



哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

汇编语言程序设计

第6讲：循环与分枝程序设计

裴文杰

控制转移指令

循环结构程序设计

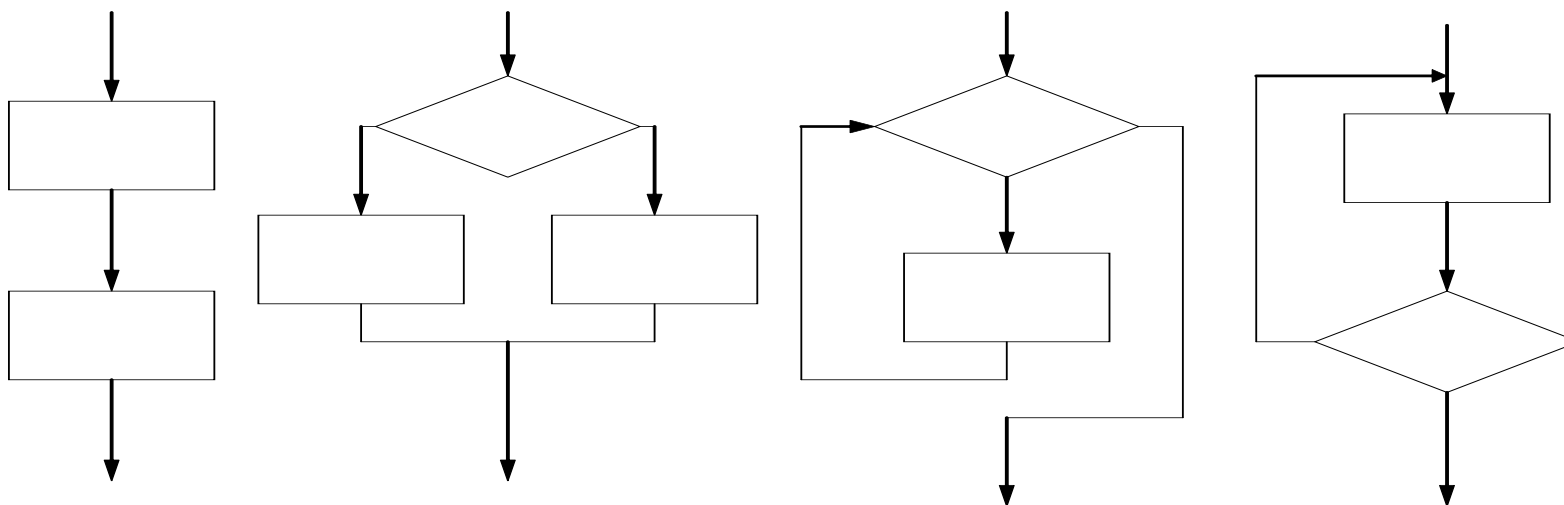
分支结构程序设计

控制转移指令

循环结构设计

分支结构设计

三种程序构件



控制转移指令：

❑ 无条件转移指令

JMP

❑ 条件转移指令

JZ / JNZ 、 JE / JNE、 JS / JNS、 JO / JNO、

JP / JNP、 JB / JNB、 JL / JNL、 JBE / JNBE、

JLE / JNLE、 JCXZ

❑ 循环指令

LOOP、 LOOPZ / LOOPE、 LOOPNZ / LOOPNE

无条件转移指令:

段内直接短转移: `JMP SHORT OPR`

执行操作: $(IP) \leftarrow (IP) + 8\text{位位移量}$

段内直接近转移: `JMP NEAR PTR OPR`

执行操作: $(IP) \leftarrow (IP) + 16\text{位位移量}$

段内间接转移: `JMP WORD PTR OPR`

执行操作: $(IP) \leftarrow (EA)$

段间直接远转移: `JMP FAR PTR OPR`

执行操作: $(IP) \leftarrow \text{OPR 的段内偏移地址}$
 $(CS) \leftarrow \text{OPR 所在段的段地址}$

段间间接转移: `JMP DWORD PTR OPR`

执行操作: $(IP) \leftarrow (EA)$
 $(CS) \leftarrow (EA+2)$

控制转移指令

条件转移指令：只能使用段内直接寻址的8 位位移量（386以后机型支持16位位移量）

(1) 根据单个条件标

操作数为目标符号地址

短转移：-128 ~ +127

条件转移指令不提供；段间远转移格式，如有需要可转换为JMP指令

指令的助忆符	检测的转移条件	功能描述
JZ/JE	ZF=1	结果为0或相等则转移
JNZ/JNE	ZF=0	结果不为0或不相等则转移
JS	SF=1	结果为负则转移
JNS	SF=0	结果不为负则转移
JO	OF=1	结果溢出则转移
JNO	OF=0	结果不溢出则转移
JP/JPE	PF=1	奇偶位为1则转移(1的个数为偶数时为1)
JNP/JPO	PF=0	奇偶位为0则转移
JB/JNAE/JC	CF=1	低于或(不高于或等于)，或CF=1则转移
JNB/JAE/JNC	CF=0	不低于或(高于或等于)，或CF=0则转移

