汇编语言答案

检测点 1.1(第 8 页)

(1) 13

(2) 1024,0,1023

(3) 8192,1024

(4) 2^30,2^20,2^10

(5) 64,1,16,4

(6) 1,1,2,2,4

(7) 512,256

(8) 二进制

注意:

1.第 4 题中的符号'^'指求幂运算(如: 2^30 指 2 的 30 次方)

检测点 2.1(第 18 页)

---

(1)写出每条汇编指令执行后相关寄存器中的值。

第一空：F4A3H

第二空：31A3H

第三空：3123H

第四空：6246H

第五空：826CH

第六空：6246H

第七空：826CH

第八空：04D8H

第九空：0482H

第十空：6C82H

第十一空：D882H

第十二空：D888H

第十三空：D810H

第十四空：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。

解答如下：

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

检测点 2.2(第 23 页)

---

(1)00010H,1000FH

(2)1001H,2000H

第 2 题说明：

因为段的起始地址要为 16 的倍数。所以当段地址小于 1001H 或大于 2000H 时 CPU 都无法寻到。

检测点 2.3(第 33 页)

答：CPU 修改了 4 次 IP 的值。

情况如下：
第 1 次：执行完 mov ax,bx 后
第 2 次：执行完 sub ax,ax 后
第 3 次：读入 jmp ax 后
第 4 次：执行完 jmp ax 后
最后 IP 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(第 33 页)

1.预备知识：Debug 的使用
<此部分略>
2.实验任务(第 43 页)
(1)
<此部分略>
(2)
<此部分略>
(3)
通过 DEBUG 中的 D 命令查看到主板的生产日期[以月、日、年，分隔符为'/'的格式]存储在内存 ffff:0005~ffff:000C(共 8 个字节单元中)处。此生产日期不能被改变，因为其具有‘只读’属性。
(4)
通过向内存中的显存写入数据，使计算机根据写入的数据进行 ASCII 转换，并将转换后且可打印的字符输出到屏幕上。<注：关于显存的详细讨论不在此题范围>
检测点 3.1(第 52 页)-
(1)(题目：略)
第一空：2662H
第二空：E626H
第三空：E626H
第四空：2662H
第五空：D6E6H
第六空：FD48H
第七空：2C14H
第八空：0000H
第九空：00E6H
第十空：0000H
第十一空：0026H
第十二空：000CH
提示：此题可在 DEBUG 中利用 E 命令在本机上按照题目中所给出的内存单元及其数据进行相应地修改，然后再用 A 命令进行写入(题目中所给出的)相应的汇编指令，最后再进行 T 命令进行逐步执行，以查看相应结果。
(2)
1.指令序列如下：
mov ax,6622h
jmp 0ff0:0100
mov ax,2000h

mov ds,ax
mov ax,[0008]
mov ax,[0002]
2.写出 CPU 执行每条指令后，CS、IP 和相关寄存器中的数值。

指令序列↓	寄存器→	CS	IP	DS	AX	BX
	初始值→	2000H	0000	1000H	0	0
mov ax,6622h		2000H	0003	1000H	6622H	0000
jmp 0ff0:0100		1000H	0000	1000H	6622H	0000
mov ax,2000h		1000H	0003	1000H	2000H	0000
mov ds,ax		1000H	0005	2000H	2000H	0000
mov ax,[0008]		1000H	0008	2000H	C389H	0000
mov ax,[0002]		1000H	000B	2000H	EA66H	0000

