

第 1 章 基础知识

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。
- (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 1024×8 个 bit，1024 个 Byte。
- (4) 1GB、1MB、1KB 分别是 2^{30} 、 2^{20} 、 2^{10} Byte。 (n^m) 的意思是 n 的 m 次幂。
- (5) 8080、8088、80286、80386 的地址总线宽度分别是 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。

第 2 章 寄存器 答案

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
```

- (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。

解：

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

检测点 2.2

(1) 给定段地址为 0001H，仅通过变化偏移地址寻址，CPU 的寻址范围为 00010H 到 1000FH。

(2) 有一数据存放在内存 20000H 单元中，先给定段地址为 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。

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
```

解：

修改 4 次；第一次在 CPU 读取 “mov ax, bx” 后，第二次在 CPU 读取 “sub ax, ax” 后，第三次在 CPU 读取 “jmp ax” 后，第四次在 CPU 执行完 “mov ax, bx” 后；最后 IP 中的值为 0。

实验 1 查看 CPU 和内存，用机器指令和汇编指令编程

1. 略

2.

(1) 略

(2) 略

(3) 查看内存中的内容。

PC 机主板上的 ROM 中写有一个生产日期，在内存 FFF00H~FFFFFH 的某几个单元中，请找出这个生产日期并试图改变它。

解：内存 FFF00H~FFFFFH 为 ROM 区，内容可读但不可写。

(4) 向内存从 B8100H 开始的单元中填写数据，如：

```
-e B810:0000 01 01 02 02 03 03 04 04
```

请读者先填写不同的数据，观察产生的现象；在改变填写的地址，观察产生的现象。

解：8086 的显存地址空间是 A0000H~BFFFFH，其中 B8000H~BFFFFH 为 80*25 彩色字符模式显示缓冲区，当向这个地址空间写入数据时，这些数据会立即出现在显示器上。

第 3 章 寄存器（内存访问）

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=0000H
mov al, [0002] AX=00E6H
mov bx, 0 BX=0000H
mov bl, [000C] BX=0026H
add al, bl AX=000CH
```

(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 和相关寄存器中的数值。
- 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？

解： 初始值：CS=2000H, IP=0, DS=1000H, AX=0, BX=0

① ②

mov ax, 6622H AX=6622H 其他寄存器保持不变，以下同理

jmp 0ff0:0100 CS=0ff0H, IP=0100H

mov ax, 2000H AX=2000H

mov ds, ax DS=2000H

mov ax, [0008] AX=C389H

mov ax, [0002] AX=EA66H

③ 没有区别，被 CS:IP 指向的信息是程序；被传送、运算等指令操作的是数据。

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，逆序复制到 20000H~2000FH 中。逆序复制的含义如图 3.17 所示(图中内存里的数据均为假设)。

```
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]
```

(2) 补全下面的程序，使其可以将 100000H~1000FH 中的 8 个字，逆序复制到 200000H~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. 预备知识：Debug 的使用

略

2. 实验任务

(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=58EA
add ax,[2] ;ax=5CCA
mov bx,[4] ;bx=30F0
add bx,[6] ;bx=6021

push ax ;sp=00FE;修改的内存单元的地址是 220FE，内容为 5CCA
push bx ;sp=00FC;修改的内存单元的地址是 220FC，内容为 6021
pop ax ;sp=00FE;ax=6021
pop bx ;sp=0100;bx=5CCA

push [4] ;sp=00FE;修改的内存单元的地址是 220FE，内容为 30F0
push [6] ;sp=00FC;修改的内存单元的地址是 220FC，内容为 2F31

