例5.1、试编制一个程序把BX寄存器内的二进制数用十六进制数的形式在屏幕上显示出来

分析问题：把BX寄存器中16位的二进制数用4位十六进制数的形式在屏幕上显示

1、初始设置

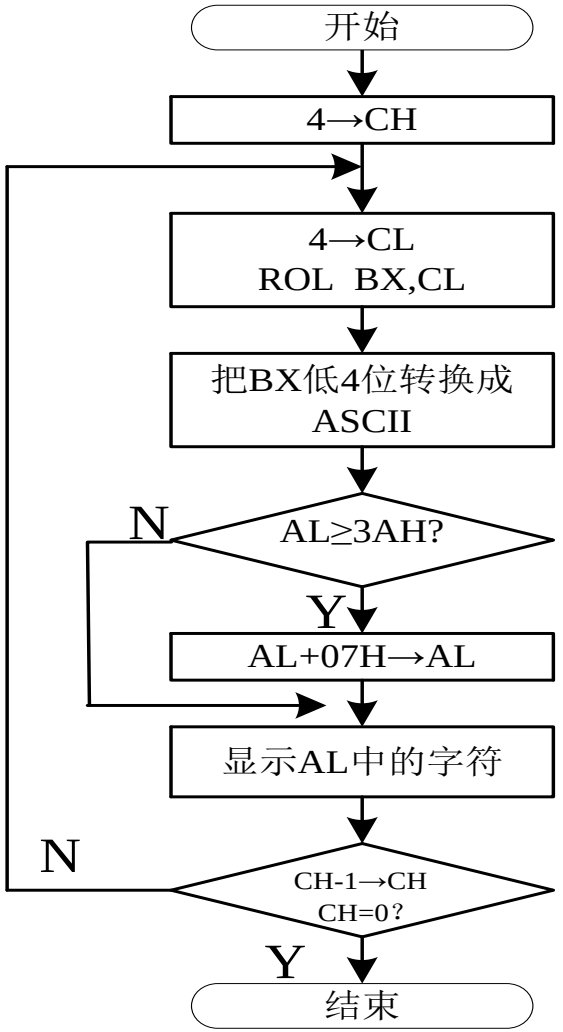
- 循环次数：每次显示1个十六进制数（送显示器相应的ASCII），循环次数=4，CH=4

2、循环体

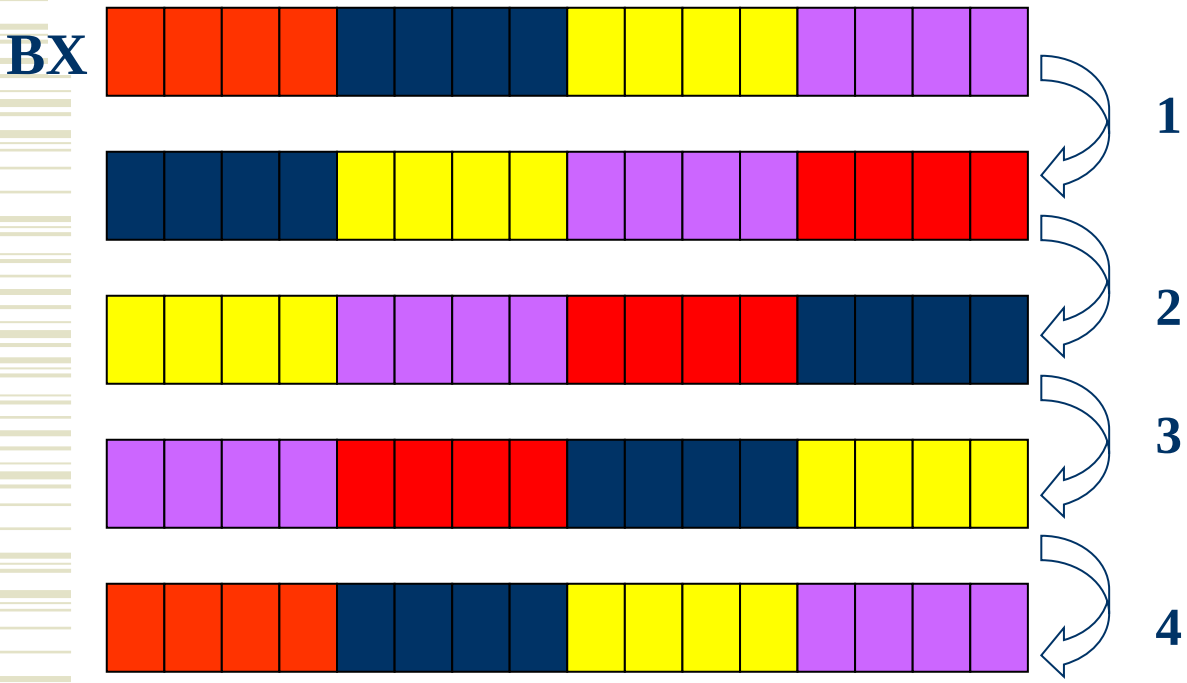
- 根据任务，选择算法
  - 每4位二进制数转换成1位十六进制数的ASCII
  - 调用DOS系统功能在屏幕上显示