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？
答：(略)
检测点 3.2(第 66 页)
(1)
mov ax,2000H
mov ss,ax
mov sp,10H
(2)
mov ax,1000H
mov ss,ax
mov sp,0H
实验 2 用机器指令和汇编指令编程(第 70 页)
1.预备知识：Debug 的使用
<此部分略>
2.实验任务
(1)使用 Debug，将下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。
从第一空开始依次如下：
ax=5BEA
ax=5CCA
bx=30F0
bx=6029
sp=FE 220FE 5CCA
sp=FC 220FC 6029
sp=FE 6029
sp=100H 5CCA
sp=FE 220FE 30F0
sp=FC 220FC 2E39
说明：此题可能因机子软、硬件环境不同而导致答案不一致！
(2)仔细观察图 3.19 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？
答：因为用 T 指令进行调试时，会产生中断。而为了保护现场，CPU 则先将标志寄存器进栈、再把当前 CS 的值进栈，最后将 IP 的值进栈。<关于中断的详细内容的讨论不在此题范

围>
实验 4 [BX]和 loop 的使用(第 113 页)
(1) 编程，向内存 0:200~0:23F 依次传送数据 0~63(3FH)。
程序如下：

```
assume cs:codesg
codesg segment
        mov ax,0020h
        mov ds,ax
        mov bx,0
        mov dl,0
        mov cx,40h
s:      mov [bx],dl
        inc dl
        inc bx
        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”。
程序如下：

```
assume cs:codesg
codesg segment
        mov ax,0020h
        mov ds,ax
        mov bl,0
        mov cx,40h
s:      mov [bx],bl
        inc bl
        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,code    ;code 为所填写的数据
        mov ds,ax
        mov ax,0020h
        mov es,ax
        mov bx,0
```

```
        mov cx,18h      ;18h 为所填写的数据
s:      mov al,[bx]
        mov es:[bx],al
        inc bx
        loop s
        mov ax,4c00h
        int 21h
code ends
end
```

提示：

1.因为题目的要求是把代码段内的指令当作数据，复制到目的地址。所以，源数据段 ds 和代码段 cs 相同，通过 mov ax,code/mov ds,ax ('/'符号是指两条指令的分隔)来设置源数据段。

2.可以先假设要复制 8 位[1h~0ffh]数据(因为我们肉眼就可以看出此程序的长度不可能大于 0ffh 个字节)的字节数(如：10h)，把程序补全，以便通过编译。这时我们以准确的第一空所填内容 code 与假想的第二空内容 10h 将程序补充完整并将其编译、连接、运行，接着进行 DEBUG，在 DEBUG 时我们可用 R 命令查看 CX 的值，这时我们可以看到 CX 的值为 1D，由此我们可以算出该程序的长度[1Dh-5h]=18h，之所以减 5 是为了满足题目的要求(因为 mov ax,4c00h/int 21h 这两条指令的长度等于 5)

检测点 6.1(第 119 页)

(1)

```
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      ;此条指令为所填指令
        add bx,2
        loop s
        mov ax,4c00h
        int 21h
codesg ends
end start
```

(2)

```
assume cs:codesg
codesg segment
        dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h
        dw 0,0,0,0,0
start:
        mov ax,cs       ;cs 为所填第一空
        mov ss,ax
        mov sp,1ah      ;此条指令为所填第二空
        mov ax,0
```

```
        mov ds,ax
        mov bx,0
        mov cx,8
s:
        push [bx]
        pop cs:[bx]       ;此条指令为所填第三空
        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(第 123 页)

(1)

1.保持不变

2.<考虑不同机子环境不同，答案无法统一>

3.X-2，X-1

(2)

1.保持不变

2.<考虑不同机子环境不同，答案无法统一>

3.X-2，X-1

4.(N/16+1)*16 [说明：N/16 只取整数部分]

(3)

1.保持不变

2.<考虑不同机子环境不同，答案无法统一>

3.X+3，X+4

(4)

答：第 3 个仍然可以正确执行。因为如果把 end 指令后的标号 start 去掉后，编译器便会顺序执行程序。换句话说：当未给编译器预先的通知，要求其从哪开始执行程序时，编译器就自动以'至上向下'的顺序进行编译执行源程序。