```

注：内存中的数据会因机器、环境而异

(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？

解：t 命令为单步中断，CPU 会保护现场，即顺序把标志寄存器、CS、IP 入栈，此题是关于后面章节的中断问题。

第 4 章 第一个程序

实验 3 编程、编译、连接、跟踪

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

解：略

(2) 用 Debug 跟踪 t1.exe 的执行过程，写出每一步执行后，相关寄存器中的内容和栈顶的内容。

解：

```

C:\WINDOWS\system32\CMD.exe - debug e:\t1.exe
-t
AX=2000 BX=0000 CX=0016 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13CA ES=13CA SS=13DA CS=13DA IP=0003  NU UP EI PL NZ NA PO NC
13DA:0003 8ED0          MOV     SS,AX
-t
AX=2000 BX=0000 CX=0016 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13CA ES=13CA SS=2000 CS=13DA IP=0008  NU UP EI PL NZ NA PO NC
13DA:0008 83C40A       ADD     SP,+0A
-t
AX=2000 BX=0000 CX=0016 DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=13CA ES=13CA SS=2000 CS=13DA IP=000B  NU UP EI PL NZ NA PE NC
13DA:000B 58          POP     AX
-t
AX=0000 BX=0000 CX=0016 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=13CA ES=13CA SS=2000 CS=13DA IP=000C  NU UP EI PL NZ NA PE NC
13DA:000C 5B          POP     BX
-t
AX=0000 BX=0000 CX=0016 DX=0000 SP=000E BP=0000 SI=0000 DI=0000
DS=13CA ES=13CA SS=2000 CS=13DA IP=000D  NU UP EI PL NZ NA PE NC
13DA:000D 50          PUSH    AX
-t
AX=0000 BX=0000 CX=0016 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=13CA ES=13CA SS=2000 CS=13DA IP=000E  NU UP EI PL NZ NA PE NC
13DA:000E 53          PUSH    BX
-t
AX=0000 BX=0000 CX=0016 DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=13CA ES=13CA SS=2000 CS=13DA IP=000F  NU UP EI PL NZ NA PE NC
13DA:000F 58          POP     AX
-t
AX=0000 BX=0000 CX=0016 DX=0000 SP=000C BP=0000 SI=0000 DI=0000
DS=13CA ES=13CA SS=2000 CS=13DA IP=0010  NU UP EI PL NZ NA PE NC
13DA:0010 5B          POP     BX
-t
AX=0000 BX=0000 CX=0016 DX=0000 SP=000E BP=0000 SI=0000 DI=0000
DS=13CA ES=13CA SS=2000 CS=13DA IP=0011  NU UP EI PL NZ NA PE NC
13DA:0011 B8004C       MOV     AX,4C00
-

```

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

解：

```

C:\WINDOWS\system32\CMD.exe - debug e:\t1.exe
-d ds:0 ff
13CA:0000 CD 20 FF 9F 00 9A F0 FE-1D F0 4F 03 DD 0D 8A 03 . . . . .0. . . .
13CA:0010 DD 0D 17 03 DD 0D CC 0D-01 01 01 00 02 FF FF FF . . . . .
13CA:0020 FF FF FF FF FF FF FF FF-FF FF FF FF 8A 13 4C 01 . . . . .L.
13CA:0030 9D 12 14 00 18 00 CA 13-FF FF FF FF 00 00 00 00 . . . . .
13CA:0040 05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 . . . . .
13CA:0050 CD 21 CB 00 00 00 00 00-00 00 00 00 20 20 20 . ? . . . .
13CA:0060 20 20 20 20 20 20 20 20-00 00 00 00 20 20 20 . . . . .
13CA:0070 20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 . . . . .
13CA:0080 00 0D 65 3A 5C 74 31 2E-65 78 65 0D 53 54 45 52 ..e:\t1.exe.STER
13CA:0090 3D 41 30 0D 64 64 72 65-73 73 2E 20 20 46 6F 72 =A0.ddress. For
13CA:00A0 20 65 78 61 6D 70 6C 65-3A 0D 20 6F 6E 20 4E 54 example:. on NT
13CA:00B0 56 44 4D 2C 20 73 70 65-63 69 66 79 20 61 6E 20 UDM, specify an
13CA:00C0 69 6E 76 61 6C 69 64 0D-20 6F 6E 6C 79 2E 0D 00 invalid. only...
13CA:00D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 . . . . .
13CA:00E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 . . . . .
13CA:00F0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 . . . . .

```

第 5 章 [BX]和 loop 指令

实验 4 [bx]和 loop 的使用

(1)编程，向内存 0:200~0:23F 依次传送数据 0~63(3FH)。

解：

```
assume cs:codesg
```

```
codesg segment
```

```
mov ax,0
```

```
mov ds,ax
```

```
mov bx,200H
```

```
mov al,0
```

```
mov cx,64
```

```
s:mov [bx],al
```

```
inc bx
```

```
inc al
```

```
loop s
```

```
mov ax,4c00h
```

```
int 21h
```

```
codesg ends
```

```
end
```

(2)编程，向内存 0:200~0:23F 依次传送数据 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”。

解:

```
assume cs:codesg
```

```
codesg segment
```

```
mov ax, 20h
```

```
mov ds, ax
```

```
mov bx, 0
```

```
mov cx, 64
```

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

```
inc bx
```

```
loop s
```

```
mov ax, 4c00h
```

```
int 21h
```

```
codesg ends
```

```
end
```

(3) 下面的程序的功能是将“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, cs
```

```
mov ds, ax
```

```
mov ax, 0020h
```

```
mov es, ax
```

```
mov bx, 0
```

```
mov cx, 17h
```

```
s:mov al, [bx]
```

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

```
inc bx
```

```
loop s
```

```
mov ax, 4c00h
```

```
int 21h
```

```
code ends
```

end

第 6 章 包含多个段的程序

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
```

(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, 0 ;10 个字单元用栈空间
```

```
start:
```

```
mov ax, cs
```

```
mov ss, ax
```

```
mov sp, 36
```

```
mov ax, 0
```

```
mov ds, ax
```

```
mov bx, 0
```

```
mov cx, 8
```

```
s:
```

```
push [bx]
```

```
pop cs:[bx]
```

```
add bx, 2
```

```
loop s
```

```
mov ax, 4c00h
```

```
int 21h
```

```
codesg ends
```

```
end start
```

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
```

- ① CPU 执行程序，程序返回前，data 段中的数据为多少？ 解：不变
- ② CPU 执行程序，程序返回前，cs=155ch、ss=155bh、ds=155ah。（此题结果因环境而异，但相对差值不变）
- 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。

