汇编语言答案

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。

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。

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 ax,2000H                    AX=2000H
mov ds,ax                       DS=20000H
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]
```

```
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
```

(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。

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
```

```
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
```

```
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
```

```
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          ;10个字单元用栈空间
start:
    mov ax,cs
```

```
    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
```

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
```

…
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]
```

(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 qingxh1 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
```

```
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
```

```
code ends
end start
```

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

```
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
```

实验7 寻址方式在结构化数据访问中的应用

```
        ;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 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
```

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
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
```

```
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]
```

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
```

```
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
```

```
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。

检测点10.1

补全程序，实现从内存1000:0000处开始执行指令。

```
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
```

```
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
```

```
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
```

```
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
```

```
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
```

```
;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
```

```
;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
```

```
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
```

```
        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 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]
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 ah,0
int 16h ;加上按任意键继续功能,可以直接双击运行
mov ax,4c00h
```

```
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
```

```
;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'
```

```
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
```

```
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
```

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

```
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
```

```
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
```

```
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
```

检测点12.1

(1) 用Debug查看内存，情况如下:

```
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!'
```

```
inc di
inc di
loop s
mov ax,4c00h
int 21h
doend:nop
code ends
end start
```

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
```

```
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
```

(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 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
```

```
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
```

```
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
```

```
        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
```

(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
```

```
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:
```

```
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
```

```
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 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
```

```
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
```

```
        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
```

```
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a1传递颜色值，(a1)∈{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 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
```

```
loop sub1s
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
```

```
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
```

```
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
```