(5)完整程序如下：

```
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 es,ax
```

```
            mov ax,c
            mov ds,ax
            mov bx,0
            mov cx,8
        s1:mov ax,es:[bx]
            add [bx],ax
            add bx,2
            loop s1
            mov ax,b
            mov es,ax
            mov ds,ax
            mov bx,0
            mov cx,8
        s2:mov ax,es:[bx]
            add [bx],ax
            add bx,2
            loop s2
            mov ax,4c00h
            int 21h
code ends
   end start
```

(6)完整程序如下：

```
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,b
            mov ss,ax
            mov sp,10h
            mov ax,a
            mov ds,ax
            mov bx,0

            mov cx,8
s:          push [bx]
            add bx,2
            loop s
            mov ax,4c00h
            int 21h
```

```
code ends
end start
```
实验 6 实践课程中的程序(第 147 页)

(2)编程：完成问题中的程序。

问题 7.9 完整程序如下：

```
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       '
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
s: ;外循环
        push cx
        mov si,3
        mov cx,4
        s0: ;内循环
                mov al,[bx+si]
                and al,11011111b
                mov [bx+si],al
                inc si
        loop s0
        add bx,16
        pop cx
loop s
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(第 160 页)

完整程序如下：

```
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 ax,table
        mov es,ax
        mov bx,0
        mov si,0
        mov di,0
        mov cx,21
        s:    ;进入循环
                mov al,[bx]
                mov es:[di],al
                mov al,[bx+1]
                mov es:[di+1],al
                mov al,[bx+2]
                mov es:[di+2],al
                mov al,[bx+3]
                mov es:[di+3],al
                ;以上 8 句的作用是存放年份
                mov ax,54h[bx]        ;第一个'年收入'的段基址为 54H
                mov dx,56h[bx]
                mov es:5h[di],ax
                mov es:7h[di],dx
                ;以上 4 句的作用是存放公司总收入
                mov ax,0A8h[si]       ;第一个'人数'的段基址为 0A8H
                mov es:0Ah[di],ax
                ;以上 2 句是存放公司的人数
                mov ax,54h[bx]
```

```
                div word ptr ds:0A8h[si]
                mov es:0dh[di],ax
                ;以上 3 句是存放人均收入

                add bx,4
                add si,2
                add di,16
                ;以上 3 句是为下一次循环时存放数据做准备
                ;3 个寄存器递增的速度决定了所要存取的数据的位置的偏移地址
        loop s    ;跳到标号 s 处
mov ax,4c00h
int 21h
codesg ends
end start
```

程序说明：此程序虽然可以达到预期效果(读者可以自行调试验证)，但实现方法比较简单，读者有兴趣的话可以寻找一种更具结构化的设计方法来完成。

检测点 9.1(第 170 页)

(1)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?

完整程序如下：

```
assume cs:code,ds:data
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]     ;段内间接转移
code ends
end start
;解题理由：为了使 IP 的值经跳转后变为 0，则需保证 ds:[bx+1]处的字型单元数据为 0000H，
;所以定义 3 个字节型数据 0 就符合“应该”的要求
```

(2)补全程序，使 jmp 指令执行后，CS:IP 指向程序的第一条指令。

完整程序如下：

```
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          ;源操作数 bx 为所填内容
    mov [bx+2],cs        ;源操作数 cs 为所填内容
```

```
    jmp dword ptr ds:[0]
code ends
end start
```

(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)=?

提示：为了使本机环境[2000:1000 至 2000:1005]中的数据与题目中所给出的数据一致，可以通过编写程序来完成，完整程序如下：