(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

```

- ① CPU 执行程序，程序返回前，data 段中的数据为多少？ 解：不变
- ② CPU 执行程序，程序返回前，cs=155ch、ss=155bh、ds=155ah。（此题结果因环境而异，但相对差值不变）
- 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，这段实际占有的空间为 $(N/16+1)*16$ 。（N/16 为取整数部分）

- (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

```

- ① CPU 执行程序，程序返回前，data 段中的数据为多少？ **解：不变**
- ② CPU 执行程序，程序返回前，cs=155ah、ss=155eh、ds=155dh。（此题结果因环境而异，但相对差值不变）
- 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。
- (4) 如果将(1)、(2)、(3)题中的最后一条伪指令“end start”改为“end”（也就是说指明程序的入口），则那个程序仍然可以正确执行？请说明原因。

解：(1)、(2)不能正确执行（入口默认为 data 段的第一条指令），(3)能正确执行。如果不指明程序的入口，编译器自动默认整个代码的第一条指令为程序的入口。

（经 qingxhl 指正，在此鸣谢）

- (5) 程序如下，编写 code 段中的内容，将 a 段和 b 段中的数据依次相加，将结果存到 c 段中。

```
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 bx,0

mov cx,8
s:
mov al,ds:[bx]
add al,ds:[bx+16]
mov ds:[bx+32],al
inc bx
loop s

mov ax,4c00h
int 21h

code ends

end start

```

(6) 程序如下,编写 code 段中的代码,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, 9, 0ah, 0bh, 0ch, 0dh, 0eh, 0fh, 0ffh

a ends

b segment

dw 0, 0, 0, 0, 0, 0, 0, 0

b ends

code segment

start:
mov ax, a
mov ds, ax
mov bx, 0

mov ax, b
mov ss, ax
mov sp, 16

mov cx, 8
s:
push [bx]
inc bx
inc bx
loop s

mov ax, 4c00h
int 21h

code ends

end start

```

第 7 章 更灵活的定位内存地址的方法

实验 6 实践课程中的程序

- (1) 略
- (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 个字母改写为大写字母。


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

```
stacksg segment
```

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

```
stacksg ends
```

```
datasg segment
```

```
db '1. display '
```

```
db '2. brows '
```

```
db '3. replace '
```

```
db '4. modify '
```

```
datasg ends
```

```
codesg segment
```

```
start:
```

```
mov ax,stacksg
```

```
mov ss,ax
```

```
mov sp,16
```

```
mov ax,datasg
```

```
mov ds,ax
```

```
mov bx,0
```

```
mov cx,4
```

```
s0:
```

```
push cx
```

```
mov si,0
```

```
mov cx,4
```

```
s:
```

```
mov al,[bx+si+3]
```

```
and al,11011111b
```

```
mov [bx+si+3],al
```

```
inc si
```

```
loop s
```

```
add bx,16
```

```

pop cx
loop s0

mov ax, 4c00h
int 21h

codesg ends

end start

```

第 8 章 数据处理的两个基本问题

实验 7 寻址方式在结构化数据访问中的应用

编程，将 data 段中的数据按如下格式写入到 table 段中，并计算 21 年中的收入(取整)，结果也按照下面的格式保存在 table 段中。

解：

```
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'
```

;以上是表示 21 年的 21 个字符串

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

```
dd
```

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

;以上是表示 21 年公司总收的 21 个 dword 型数据

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

```
dw 11542, 14430, 45257, 17800
```

;以上是表示 21 年公司雇员人数的 21 个 word 型数据

```

data ends

table segment

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

table ends

codesg segment

start:
mov ax, data
mov ds, ax
mov si, 0

mov ax, table
mov es, ax
mov di, 0

mov cx, 21
s:
mov ax, ds:[si] ;年份转送
mov es:[di], ax
mov ax, ds:[si+2]
mov es:[di+2], ax

mov ax, ds:[si+84] ;收入转送
mov es:[di+5], ax
mov dx, ds:[si+84+2]
mov es:[di+7], dx

push cx ;保护 cx
mov cx, ds:[84+84+bx] ;雇员数转送
mov es:[di+0ah], cx

div cx ;计算人均收入
pop cx

mov es:[di+0dh], ax ;人均收入转送

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

```

```
loop s

mov ax, 4c00h

int 21h

codesg ends

end start
```

第9章 转移指令的原理

检测点 9.1

(1) 程序如下。

```
assume cs:code

data segment
db 0, 0, 0
data ends

code segment

start: mov ax, data
mov ds, ax
mov bx, 0
jmp word ptr [bx+1]

code ends
end start
```

若要使程序中的 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？

(2) 程序如下。

```
assume cs:code, ds:data

data segment
dd 12345678h
data ends
```

```

code segment

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

code ends

end start

```

补全程序，使 jmp 指令执行后，CS:IP 指向程序的第一条指令。

(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)=? ， (IP)=?

解：CS=0006H, IP=00BEH

检测点 9.2

补全编程，利用 jcxz 指令，实现在内存 2000H 段中找查第一个值为 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
inc bx
jmp short s
ok: mov dx, bx
mov ax, 4c00h
int 21h
code ends
end start

