0, 1



- 2、 0,0
- 3、 1,0
- 4、 1,1

(5) mov ax,0

push ax

popf

mov ax,0fff0h

add ax,0010h

pushf

pop ax

and al,11000101B

and ah,00001000B

上面指令执行后ax的值是\_\_\_\_\_。

- 1、 845H
- 2、 8C5H
- 3、 45H
- 4、 C5H

\*\*\*\*\*

## 第12章 内中断

\*\*\*\*\*

## 第13章 INT指令

(1)用7ch中断例程模拟loop指令的功能，指令序列如下：

lp: push bp

mov bp, sp

dec cx

jcxz lpret

add [bp+2],bx

lpret: pop bp

iret

关于7ch中断下列说法正确的是\_\_\_\_\_。

- 1、此中断的最大转移位移为128
- 2、此中断的最大转移位移为1K
- 3、此中断根据bx中的偏移量实现跳转
- 4、此中断不能设定跳转的次数

\*\*\*\*\*

## 第14章 端口

(1)下列各代码序列，要从端口号为6FFh的端口读取一个字节的数，正确的是\_\_\_\_\_。

- 1、 mov bx,6FFH  
out al,bx
- 2、 in al,6FFH
- 3、 mov dx,6FFH  
in al,dx
- 4、 mov dx,6FFH

```
out al,dx
```

(2) 以下关于移位错误的是\_\_\_\_\_。

- 1、 shl的功能是将寄存器或内存单元的数据向左移位。
- 2、 shl移位时将最后移出的一位放入OF中。
- 3、 shr把最高位用0补充。
- 4、 把al中的数据向左移3位的代码是

```
mov cl,3  
shl al,cl
```

(3) 下列说法正确的是\_\_\_\_\_。

- 1、 我们可以编程改变FFFF:0处的指令，使CPU不去执行BIOS中的硬件系统检测和初始化程序。
- 2、 int 19h 中断例程可以由DOS提供。
- 3、 中断例程都是由BIOS提供的。
- 4、 CPU可以直接读取端口中的数据。

(4) 下列指令序列能读取CMOS中的2号单元内容的是\_\_\_\_\_。

- 1、

```
mov ax,0c000h  
mov ds,ax  
mov bx,2  
mov al,[bx]
```
- 2、

```
out 70h,2  
in al,71h
```
- 3、

```
mov al,2  
out 70h,al  
in al,71h
```
- 4、

```
mov ax,70h  
out 2,ax  
mov ax,71h  
in ax,71h
```

\*\*\*\*\*

## 第15章 外中断

(1) 下面的说法正确的是\_\_\_\_\_。

- 1、 外设的输入随时都能获得CPU的处理。
- 2、 计算机外设的输入不直接送入CPU，而是直接送入内存。
- 3、 标志寄存器IF位决定着中断信息能否被CPU处理。
- 4、 以上说法都有错误。

(2) 以下说法错误的是\_\_\_\_\_。

- 1、 CPU通过总线和端口来与外部设备进行联系。
- 2、 不可屏蔽中断过程的第一步是取中断类型码。
- 3、 外部可屏蔽中断的中断类型码是通过数据总线送入CPU的。
- 4、 中断过程中将IF置为0是为了在中断过程中禁止其他的可屏蔽中断。

(3) 下列关于9号中断的说法正确的是\_\_\_\_\_。

- 1、 9号中断是不可屏蔽中断。

- 2、 9号中断的作用是读出按键的扫描码，如果是字符转换成字符码，并将其显示在屏幕上。
- 3、 9号中断例程是DOS提供的。
- 4、 9号中断例程中肯定包含读端口数据指令in al,60h。

(4)下列指令中不会影响标志寄存器内容的是\_\_\_\_\_。

- 1、 cld
- 2、 or al,0fh
- 3、 nop
- 4、 shl al,1

\*\*\*\*\*

#### 第16章 直接定址表

(1)将标号a处的8个数累加的和放到标号b处所在的单元中，下面选项中的程序正确的是\_\_\_\_\_。

- 1、

```
assume cs:code
code segment
mov si,0
mov cx,8
s: add b,a[si]
inc si
loop s
mov ax,4c00h
int 21h
a db 1,2,3,4,5,6,7,8
b db 0
code ends
end
```
- 2、

```
assume cs:code,ds:data
data segment
a db 1,2,3,4,5,6,7,8
b db 0
data ends
code segment
start:
mov si,0
mov cx,8
s: mov al,a[si]
add b,al
inc si
loop s
mov ax,4c00h
int 21h
code ends
end start
```
- 3、

```
assume cs:code
data segment
a db 1,2,3,4,5,6,7,8
b db 0
```

```

data ends
code segment
mov ax,data
mov ds,ax
mov si,0
mov cx,8
start:
mov al,a[si]
add b,al
inc si
loop start
mov ax,4c00h
int 21h
code segment
end