```
assume cs:code
code segment
start:
    mov ax,2000h
    mov ds,ax
    mov bx,1000h
    mov word ptr [bx].0,0BEH
    mov word ptr [bx].2,6h
    mov word ptr [bx].4,0
    ;运行完上 6 句则使 2000:1000--2000:1005 中的数据依次为:BE,00,06,00,00,00
    ;以上 6 句则按题目中的数据进行初始化，以便使运行环境符合题目要求
    ;mov ax,2000h
    mov es,ax
    jmp dword ptr es:[1000h]
code ends
end start
```

经上机调试得出：CS=0006H,IP=00BEH

检测点 9.2(第 172 页)

从标号 s 处开始所要填写的四条指令依次如下：

第一条指令：mov cl,[bx]

第二条指令：mov ch,0

第三条指令：jcxz ok

第四条指令：inc bx

检测点 9.3(第 173 页)

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 byte，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      ;此条指令为题目要求补全的指令
                inc bx
        loop s
        ok:
                dec bx
                mov dx,bx
                mov ax,4c00h
                int 21h
code ends
end start
```

解答提醒：此题可用假设法来完成(比如设 2000:0000 至 2000:0003 的内容依次为：1E 06 00 0A)。此题要注意 loop 指令的使用规则，同时要注意区别[内存单元]与[内存单元中的数据(或内容)]的不同。

实验 8 分析一个奇怪的程序(第 174 页)

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
程序可以正常返回。
详细分析：
在此题中较为深入地考察了‘段内直接短转移’[形如：jmp short 标号]的概念。
我们知道程序中：
mov di,offset s
mov si,offset s2
mov ax,cs:[si]
mov cs:[di],ax
四条指令的作用是将标号 s2 处的一条指令复制到标号 s 处。这时我们应该关心所复制的语句"jmp short s1"对程序的影响：我们知道在段内直接短转移指令所对应的机器码中，并不包含转移的目的地址，而包含的是转移的位移量(如对此概念还不太熟悉，请查看书中第 167 页的内容)。也就是说，在源程序的编译过程中，编译器遇到‘段内直接短转移’[形如：jmp short 标号]时就会自动算出其要跳转的位移量，以便程序在执行‘段内直接短转移’的指令时就根据位移量进行(向前或向后)跳转。通过调试中的 U 命令我们可以看到指令's2:jmp short s1'所对应的机器码是 EBF6，F6h(-10d 的补码)就是跳转的位移量[此位移量也可由指令's2:jmp short s1'处的偏移地址 18h 减去指令's2:jmp short s1'后一个字节的偏移地址 22h 得出]。这时我们就知道了其实复制到标号 s 处的指令所对应的机器码就是 EBF6(刚好取代两个 nop 所对应的机器码)，它的作用就是将当前 IP 向前移动 10 个字节。当程序执行标号 s0 处的指令后，程序便跳到标号 s 处接着执行标号 s 处的指令。s 处的指令的作用是向前跳 10 字节，于是便跳到了代码中的第一条指令，继续执行后便实现了程序的正常返回。
[注意：此程序不会也不可能执行标号 s1 处后的指令。]
实验 9 根据材料编程(第 175 页)

```
assume cs:code,ds:data,ss:stack
data segment
        db 'welcome to masm!'    ;定义要显示的字符串(共 16 字节)
        db 02h,24h,71h       ;定义三种颜色属性
data ends
stack segment
        dw 8 dup(0)
stack ends
code segment
start:
        mov ax,data
        mov ds,ax
        mov ax,stack
        mov ss,ax
        mov sp,10h
        mov bx,0
        mov di,0
        mov ax,0b872h       ;算出屏幕第 12 行中间的显存的段起始位置放入 ax 中
        mov cx,3         ;外循环为 3 次，因为要显示三个字符串
s3:     push cx              ;三个进栈操作为外循环 s3 保存相关寄存器的值
        push ax              ;以防止它们的值在内循环中被破坏
```

```
        push di
        mov es,ax          ;此时 es 为屏幕第 12 行中间的显存的段起始位置
        mov si,0
        mov di,0
        mov cx,10h         ;内循环为 10h 次，因为一个字符串中含 10h 个字节
s1:     mov al,ds:[bx+si]
        mov es:[bx+di],al
        inc si
        add di,2
        loop s1            ;此循环实现偶地址中存放字符
        mov si,1     ;si 的值设为 1,从而为在显存奇地址中存放字符的颜色属性做准备
        pop di             ;将 di 的值恢复成进入内循环之前的时候的值
        mov al,ds:10h[bx+di]       ;取颜色属性[源 OP 寻址方式：相对基址变址]
        mov cx,10h         ;第二个内循环也为 10h 次
s2:     mov es:[bx+si],al
        add si,2
        loop s2            ;此循环实现奇地址中存放字符的颜色属性
        ;以下 4 句为下一趟外循环做准备
        inc di
        pop ax
        add ax,0ah     ;将显存的段起始地址设为当前行的下一行
      ;[在段地址中加 0ah,相当于在偏移地址中加了 0a0h(=160d)]
        pop cx
        loop s3
        mov ax,4c00h
        int 21h
code ends
end start
```