```

检测点 9.3

补全程序，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

```

实验 8 分析一个奇怪的程序

分析下面的程序，在运行前思考：这个程序可以正确返回吗？

运行后再思考：为什么是这种结果？

通过这个程序加深对相关内容的理解。

```
assume cs:codesg
```

```
codesg segment
```

```
mov ax,4c00h
```

```
int 21h
```

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

```
s: nop
```

```
nop
```

```
mov di,offset s
```

```
mov si,offset s2
```

```
mov ax,cs:[si]
```

```
mov cs:[di],ax
```

```
s0: jmp short s
```

```
s1: mov ax,0
```

```
int 21h
```

```
mov ax,0
```

```
s2: jmp short s1
```

```
nop
```

```
codesg ends
```

```
end start
```

解：可以正常返回，`jmp short s1` 的机器码是 `EBF6`，即使当前的 `IP=IP-10`，将这条指令移动到 `s:` 处后，`jmp short s1` 不会指到 `s1` 了，而是指到相对当前位置 (`jmp short s1` 的下一条指令) 的 `-10` 的位置 (`mov ax,4c00h`)，所以这个程序可以正常返回。

实验 9 根据材料编程

编程：在屏幕中间分别显示绿色、绿底红色、白底蓝色的字符串 'welcome to masm!'。

解：

```

assume cs:code

data segment
db 'welcome to masm!'
data ends

code segment

start: mov ax,data
mov ds,ax

mov ax,0b800h
mov es,ax

mov si,0
mov di,10*160+80 ;第十行中间
mov cx,16
s1: mov al,ds:[si]
mov ah,00000010B ;绿色
mov es:[di],ax
inc si
inc di
inc di
loop s1

mov si,0
mov di,11*160+80 ;第十一行中间
mov cx,16
s2: mov al,ds:[si]
mov ah,00100100B ;绿底红色
mov es:[di],ax
inc si
inc di
inc di
loop s2

mov si,0
mov di,12*160+80 ;第十二行中间
mov cx,16
s3: mov al,ds:[si]
mov ah,01110001B ;白底蓝色
mov es:[di],ax
inc si
inc di
inc di

```



```

loop s3

mov ax,4c00h
int 21h ;如果要看到完整的显示请输入：“-g 4c”，即立即运行到此条指令

code ends

end start

```

注：此程序如果利用后面所学知识，可以将三次显示嵌套简化为一次。

第 10 章 CALL 和 RET 指令

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

```

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
1000:0	b8 00 00	mov ax, 0
1000:3	e8 01 00	call s

```
1000:6 40 inc ax
1000:7 58 s:pop ax
```

解: ax=6

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
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
```

解: ax=1010h

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
1000:0 b8 06 00 mov ax,6
1000:2 ff d0 call ax
1000:5 40 inc ax
1000:6 mov bp,sp
add ax,[bp]
```

解: ax=11

检测点 10.5

(1) 下面的程序执行后, ax 中的数值为多少?

注: 不能用单步中断测试程序, 中断涉及堆栈操作, 不能带便 CPU 的真实执行结果。

```
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

```

解: ax=3

(2) 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 word ptr ss:[0], offset s
mov ss:[2], cs
call dword ptr ss:[0]
nop
s: mov ax, offset s
sub ax, ss:[0cH]
mov bx, cs
sub bx, ss:[0eH]
mov ax, 4c00h
int 21h
code ends

end start

```

解: ax=1, bx=0

实验 10 编写子程序

1. 显示字符串

;名称: show_str
;功能: 在屏幕的指定位置, 用指定颜色, 显示一个用 0 结尾的字符串
;参数: (dh)=行号, (dl)=列号(取值范围 0~80), (cl)=颜色, ds: si:
该字符串的首地址
;返回: 显示在屏幕上

```
assume cs:daima
shuju segment
db 'fghfghf',0
shuju ends
```

```
daima segment
kaishi:
mov dh,8
mov dl,21
mov cl,2
mov ax,shuju
mov ds,ax
mov si,0
call show_str
mov ax,4c00h
int 21h
;-----
show_str:
push ax
push cx
push dx
push es
push si
push di

mov ax,0b800h
mov es,ax

dec dh
```

```

mov al,160
mul dh
add dl,dl
mov dh,0 ;计算显示在屏幕位置
add ax,dx
mov di,ax

mov ah,cl

x:
mov cl,ds:[si]
mov ch,0
jcxz f
mov al,cl
mov es:[di],ax
inc si
inc di
inc di
jmp x

f:
pop di
pop si
pop es
pop dx
pop cx
pop ax
ret
;-----
daima ends
end kaishi

```

2. 解决除法溢出问题

;名称: divdw
 ;功能: 除法, 被除数 32 位, 除数 16 位, 商 32 位, 余数 16 位, 不会溢出
 ;参数: (dx) = 被除数高 16 位, (ax) = 被除数低 16 位, (cx) = 除数
 ;返回: (dx) = 商高 16 位, (ax) = 商低 16 位, (cx) = 余数