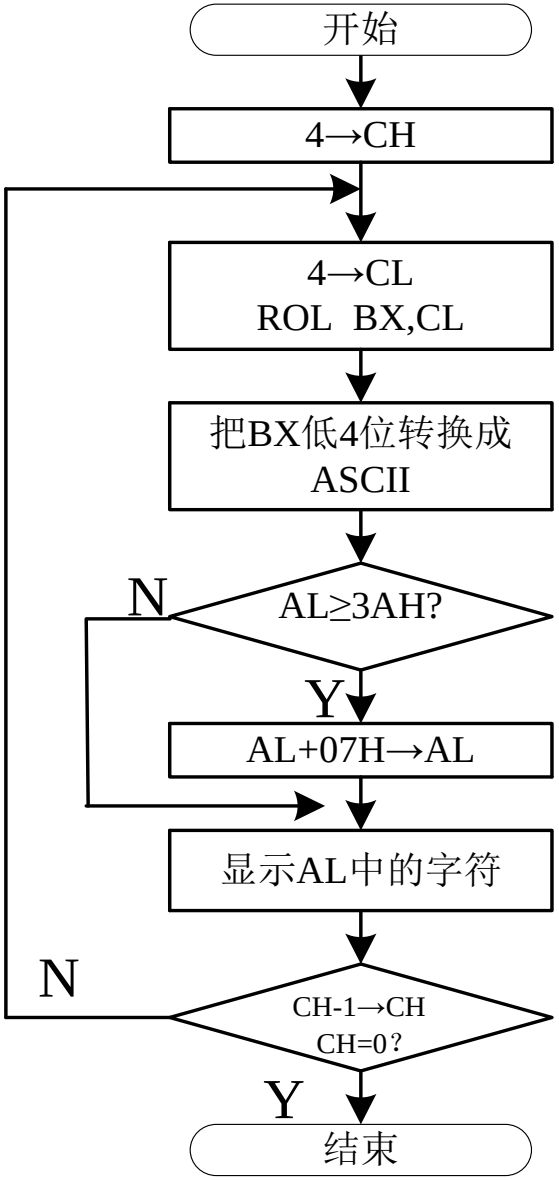
3、循环控制转移

- 根据计数控制循环次数

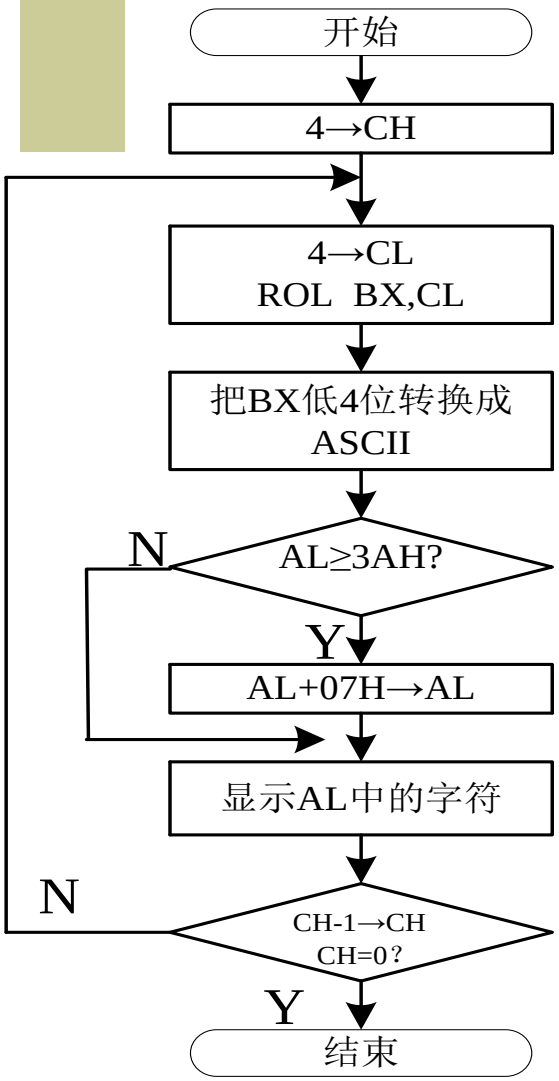


流程图


BX
1
2
3
4


开始
4→CH
4→CL
ROL BX,CL
把BX低4位转换成
ASCII
N
AL≥3AH?
Y
AL+07H→AL
显示AL中的字符
N
CH-1→CH
CH=0?
Y
结束

没有数据分配问题，直接编写程序代码段



流程图

```
prognam segment
main proc far
        Assume cs:prognam
start:
        push ds
        sub ax, ax
        push ax
        mov ch, 4
rotate: mov cl, 4
        …
printit:
        mov dl, al
        mov ah, 2
        int 21h
        dec ch
        jnz rotate
        ret
main    endp
prognam ends
        end start
```

初始化

循环体

循环控制转移

系统调用
显示1个字符

DOS环境下
返回DOS

熟记程序框架结构和系统调用约定

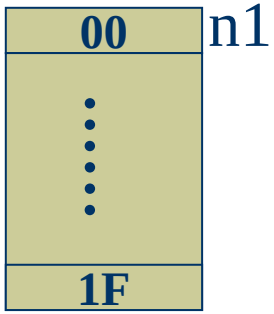
```
          mov   ch, 4
rotate:   mov   cl,  4
          rol   bx, cl
          mov   al,  bl
          and   al,  0fh          ;al的低4位0-9，A-F
          add   al,  30h          ;al的低4位30-39，3A-3F
          cmp   al,  3ah
          jl    printit                ;'0'-'9' ASCII 30H-39H
          add   al,  7h           ;'A'-'F' ASCII 41H-46H
printit:  mov   dl,  al
          mov   ah,  2
          int   21h
          dec   ch
          jnz   rotate
```

