汇编语言答案（王爽）

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、80296、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）1KB=1024B，8KB=1024B*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，1KB=1024B。

（3）8Bit=1Byte，1024Byte=1KB（1KB=1024B=1024B*8Bit）。

（4）1GB=1073741824B（即 2^30）1MB=1048576B（即 2^20）1KB=1024B（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 的 N 次方个内存单元。（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）1024B/2B=512，同理 1024B/4B=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

检测点 2.1

(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

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)~(SA*16+ffffh)

现在 SA=0001h,那么寻址范围为

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

=0010h~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

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

检测点 3.1

(1)　在 DEBUG 中,用 "D 0:0 lf" 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

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；

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]
```

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
```

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
```

（1）程序如下。

```
assume cs:code
data segment
    dw 2 dup (0)
data ends
code segment
```

```
   start: mov ax,dtat
          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

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

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

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 reft 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。

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
```

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=fffeh
		add ax,[bp]	ax=[6+ds:(fffeh)]=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~~~~

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
```

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,bl　　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。

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

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
        jb s0           ;如果低于 al 转到 s0,继续循环
        cmp al,128      ;和 128 进行比较
        ja s0           ;如果高于 al 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 al 转到 s0,继续循环
        cmp al,128         ;和 128 进行比较
        jnb s0             ;如果不低于 al 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

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

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。

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
```

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