```

assume cs:daima
daima segment
kaishi:
mov ax,2390

```

```

mov dx,0
mov cx,10
call divdw
mov ax,4c00h
int 21h
;-----
divdw:
push bx
push ax
mov ax,dx
mov dx,0
div cx
mov bx,ax
pop ax
div cx
mov cx,dx
mov dx,bx
pop bx
ret
;-----
daima ends
end kaishi

```

3. 数值显示

;名称: dtoc_word
 ;功能: 将一个 word 型数转化为字符串
 ;参数: (ax) =word 型的数据, ds:si 指向字符串的首地址
 ;返回: ds:[si]放此字符串, 以 0 结尾

```
assume cs:daima
```

```
shuju segment
db 20 dup(1)
shuju ends
```

```
daima segment
kaishi:
mov ax,shuju
mov ds,ax
mov ax,10100
```

```
call dtoc_word
```

```
mov ax,4c00h
```

```
int 21h
```

```
;
```

```
dtoc_word:
```

```
push ax
```

```
push bx
```

```
push cx
```

```
push dx
```

```
push si
```

```
mov bx,0
```

```
x:
```

```
mov dx,0
```

```
mov cx,10
```

```
div cx
```

```
mov cx,ax
```

```
add dx,'0'
```

```
push dx
```

```
inc bx
```

```
jcxz f
```

```
jmp x
```

```
f:
```

```
mov cx,bx
```

```
x1:
```

```
pop ds:[si]
```

```
inc si
```

```
loop x1
```

```
pop si
```

```
pop dx
```

```
pop cx
```

```
pop bx
```

```
pop ax
```

```
ret
```

```
;
```

```
daima ends
```

end kaishi

课程设计 1

任务：将实验 7 中的 Power idea 公司的数据按照图 10. 所示的格式在屏幕上显示出来。

解：

;注：函数中的标号为防止冲突，都加了本函数名为前缀

;在 Debug 中输入 “-g 90”，直接运行到结束

assume cs:code

data segment

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

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

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

;以上是表示 21 年的 21 个字符串

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

dd

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

;以上是表示 21 年公司总收的 21 个 dword 型数据

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

dw 11542, 14430, 45257, 17800

;以上是表示 21 年公司雇员人数的 21 个 word 型数据

data ends

agency segment

db 8 dup(0)

agency ends


```

code segment

start: mov ax, 0b800h
mov es, ax
mov di, 0
mov cx, 80*24
x: mov byte ptr es:[di], ' ' ;将屏幕清空
mov byte ptr es:[di+1], 0
inc di
inc di
loop x

mov ax, data
mov es, ax
mov di, 0
mov bx, 0

mov ax, agency
mov ds, ax
mov si, 0

mov dh, 4

mov cx, 21
x1: push cx

mov ax, es:[di]
mov ds:[si], ax
mov ax, es:[di+2]
mov ds:[si+2], ax
mov byte ptr ds:[si+4], 0 ;显示年份
mov dl, 0
mov cl, 2
call show_str

mov ax, es:[84+di]
push dx
mov dx, es:[84+di+2]
call dtoc_dword ;显示收入
pop dx
mov dl, 20
mov cl, 2
call show_str

```

```

mov ax,es:[84+84+bx]
call dtoc_word
mov dl,40 ;显示雇员数
mov cl,2
call show_str

```

```

mov ax,es:[84+di]
push dx
mov dx,es:[84+di+2]
div word ptr es:[84+84+bx] ;计算人均收入并显示
call dtoc_word
pop dx
mov dl,60
mov cl,2
call show_str

```

```

add di,4
add bx,2
add dh,1

```

```

pop cx
loop x1

```

```

mov ax,4c00h
int 21h

```

;名称: show_str
 ;功能: 在屏幕的指定位置, 用指定颜色, 显示一个用 0 结尾的字符串
 ;参数: (dh)=行号, (dl)=列号(取值范围 0~80), (cl)=颜色, ds: si:
 该字符串的首地址
 ;返回: 显示在屏幕上

```

show_str:
push ax
push cx
push dx
push es
push si
push di

```

```

mov ax,0b800h

```

```

mov es, ax

mov al, 160
mul dh
add dl, dl
mov dh, 0
add ax, dx
mov di, ax

mov ah, cl

show_str_x:
mov cl, ds:[si]
mov ch, 0
jcxz show_str_f
mov al, cl
mov es:[di], ax
inc si
inc di
inc di
jmp show_str_x

show_str_f:
pop di
pop si
pop es
pop dx
pop cx
pop ax
ret

;名称: dtoc_word
;功能: 将一个 word 型数转化为字符串
;参数: (ax) =word 型的数据, ds:si 指向字符串的首地址
;返回: ds:[si]放此字符串, 以 0 结尾
dtoc_word:
push ax
push bx
push cx
push dx
push si

mov bx, 0
dtoc_word_x:
mov dx, 0

```

```

mov cx,10
div cx
mov cx,ax
add dx,'0'
push dx
inc bx
jcxz dtoc_word_f
jmp dtoc_word_x

```