例5.1几点说明：

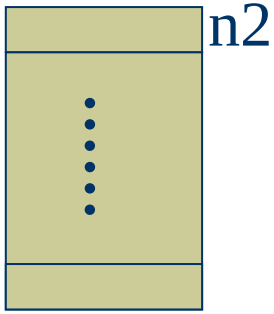
- 程序实现没有使用LOOP指令
  - 解决寄存器使用冲突
  - 用计数值控制循环结束，不是非得用LOOP指令
- 关于循环次数控制
  - 也可以把计数值初始化为0，每循环一次加1，然后比较判断
- 也可用LOOP指令
  - 通过堆栈保存信息解决CX冲突问题

例：编制一个数据块移动程序 (已知循环次数)

- 1) 任务1：用程序设置数据
  - 给内存数据段（DATA）中偏移地址为n1开始的连续32个字节单元，置入数据00H，01H，02H，¨ ¨ ¨，1FH



- 2) 任务2：移动数据
  - 将内存数据段（DATA）中偏移地址为n1的数据传送到偏移地址为n2开始的连续的内存单元中去



1）任务1：用程序设置数据

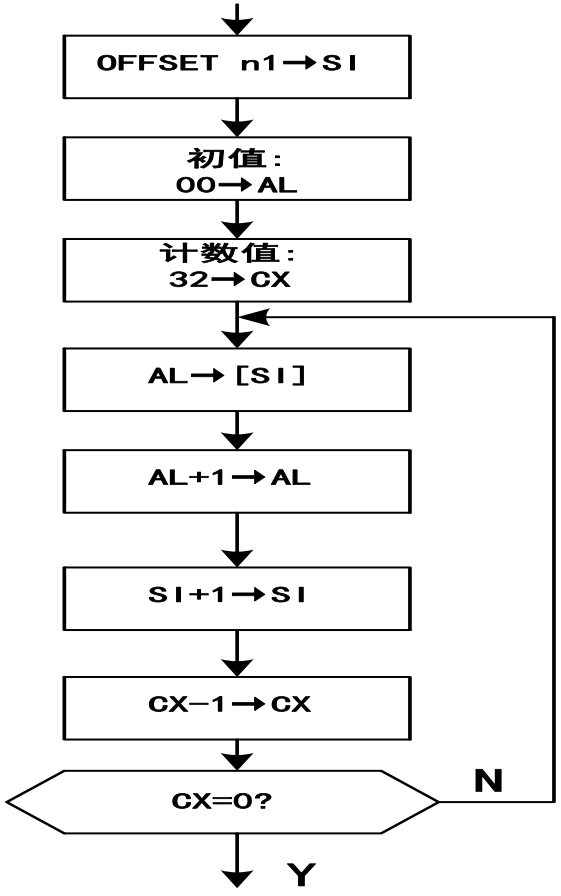
给内存数据段（DATA）中偏移地址为n1开始的连续32个字节单元，置入数据00H，01H，02H，……，1FH

- 对有规律（连续）的内存单元操作，地址有规律变化采用变址寻址方式
- 多次同样功能的处理，数据处理有规律，
- 采用循环程序结构

1、初始设置：
- 循环次数：连续32个字节单元，循环次数=32
- 数据初值：数据有规律变化，程序中可以自动生成数据，数据初值=00H
- 变址指针初始化：内存数据段中偏移地址为n1