检测点 10.1(第 179 页)

第一空：1000h

第二空：0

提示：此题等效于把 CS 的值改为 1000H，把 IP 的值改为 0。因为 retf 指令进行的操作是先将 IP 出栈，再将 CS 出栈，所以在进栈时应当进行相反的操作。

检测点 10.2(第 181 页)

ax=6

提示：在执行指令"call s"时，IP 的值变为 6，接着进栈。此时程序直接执行指令"s:pop ax"，这就等于把栈中 IP 的值放入 ax 中。所以答案为 6。关于更多的 call 指令的问题请看附注中的“错误指出”中的第 6 条。

检测点 10.3(第 181 页)

ax=1010

提示：

1.寄存器中存放的值为 16 进制数

2.关于更多的 call 指令的问题请看附注中的“错误指出”中的第 6 条。

检测点 10.4(第 182 页)

ax=000B

提示：关于更多的 call 指令的问题请看附注中的“错误指出”中的第 6 条。

检测点 10.5(第 183 页)

(1)答：ax 中的数值为 3

提示：不能利用 T 命令进行调试，则改用 U 和 G 命令来调试。可用 U 命令先查看指令"mov ax,4c00h"处的偏移地址，然后用 G 命令直接执行到指令"mov ax,4c00h"的偏移地址处。

(2)

ax=1

bx=0

提示：关于更多的 call 指令的问题请看附注中的“错误指出”中的第 6 条。

实验 10 编写子程序(第 194 页)

1.显示子程序

完整程序如下：

```
data segment
        db 'Welcome to masm!',0
data ends
code segment
        assume cs:code,ds:data
start:
        mov dh,1                ;dh 装行号(范围:1--25)
        mov dl,1                ;dl 装列号(范围:1--80)[注:每超过 80 等于行号自动加 1]
        mov cl,0cah             ;cl 中存放颜色属性(0cah 为红底高亮闪烁绿色属性)
        mov ax,data
        mov ds,ax
        mov si,0
        call show_str
        mov ax,4c00h
        int 21h
show_str:   ;显示字符串的子程序[定义开始]
        push cx
        push si
        mov al,0A0h
        dec dh              ;行号在显存中下标从 0 开始,所以减 1
        mul dh
        mov bx,ax
        mov al,2
        mul dl
        sub ax,2            ;列号在显存中下标从 0 开始,又因为偶字节存放字符,所以减 2
        add bx,ax           ;此时 bx 中存放的是行与列号的偏移地址
        mov ax,0B800h
        mov es,ax           ;es 中存放的是显存的第 0 页(共 0--7 页)的起始的段地址
        mov di,0
        mov al,cl
        mov ch,0
```

```
s:          mov cl,ds:[si]
            jcxz ok
            mov es:[bx+di],cl           ;偶地址存放字符
            mov es:[bx+di+1],al         ;奇地址存放字符的颜色属性
            inc si
            add di,2
            jmp short s
ok:         pop si
            pop cx
            ret       ;显示字符串的子程序[定义结束]
code ends
end start
```

2.解决除法溢出的问题(第 197 页)