```

dtoc_word_f:
mov cx,bx
dtoc_word_x1:
pop ds:[si]
inc si
loop dtoc_word_x1

```

```

pop si
pop dx
pop cx
pop bx
pop ax

```

```

ret

```

;名称: dtoc_dword
;功能: 将一个 double word 型数转化为字符串
;参数: (dx)=数的高八位, (ax)=数的低八位
;返回: ds:[si]放此字符串, 以 0 结尾
;备注: 会用到 divdw 函数

```

dtoc_dword:
push ax
push bx
push cx
push dx
push si

```

```

mov bx,0
dtoc_dword_x:
mov cx,10
call divdw
push cx
inc bx
cmp ax,0
jne dtoc_dword_x

```

```
cmp dx,0
jne dtoc_dword_x
```

```
mov cx,bx
dtoc_dword_x1:
pop ds:[si]
add byte ptr ds:[si], '0'
inc si
loop dtoc_dword_x1
```

```
pop si
pop dx
pop cx
pop bx
pop ax
```

```
ret
```

;名称: divdw

;功能: 除法, 被除数 32 位, 除数 16 位, 商 32 位, 余数 16 位, 不会溢出

;参数: (dx) =被除数高 16 位, (ax) =被除数低 16 位, (cx) =除数

;返回: (dx) =商高 16 位, (ax) =商低 16 位, (cx) =余数

```
divdw:
push bx
push ax
mov ax,dx
mov dx,0
div cx
mov bx,ax
pop ax
div cx
mov cx,dx
mov dx,bx
pop bx
ret
```

```
code ends
```

```
end start
```

第 11 章 标志寄存器

检测点 11.1

写出下面每条指令后，ZF、PF、SF 等标志位的值。

ZF PF SF

sub al,al 1 1 0

mov al,1 1 1 0

push ax 1 1 0

pop bx 1 1 0

add al,bl 0 0 0

add al,10 0 1 0

mul al 0 1 0

检测点 11.2

CF OF SF ZF PF

sub al,al 0 0 0 1 1

mov al,10H 0 0 0 1 1

add al,90H 0 0 1 0 1

mov al,80H 0 0 1 0 1

add al,80H 1 1 0 1 1

mov al,0FCH 1 1 0 1 1

add al,05H 1 0 0 0 0

mov al,7DH 1 0 0 0 0

add al,0BH 0 1 1 0 1

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]的数据的个数。

mov ax,0f000h

mov ds,ax

mov bx,0

mov dx,0

mov cx,32

s:mov al,[bx]

cmp al,32

jb s0

cmp al,120

ja s0

inc dx

s0:inc bx

loop s

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

```
mov ax, 0f000h
mov ds, ax
```

```
mov bx, 0
mov dx, 0
mov cx, 32
s: mov al, [bx]
cmp al, 32
jna s0
cmp al, 120
jnb s0
inc dx
s0: inc bx
loop s
```

检测点 11.4

下面的程序执行后：(ax)=?

```
mov ax, 0
push ax
popf
mov ax, 0fff0h
add ax, 0010h
pushf
pop ax
and al, 11000101B
and ah, 00001000B
```

解：(ax)=01000101B

实验 11 编写子程序

;名称: letterc
;功能: 将以 0 结尾的字符串中的小写字母转变成大写字母
;参数: ds:si 开始存放的字符串
;返回: ds:si 开始存放的字符串

```
assume cs:codesg
```

```

datasg segment
db "Beginner's All-purpose Symbolic Instruction Code.",0
datasg ends

codesg segment

begin:
mov ax,datasg
mov ds,ax
mov si,0

call letterc

mov ax,4c00h
int 21h

letterc:
push si
push ax

x: mov al,ds:[si]
cmp al,0
je f
inc si
cmp al,'a'
jb x
cmp al,'z'
ja x
add al,'A'-'a'
mov ds:[si-1],al
jmp x

f: pop ax
pop si
ret

codesg ends

end begin

```

第 12 章 内中断

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的入口地址为：0070:018B 。

(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 。

实验 12 编写 0 号中断的处理程序

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

解：

```
assume cs:code

code segment

start:
mov ax,cs
mov ds,ax
mov si,offset do

mov ax,0
mov es,ax
mov di,200h

mov cx,offset doend-offset do ;安装中断例程
cld
rep movsb

mov word ptr es:[0],200h
mov word ptr es:[2],0 ;设置中断向量表

mov dx,0ffffh
mov bx,1 ;测试一下
div bx

mov ax,4c00h
int 21h

do: jmp short dostart
```

```

db 'divide error!'
dostart:
mov ax, 0
mov ds, ax
mov si, 202h

mov ax, 0b800h
mov es, ax
mov di, 160*10+80

mov cx, 13
s:
mov al, ds:[si]
mov ah, 2
mov es:[di], ax
inc si
inc di
inc di
loop s

mov ax, 4c00h
int 21h

doend:nop

code ends

end start

```

第 13 章 int 指令

检测点 13.1

(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 中断例程能进行的最大转移位移是多少?

解: 8000H~7FFFH 即 (-32768~32767)

(2) 用 7ch 中断例程完成 jmp near ptr s 指令的功能, 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,cs
mov ds,ax
mov si,offset jp

mov ax,0
mov es,ax
mov di,200h

mov cx,offset jpend-offset jp ;安装中断例程
cld
rep movsb

mov word ptr es:[7ch*4],200h
mov word ptr es:[7ch*4+2],0 ;设置中断向量表


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 ;测试 int 7ch
int 7ch
ok:mov ax,4c00h
int 21h

jp:push bp