2、循环体
- 根据任务，选择算法：一次设置一个字节
- 修改指针：变址指针修改
- 修改数据，生成新的数据
- 设置循环控制转移其他条件：无

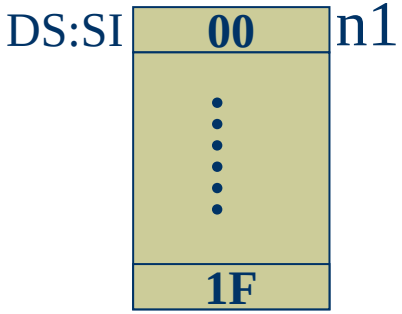
3、循环控制转移
- 根据计数控制循环次数

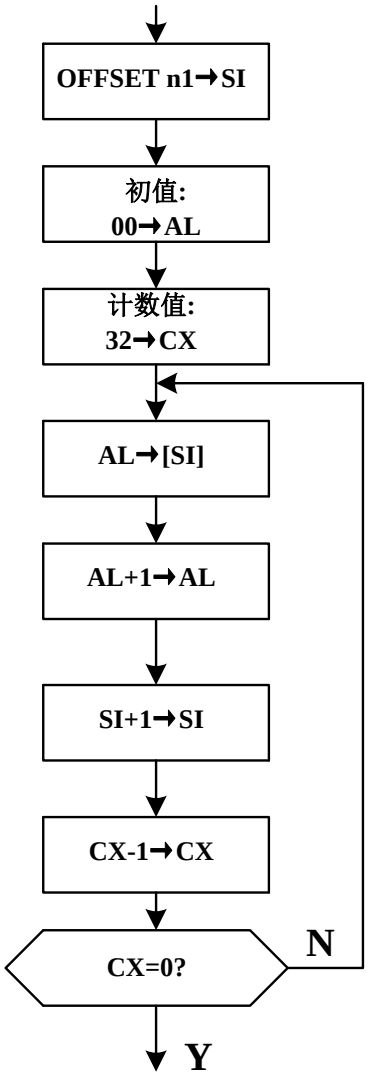


置入数据程序流程图

选择寄存器
进一步细化程序流程

- 循环的初始化部分完成
  - (1) 将n1的偏移地址置入变址寄存器SI，这里的变址寄存器作为地址指针用
  - (2) 待传送数据的起始值00H 送 AL
  - (3) 循环计数值32（或20H）送计数寄存器CX
- 循环体中完成
  - (4) AL中内容送地址指针所指存储器单元
  - (5) AL加1送AL；依次得到01H，02H，03H，…，1FH
  - (6) 地址指针SI加1，指向下一个地址单元
- 循环结束判断
  - (7) 计数器CX-1→CX
  - (8) 若CX≠0继续循环；否则结束循环
- 程序源代码请自己编写



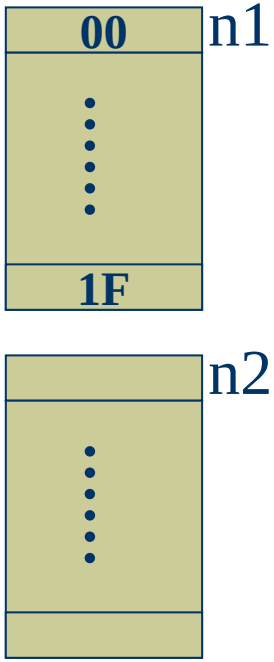


置入数据程序流程图

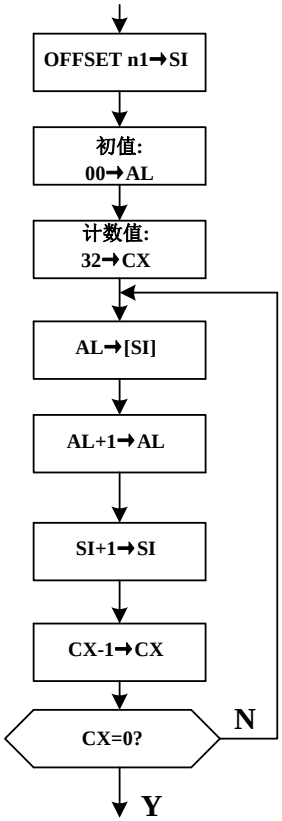
2)任务2：移动数据

将内存数据段（DATA）中偏移地址为n1的数据传送到偏移地址为n2开始的连续的内存单元中去

- 程序结构：这个问题是数据块移动问题，可选择循环结构或串传送指令来处理
  - 选择串传送指令MOVSB，或REP MOVSB
- 数据定义：要定义源串和目的串
  - 源串是任务（1）中所设置的数据
  - 目的串要保留相应长度的空间
- 处理方法：MOVSB指令是字节传送指令，它要求事先设置约定寄存器：
  - ① 将源串的首偏移地址送SI，段地址为DS
  - ② 目的串的首偏移地址送DI，段地址为ES
  - ③ 串长度送CX寄存器中
  - ④ 并设置方向标志DF

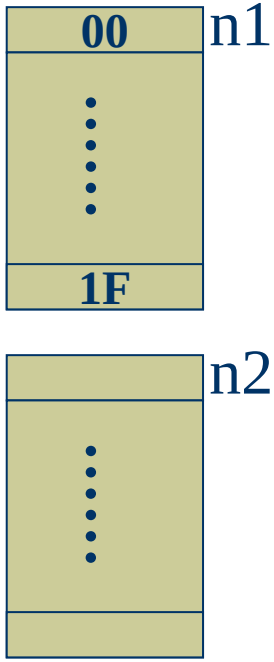


程序源代码请自己编写



置入数据程序流程图

```
data1     segment
n1        db 32 dup (?)
data1     ends

data2     segment
n2        db 32 dup (?)
data2     ends

prognam   segment
main      proc  far
          assume cs:prognam,
                 ds:data1, es:data2
start:    push  ds
          sub  ax, ax
          push  ax

          mov ax,  data1
          mov ds,  ax
          mov ax,  data2
          mov es,  ax

          mov  si, offset n1
          mov al,  0
          mov cx, 32
rotate:   mov [si],  al
          inc   si
          inc   al
          dec   cx
          jnz  rotate

          mov si,  offset n1
          mov di,  offset n2
          cld
          mov cx,  32
          rep  movsb

          ret

main      endp
prognam   ends
          end  start
```