完整程序如下：

```
assume cs:code,ss:stack
stack segment
            dw 8 dup(0)
stack ends
code segment
start:
            mov ax,stack
            mov ss,ax
            mov sp,10h
            mov ax,4240h
            mov dx,0fh
            mov cx,0ah
            call divdw
            mov ax,4c00h
            int 21h
divdw:    ;子程序定义开始
            push ax
            mov ax,dx
            mov dx,0
            div cx
            mov bx,ax
            pop ax
            div cx
            mov cx,dx
            mov dx,bx
            ret     ;子程序定义结束

code ends
end start
```

3.数值显示(第 198 页)

完整程序如下：

```
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
        call dtoc
        mov dh,8
        mov dl,3
        mov cl,0cah
        call show_str
        mov ax,4c00h
        int 21h
dtoc:   ;数值显示的子程序定义
        push dx
        push cx
        push ax
        push si
        push bx
        mov bx,0
s1:     mov cx,10d
        mov dx,0
        div cx
        mov cx,ax
        jcxz s2
        add dx,30h
        push dx
        inc bx
        jmp short s1
s2:     add dx,30h
        push dx
        inc bx          ;再进行一次栈操作(补充当"商为零而余数不为零"时的情况)
        mov cx,bx
        mov si,0
s3:     pop ax
        mov [si],al
        inc si
        loop s3
okay:   pop bx
```

```
            pop si
            pop ax
            pop cx
            pop dx
            ret     ;数值显示的子程序定义结束
show_str:   ;显示字符串的子程序已经在第一题中说明，在此不再赘述。
            push bx
            push cx
            push si
            mov al,0A0h
            dec dh
            mul dh
            mov bx,ax
            mov al,2
            mul dl
            sub ax,2
            add bx,ax
            mov ax,0B800h
            mov es,ax
            mov di,0
            mov al,cl
            mov ch,0
s:          mov cl,ds:[si]
            jcxz ok
            mov es:[bx+di],cl
            mov es:[bx+di+1],al
            inc si
            add di,2
            jmp short s
ok:         pop si
            pop cx
            pop bx
            ret
code ends
end start
```

检测点 11.1(第 205 页)

ZF	PF	CF
1	1	0
1	1	0
1	1	0
1	1	0
0	0	0
0	1	0
0	1	0

检测点 11.2(第 208 页)

CF	OF	SF	ZF	PF
0	0	0	1	1
0	0	0	1	1
0	0	1	0	1
0	0	1	0	1
1	1	0	1	1
1	1	0	1	1
1	0	0	0	0
1	0	0	0	0
0	1	1	0	1
0	1	1	0	1

检测点 11.3(第 219 页)

(1)

第一空：jb s0

第二空：ja s0

(2)

第一空：jna s0

第二空：jnb s0

提示：注意区别闭区间[32,128]与开区间(32,128)所表示的数值范围。

检测点 11.4(第 223 页)

ax=45H

提示：

1.有符号数在计算机中是以其补码形式进行存储的。

2.注意执行指令"add ax,0010h"后影响的相关标志位的情况。详情如下：

CF	OF	SF	ZF	PF
1	0	0	1	1

实验 11 编写子程序(第 224 页)

完整程序如下：

```
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: ;子程序部分[开始]
        push si
s0:     mov al,[si]
```

```
            cmp al,0
            je exitsub
            cmp al,61h    ;61h 为'a'的 ASCII 码
            jb next
            cmp al,7ah    ;7ah 为'z'的 ASCII 码
            ja next
            and al,11011111B       ;或使用 sub al,20h
            mov [si],al
next:     inc si
            jmp short s0
exitsub:
            pop si
            ret    ;子程序部分[结束]
codesg ends
end begin
```

提示：

1.相当于将字符串中的每个字符在闭区间['a','z']内进行比较；

2.欲显示相关字符串，可调用实验 10 中的"显示子程序"(属第十章内容),在调用时注意相关参数的传递。