```

```

mov bp, sp
add [bp+2], bx ;中断例程
pop bp
iret
jpend: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。因为该内存单元具有‘只读’属性。

(2) int 19h 中断例程，可以由 DOS 提供。

答：这种说法是错误的。因为 int 19h 是在 DOS 启动之前就被执行的中断例程，是由 BIOS 提供的。

实验 13 编写、应用中断例程

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

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

```

assume cs:code

data segment
db 'welcome to masm!',0
data ends

code segment

start:

mov ax, cs

```

```

mov ds, ax
mov si, offset dp

mov ax, 0
mov es, ax
mov di, 200h

mov cx, offset dpend - offset dp ;安装中断例程
cld
rep movsb

mov word ptr es:[7ch*4], 200h
mov word ptr es:[7ch*4+2], 0 ;设置中断向量表


mov dh, 10
mov dl, 10
mov cl, 2
mov ax, data
mov ds, ax ;测试 int 7ch
mov si, 0
int 7ch

mov ax, 4c00h
int 21h

dp:
mov al, 160
mul dh
add dl, dl
mov dh, 0
add ax, dx
mov di, ax

mov ax, 0b800h
mov es, ax
;中断例程
s:
mov al, ds:[si]
mov ah, 0
cmp ax, 0
je f
mov ah, cl
mov es:[di], ax
inc si

```

```

inc di
inc di
jmp s

f:
iret
dpend:nop

code ends

end start

```

(2) 编写并安装 int 7ch 中断例程，功能为完成 loop 指令的功能。

参数：(cx)=循环次数，(bx)=位移

```

assume cs:code

code segment

start:

mov ax,cs
mov ds,ax
mov si,offset lp

mov ax,0
mov es,ax
mov di,200h

mov cx,offset lpend-offset lp ;安装中断例程
cld
rep movsb

mov word ptr es:[7ch*4],200h
mov word ptr es:[7ch*4+2],0 ;设置中断向量表

mov ax,0b800h
mov es,ax
mov di,160*12
mov bx,offset s-offset se
mov cx,80

```

```

s:
mov byte ptr es:[di], '!' ;测试 int 7ch
add di, 2
int 7ch
se:
nop

mov ax, 4c00h
int 21h

lp:
push bp

dec cx
jcxz f
mov bp, sp
add [bp+2], bx ;中断例程

f:
pop bp
iret

lpend:nop

code ends

end start

```

(3) 下面的程序，分别在屏幕的第 2、4、6、8 行显示四句英文诗，补全程序。

```

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
    mov dh, [si]
    mov dl, 0
    mov ah, 2
    int 10h

    mov dx, [bx]
    mov ah, 9
    int 21h
    inc si
    add bx, 2
    loop ok

mov ax, 4c00h
int 21h

code ends

end start

```

第 14 章 端口

检测点 14.1

(1)编程：读取 CMOS RAM 的 2 号单元的内容。

解：

```
assume cs:code
```

```
code segment
```

```
start:
```

```

mov al, 2
out 70h, al
in al, 71h

```



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

```
code ends
```

```
end start
```

(2) 编程：向 CMOS RAM 的 2 号单元写入 0。

解：

```
assume cs:code
```

```
code segment
```

```
start:
```

```
mov al,2
out 70h,al
mov al,0
out 71h,al
```

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

```
code ends
```

```
end start
```

检测点 14.2

编程：用加法和移位指令计算 $(ax) = (ax) * 10$

提示： $(ax) * 10 = (ax) * 2 + (ax) * 8$

解：

```
assume cs:code
```

```
code segment
```

```
start:
```

```
mov ax,2
```

```
shl ax,1
mov bx,ax
shl ax,1
```

```

shl ax,1
add ax,bx

mov ax,4c00h
int 21h

code ends

end start

```

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。

解：

```

assume cs:code

data segment

time db 'yy/mm/dd hh:mm:ss$' ;int 21h 显示字符串，要求以$结尾
table db 9,8,7,4,2,0 ;各时间量的存放单元

data ends

code segment
start:
mov ax,data
mov ds,ax
mov si,offset table
mov di,offset time

mov cx,6
s:
push cx
mov al,ds:[si] ;读端口
out 70h,al
in al,71h

mov ah,al
mov cl,4
shr ah,cl ;将压缩 BCD 码分为两个 BCD 码

```

```

and al,00001111b

add ah,30h ;变为字符
add al,30h

mov ds:[di],ah
mov ds:[di+1],al ;写进 time

inc si
add di,3

pop cx
loop s

mov ah,0
mov bh,0
mov dh,10 ;置光标于 10 行 40 列
mov dl,40
int 10h

mov dx,offset time
mov ah,9 ;显示字符串
int 21h

mov ax,4c00h
int 21h

code ends

end start

```

第 15 章 外中断

检测点 15.1

(1) 仔细分析一下上面的 int 9 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 都已经置 0，没有必要再进行设置可。对于程序段：