例子5.2 数“1”的个数

在addr单元中存放着数 Y 的地址，把 Y 中1的个数存入 COUNT 单元中

参看p178

（1）数“1”的方法

a） 移位到进位，测试进位；测试符号位

b） 判最高位是否为1

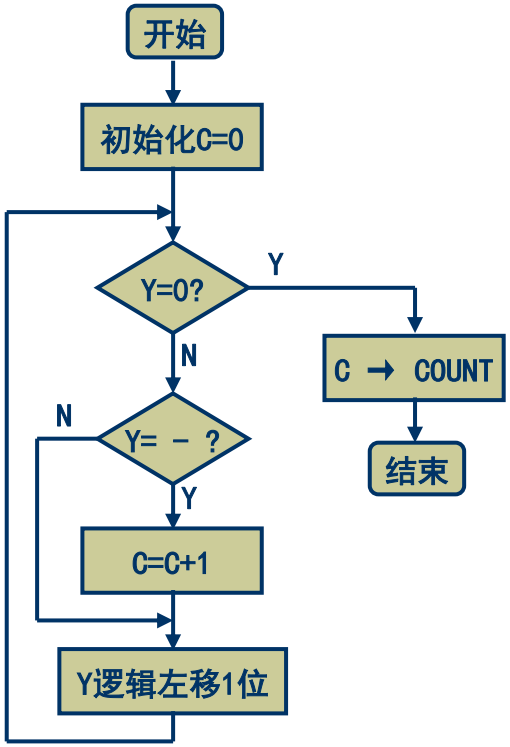
（2）循环控制条件

a） 简单思路：计数值以16控制

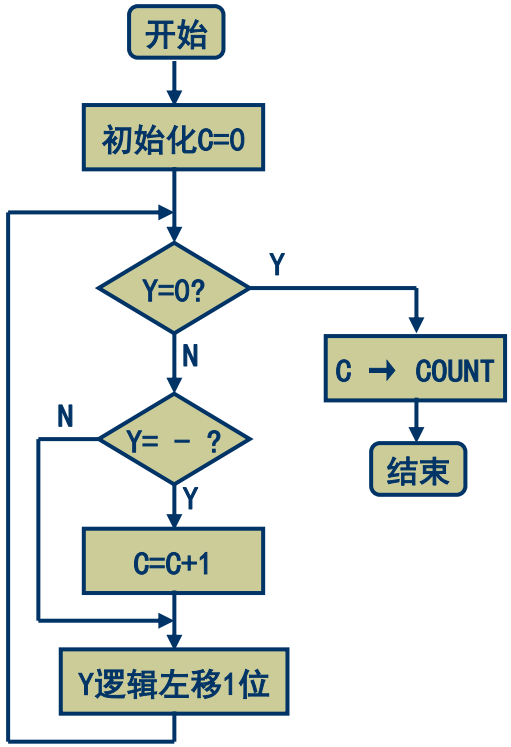
b） 比较好的方法：简单思路结合测试数是否为0

（3）DO_WHILE 结构

Y本身为0的可能性



```
datarea segment
        addr        dw  number
        number      dw  y
        count       dw  ?
datarea segment
prognam             segment
mian    proc        far
        assume      cs:prognam, ds:datarea
start:  push        ds
        sub         ax, ax
        push        ax
        mov         ax, datarea
        mov         ds, ax
        mov         cx, 0
        mov         bx, addr
        mov         ax, [bx]
repeat: test        ax, 0ffffh
        jz          exit
        jns         shift
        inc         cx
shift:  shl         ax, 1
        jmp         repeat
exit:   mov         count, cx
        ret
mian:   endp
prognam ends
        end         start
```