```

4、 assume cs:code,ds:data

```

code segment
p: mov si,0
mov cx,8
mov ax,data
mov ds,ax
s: mov al,a[si]
mov ah,0
add b,ax
inc si
loop s
mov ax,4c00h
int 21h
code ends
data segment
a db 1,2,3,4,5,6,7,8
b dw 0
data ends
end p

```

(2) 下列说法中正确的是\_\_\_\_\_。

- 1、 当数据的标号不在代码段时，只要用assume伪指令将数据段和相应的段寄存器连接起来就可以了。
- 2、 数据标号和地址标号唯一的区别就是，数据标号既表示内存单元的地址，还表示内存单元的长度，而地址标号只表示内存单元的地址。
- 3、 直接定址表只可以存储数据的地址，不可以存储程序段的地址。
- 4、 在中断服务程序用到直接定址表时，和不在中断服务程序的使用一样，直接调用即可。

(3) 关于直接定址表描述错误的是\_\_\_\_\_。

- 1、 直接定址表和数据标号一样只是用来标记地址的。
- 2、 直接定址表可以方便的编写一些查表类的程序。
- 3、 直接定址表中包含了数据长度信息。

4、直接定址表的数据可以定义在代码段。

\*\*\*\*\*

#### 第17章 使用BIOS进行键盘输入和磁盘读写

(1)把键盘的扫描码读入并将其转化成ASCII码或状态信息，存储在内存的指定位置的中断例程是\_\_\_\_\_。

- 1、 int 16h
- 2、 int 21h
- 3、 int 9h
- 4、 int 10h

(2)提供读取键盘缓冲区功能的BIOS中断例程是\_\_\_\_\_。

- 1、 int 9h
- 2、 int 21h
- 3、 int 10h
- 4、 int 16h

(3)当有键按下时，将按键的\_\_\_\_\_依次存储在键盘缓冲区中。

- 1、 通码和断码
- 2、 通码和扫描码
- 3、 通码和ASCII码
- 4、 断码和ASCII码

\*\*\*\*\*

#### 第18章 综合研究

(1)就C语言而言，对于函数int main() {return 1;}和int func() {return 1;}以下说法错误的是\_\_\_\_\_。

- 1、 两个函数对应的可执行程序的返回值传送都通过寄存器AX
- 2、 字符串“main”和“func”在编译过程中都被处理为一个偏移地址
- 3、 函数func()对应的C程序无法通过编译和连接
- 4、 从本质上说，两个函数的具有的功能是一样的，没有任何区别

(2)下列关于C语言变量与内存空间的说法错误的是\_\_\_\_\_。

- 1、 全局变量存储在程序向系统申请的数据段所在的内存空间中
- 2、 局部变量存储在程序向系统申请的栈段所在的内存空间中
- 3、 变量是C语言程序访问内存空间的唯一方式
- 4、 指针变量包涵所指向数据所要占用的内存空间的地址信息及其长度信息

(3)下列无法完成向内存空间写入数据的指令语句是\_\_\_\_\_。

- 1、 \*(char \*)0x2000='a';
- 2、 \*(int \*)(\_BX\*2)='b';
- 3、 int \*c = \*(\_DL\*160+\_DH\*2+1);
- 4、 \*(char far \*) (0xb8000000+\_DL\*160+\_DH\*2)=('d'+1);

(4)下列关于C语言不定形参的说错误的是\_\_\_\_\_。

- 1、 不定形参函数的形式参数的类型可以不一样
- 2、 不定形参函数的定义格式为:返回值类型 函数名(...);

- 3、 不定形参函数的定义格式为:返回值类型 函数名(参数类型,...);
- 4、 不定形参函数中可以通过第一个实参的地址来确定参数的个数及每个参数的类型

摘录的汇编网在线测试题目答案（部分）

	(1)	(2)	(3)	(4)	(5)	(6)
第1章 基础知识	4	2	2	3	2	
第2章 寄存器(CPU工作原理)	2	2	3	1	4	3
第3章 寄存器(内存访问)	4	4	3	3	1	
第4章 第一个程序	2	2				
第5章 [bx]和loop指令	4	1	4	4	3	
第6章 包含多个段的程序	3	3	2	1	4	
第7章 更灵活的定位内存地址的方法	3	3	4			
第8章 数据处理的两个基本问题	3	3	1	4		
第9章 转移指令的原理						
第10章 CALL和RET指令	3	3	4	2	2	
第11章 标志寄存器	2	2			3	
第12章 内中断						
第13章 INT指令	3					
第14章 端口	3	2	4	3		
第15章 外中断	4	2	4	3		
第16章 直接定址表	4	2	1			
第17章 使用BIOS进行键盘输入和磁盘读写	3	4	3			
第18章 综合研究	3	3	3	2		