```
pushf
```

```
pushf
pop ax
and ah,11111100b
push ax
popf
call dword ptr ds:[0]
```

可以精简为：

```
pushf
call dword ptr ds:[0]
```

两条指令。

(2) 仔细分析上面程序中的主程序[第 269 页]，看看有什么潜在的问题？
在主程序中，如果在执行设置 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，改写它们，排除潜在的问题。
提示：注意 sti 和 cli 指令的用法。

解：

将

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

扩充为：

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

实验 15 安装新的 int 9 中断例程

安装一个新的 int9 中断例程，功能：在 DOS 下，按下“A”键后，除非不再松开，如果松开，就显示满屏幕的“A”；其他的键照常处理。

提示：按下一个键时产生的扫描码称为通码，松开一个键产生的扫描码称为断码。
断码=通码+80H。

解：

```
assume cs:code
```

```

code segment

start:

mov ax,cs
mov ds,ax ;安装自定义的 int9 中断例程
mov ax,0
mov es,ax
mov si,offset int9
mov di,204h
mov cx,offset int9end-offset int9
cld
rep movsb


push es:[9*4]
pop es:[200h]
push es:[9*4+2]
pop es:[202h] ;保存原中断向量


cli
mov word ptr es:[9*4],204h
mov word ptr es:[9*4+2],0 ;设置自定义的中断向量
sti


mov ax,4c00h


int 21h


int9:
push ax
push cx
push es
push di

in al,60h ;读入扫描码

pushf
call dword ptr cs:[200h] ;调用原 int 9 终端

cmp al,1EH+80H ;是否为 A 的断码
jne int9ret

mov ax,0b800h

```

```

mov es, ax
mov di, 0

mov cx, 80*20
s: mov byte ptr es:[di], 'A' ;显示满屏 A
add di, 2
loop s

int9ret: pop di
pop es
pop cx
pop ax

iret

int9end: nop

code ends

end start

```

第 16 章 直接定址表

检测点 16.1

下面的程序将 code 段中 a 处的 8 个数据累加，结果存储到 b 处的双字中，补全程序。

```

assume cs:code
code segment
a dw 1, 2, 3, 4, 5, 6, 7, 8
b dw 0, 0 ;原题是 b dd 0, 0 但这样下面无法体现本节内容，估计是作者写错了，
我在此更正

start: mov si, 0
mov cx, 8
s: mov ax, a[si]
add b, ax
adc 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 code 段中 a 处的 8 个数据累加，结果存储到 b 处的双字中，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

```

实验 16 编写包含多个功能子程序的中断例程

安装一个新的 int 7ch 中断例程，为显示输出提供如下功能子程序。

- (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 传递颜色值， $(a1) \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ 。

解：

；介绍：编写中断例程：显示字符串

```
assume cs:daima
```

```
daima segment
```

```
kaishi:
```

```
mov ax,0
```

```
mov ds,ax
```

```
mov word ptr ds:[7ch*4],200h ;设置中断向量表
```

```
mov word ptr ds:[7ch*4+2],0
```

```
mov ax,cs
```

```
mov ds,ax
```

```
mov si,offset int7ch
```

```
mov ax,0
```

```
mov es,ax
```

```
mov di,200h
```

```
mov cx,offset int7chend-offset int7ch ;安装中断例程
```

```
cld
```

```
rep movsb
```

```
mov ah,2
```

```
mov al,2 ;测试一下
```

```
int 7ch
```

```
mov ax,4c00h
```



```
int 21h
```

ORG 200H ;此程序的点睛之笔，
;伪指令，表示下一条指令从偏移地址 200H 开始，正好和安装后的偏移地址相同
;因为如果没有 ORG 200H，此中断例程被安装以后，标号所代表的地址变了，和之前编译器编译的有别

```
int7ch: jmp short begin  
table dw sub1,sub2,sub3,sub4
```

```
begin: push bx
```

```
cmp ah,3  
ja f  
mov bl,ah  
mov bh,0  
add bx,bx
```

```
call word ptr table[bx]
```

```
f: pop bx  
iret
```

```
sub1: push bx  
push cx  
push es  
mov bx,0b800h  
mov es,bx  
mov bx,0  
mov cx,2000  
subls: mov byte ptr es:[bx],'  
add bx,2  
loop subls  
pop es  
pop cx  
pop bx  
ret
```

```
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],al
add bx,2
loop sub2s

```

```

pop es
pop cx
pop bx
ret

```

```

sub3: 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],al
add bx,2
loop sub3s
pop es
pop cx
pop bx
ret

```

```

sub4: push cx
push si
push di
push es
push ds

```

```

mov si,0b800h
mov es,si
mov ds,si
mov si,160
mov di,0

```

```

cld
mov cx,24

sub4s: push cx
mov cx,160
rep movsb
pop cx
loop sub4s

mov cx,80
mov si,0
sub4s1: mov byte ptr [160*24+si],' '
add si,2
loop sub4s1

pop ds
pop es
pop di
pop si
pop cx
ret

int7chend:nop

daima ends
end kaishi

```