例子5.3 删除在未经排序的数组中找到的数（待找的数存放在AX中）P179

分析题意：

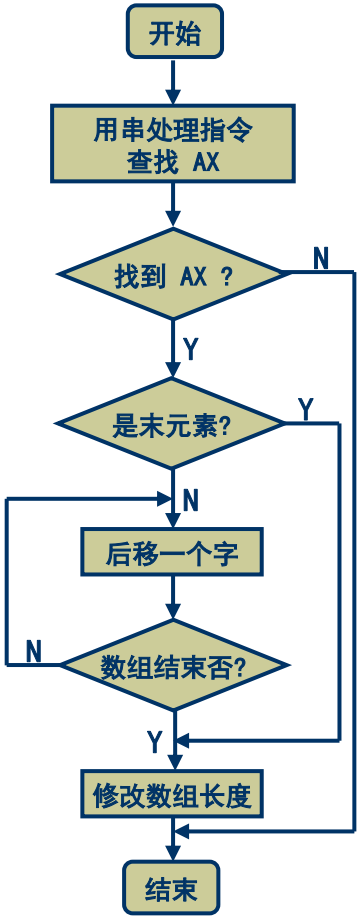
（1）如果没有找到，则不对数组作任何处理

（2）如果找到这一元素，则应把数组中位于高地址中的元素向低地址移动一个字，并修改数组长度值

（3）如果找到的元素正好位于数组末尾，只要修改数组长度值

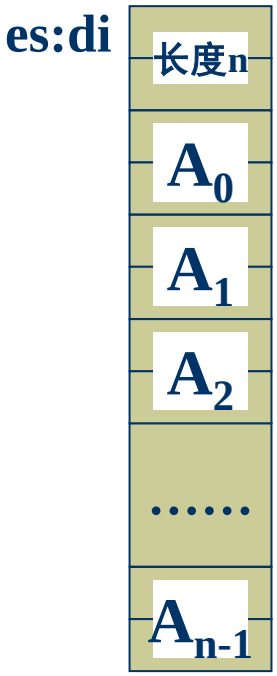
（4）查找元素用串处理指令（repnz　scasw）

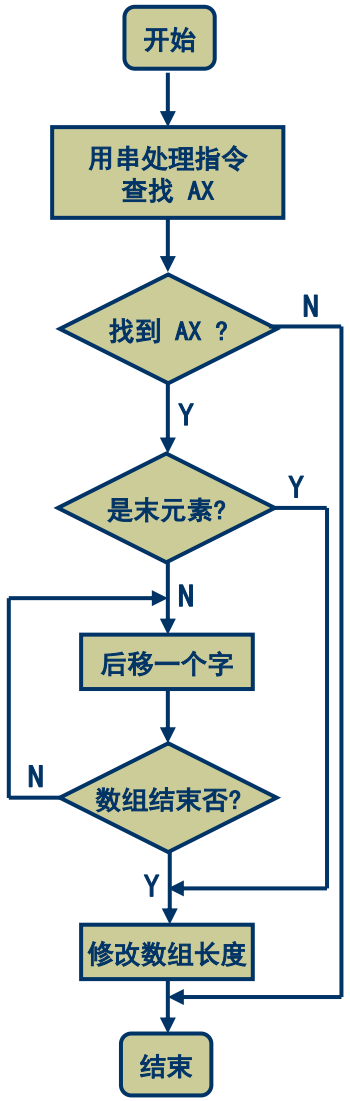
（5）删除元素用循环结构



子程序源代码

```
del_ul    proc     near
          cld
          push     di
          mov      cx, es:[di]
          add      di, 2
          repne    scasw
          je       delete
          pop      di
          jmp      short  exit
delete:   jcxz     dec_cnt     ;  如果CX=0, 转移
next_el:  mov      bx, es:[di]
          mov      es:[di-2], bx
          add      di, 2
          loop     next_el
dec_cnt:  pop      di
          dec      word ptr es:[di]
exit:     ret
del_ul    endp
```





执行了几次pop操作?

SCAS指令执行的操作：

1. DST-AL/AX/EAX，但结果不保存，根据结果设置标志位
2. 目标操作数的地址指针（变址寄存器DI）的修改

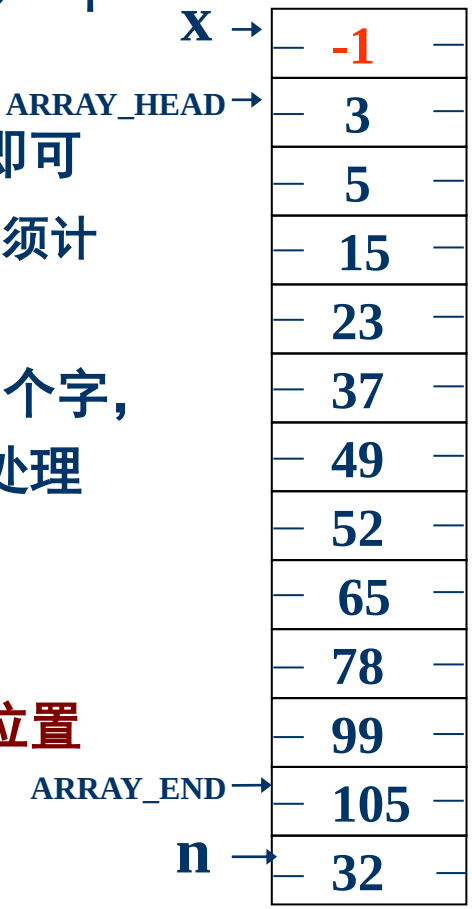
REPNE执行的操作：

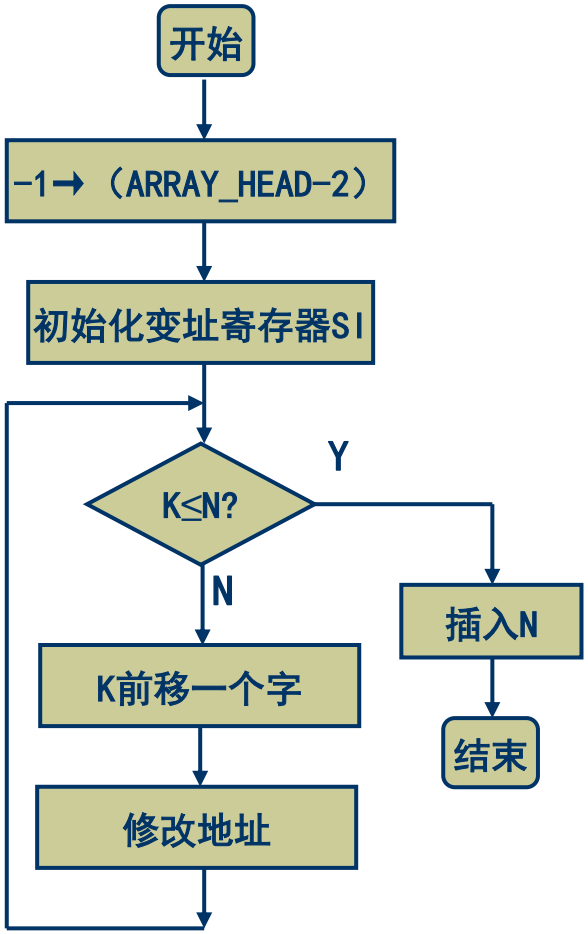
1. 若 CX＝0（计数到)或ZF＝1(相等),则结束重复
2. 否则,修改计数器 CX－1→CX, 执行后跟的串操作指令。转1，继续重复上述操作

例子5.4　在已整序的正数字数组中插入正数N

找到位置，将数据向高地址移一个字，插入N, 结束

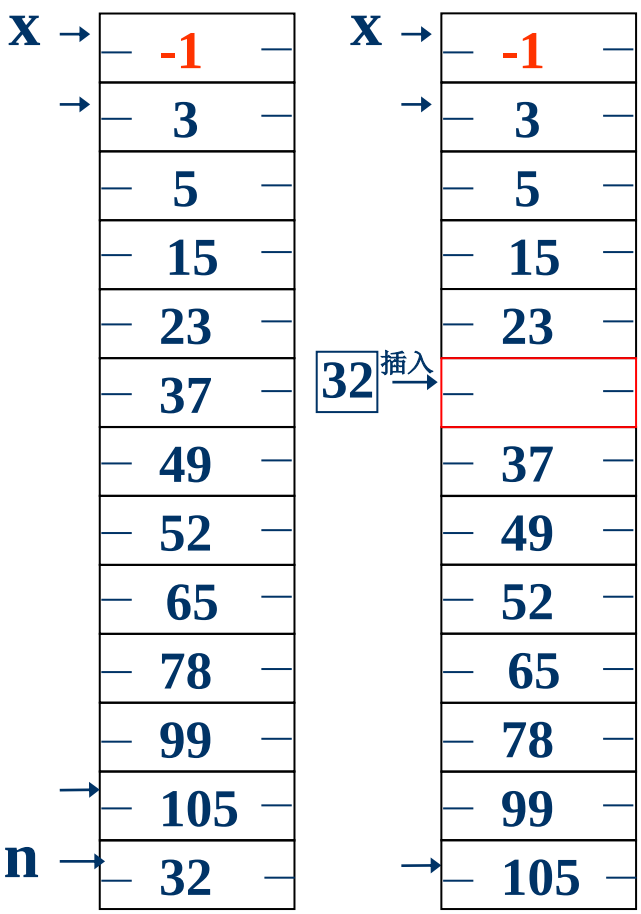
- 循环控制条件：找到位置即可
  - 位置一定能找到，因此无须计数次数等
- 将高于N的数向高地址移一个字，边找边移，因此循环体内处理为向高地址移一个字
- 插入N在循环结构外
- 循环结构的主要任务是找位置和移字数据





```
x           dw    ?
array_head  dw    3,5,15,23,37,49,52,65,78,99
array_end   dw    105
n           dw    32
```

```
        mov   ax, n
        mov   array_head-2, 0ffffh
        mov   si, 0
compare:
        cmp   array_end[si], ax
        jle   insert
        mov   bx, array_end[si]
        mov   array_end[si+2], bx
        sub   si, 2
        jmp   short compare
insert:
        mov   array_end[si+2], ax
```



例子5.5

- 设数组X、Y中分别存有10个字型数据
  试实现以下计算并把结果存入数组Z单元

Z0＝X0+Y0　　Z1＝X1+Y1
Z2＝X2-Y2　　Z3＝X3-Y3
Z4＝X4-Y4　　Z5＝X5+Y5
Z6＝X6-Y6　　Z7＝X7-Y7
Z8＝X8+Y8　　Z9＝X9+Y9

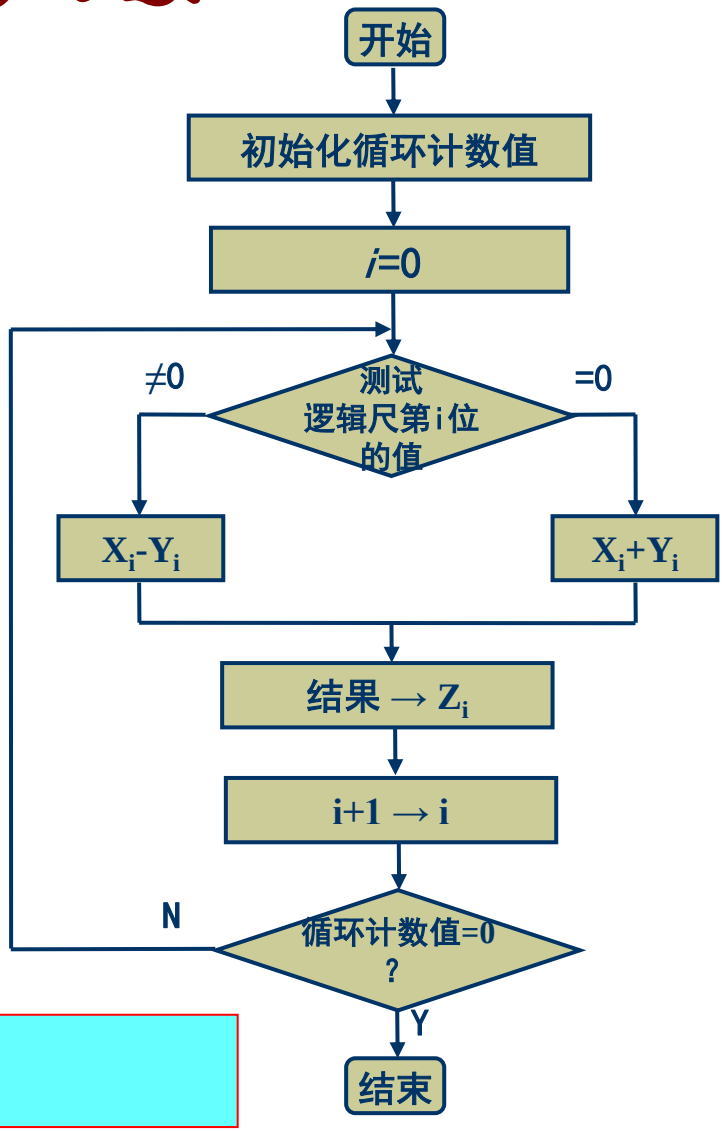
逻辑尺方法

（1）设立标志位

0000000011011100

$Z_2=X_2-Y_2$　　$Z_0=X_0+Y_0$

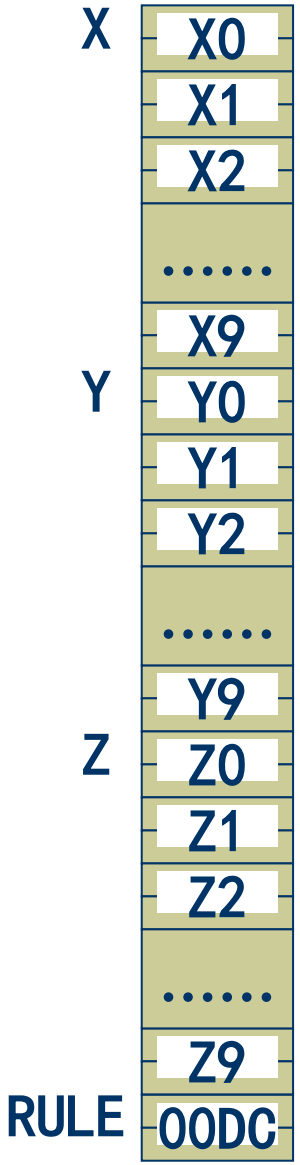
（2）进入循环后判断标志位来确定该做的工作



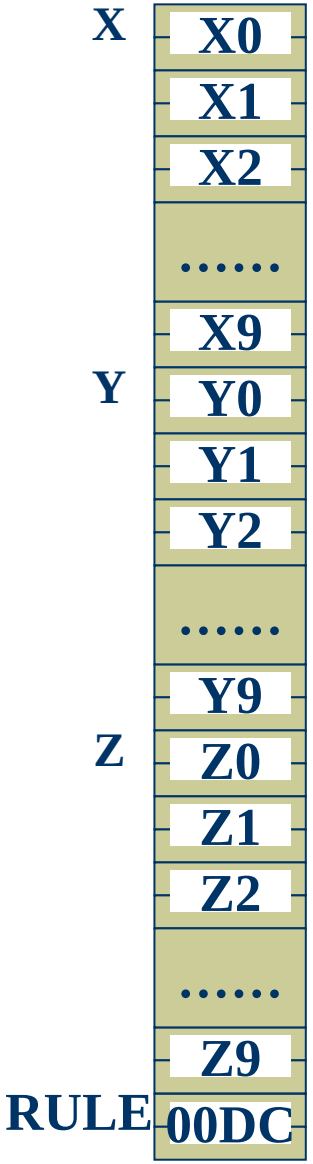
建立逻辑尺：++−−+−−−++

0000000011011100

```
DATA    SEGMENT
   X    DW  X0,X1,X2,X3,X4
        DW  X5,X6,X7,X8,X9
   Y    DW  Y0,Y1,Y2,Y3,Y4
        DW  Y5,Y6,Y7,Y8,Y9
   Z    DW  10 DUP(?)
   RULE DW  0000000011011100B;逻辑尺
DATA    ENDS
```



```
CODE   SEGMENT
       ASSUME CS:CODE,DS:DATA
MAIN   PROC   FAR
       MOV    AX,DATA
       MOV    DS,AX
       MOV    CX,10          ;循环次数
       MOV    DX,RULE        ;逻辑尺
       MOV    BX,0           ;地址指针
NEXT:  MOV    AX,X[BX]       ;取X中的一个数
       SHR    DX,1           ;逻辑尺右移一位
       JC     SUBS           ;分支判断并实现转移
       ADD    AX,Y[BX]       ;两数加
       JMP    SHORT RESULT
SUBS:  SUB    AX,Y[BX]       ;两数减
RESULT:MOV    Z[BX],AX       ;存结果
       ADD    BX,2           ;修改地址指针
       LOOP   NEXT
       MOV    AX,4C00H       } 返回DOS
       INT    21H            }
MAIN   ENDP
CODE   ENDS
       END    MAIN
```



- 返回DOS操作系统有两种方法：

通用方法：

```
start:
        push  ds
        sub  ax, ax
        push  ax
        ……
        ret
main  endp
```

高级DOS版本可用方法：

```
start:
        ……
        mov  ax,  4c00h
        int  21h
main  endp
```