比较两个无符号数，并根据比较结果转移*

两数的高低分成4种关系：

- (1) <, 低于（不高于等于）：JB (JNAE) , CF=1
- (2) ≥, 不低于（高于等于）：JNB (JAE) , CF=0
- (3) ≤, 低于等于（不高于）：JBE (JNA) , CF ∨ ZF=1
- (4) >, 不低于等于（高于）：JNBE (JA) , CF ∨ ZF=0

* 适用于地址或双精度数低位字的比较

(3) 测试 CX 的值为 0 则转移

格式	测试条件
JCXZ OPR	(CX)=0

(4) 比较两个带符号数，并根据比较结果转移*

	格式	测试条件
<	JL (JNGE) OPR	SF \neq OF = 1 (异或)
\geq	JNL (JGE) OPR	SF \neq OF = 0
\leq	JLE (JNG) OPR	(SF \neq OF) \vee ZF = 1
>	JNLE (JG) OPR	(SF \neq OF) \vee ZF = 0

* 适用于带符号数的比较

< JL (JNGE) OPR SF \square OF = 1

带符号数比较结果的几种情况：

两个异号数相加或同号数相减，结果不会溢出。

两个同号数相加或异号数相减，有可能溢出。

正溢出：结果大于机器能表示的最大正数

负溢出：结果小于机器能表示的最小负数

例：比较两个数，若A<B，则转到label去执行。

```
MOV AX, A
CMP AX, B
JL label
```

若A-B的结果使得SF=0，OF=0，说明差值为正，且未溢出，可以判断A ≥ B，SF \square OF = 0，不满足转移条件。（例：比较20, 8）

若A-

B的结果使得SF=0，OF=1，说明差值为正，且溢出，这种情况必为负溢出，可以判断A<B，SF \square OF = 1，满足转移条件。

若A-B的结果使得SF=1，OF=0，说明差值为负，且未溢出，可以判断A<B，SF \square OF = 1，满足转移条件。（例：比较8, 20）

若A-

B的结果使得SF=1，OF=1，说明差值为负，且溢出，这种情况必为正溢出，可以判断A>B，SF \square OF = 0，不满足转移条件。

< JL (JNGE) OPR SF \oplus OF = 1

正溢出：结果大于机器能表示的最大正数

负溢出：结果小于机器能表示的最小负数

若A-B的结果使得SF=0, OF=1, 说明差值为正, 且溢出, 这种情况必为**负溢出**, 可以判断A<B, SF \oplus OF = 1, 满足转移条件。

若A-B的结果使得SF=1, OF=1, 说明差值为负, 且溢出, 这种情况必为**正溢出**, 可以判断A>B, SF \oplus OF = 0, 不满足转移条件。

正溢出: 89-(-108)

01011001 (89)

01101100 (108)

11000101 (-59)

负溢出: -110-92

10010010 (-110)

10100100 (-92)

1 00110110 (54)

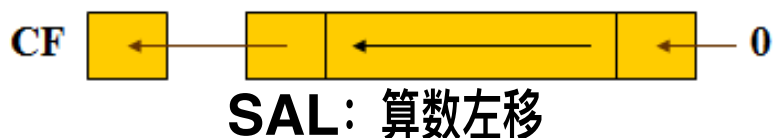
SF=1, OF=1

A>B, 不满足转移条件

SF=0, OF=1

A<B, 满足转移条件

例：统计AX寄存器中为1位数的，并将统计结果放在CL寄存器中。



```
MOV CL, 0          ; 置循环初值
MOV BX, 16
LAB1: SAL AX, 1     ; 将AX的内容左移一位，即最高位移到CF
      JNC LAB2      ; 如果CF=0则表示AX的最高位为0，转LAB2
      INC CL        ; 如果CF=1则表示AX的最高位为1，个数加1
LAB2: DEC BX        ; 修改循环次数，未完则转LAB
      JNZ LAB1
EXIT:...
```

```
MOV CL, 0
LAB: AND AX, AX     ; AX=0时循环结束，转到EXIT
      JZ EXIT
      SAL AX, 1     ; 将AX中的最高位移入CF中
      JNC LAB       ; 如果CF=0则转LAB
      INC CL        ; 如果CF=1则CL+1→CL
      JMP LAB       ; 转LAB处继续循环
EXIT:...
```

例：[?]、[?] 是带符号双精度数，分别存于DX, AX及BX, CX 中，[?] > [?] 时转 L1，否则转 L2。

```
CMP DX, BX
```

```
JG L1
```

```
JL L2
```

```
CMP AX, CX
```

```
JA L1
```

```
L2:
```

```
.....
```

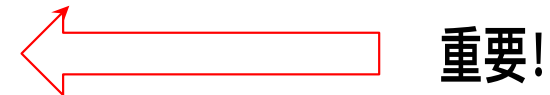
```
JMP L3
```

```
L1:
```

```
.....
```

```
L3:
```

```
.....
```



循环指令 LOOP

操作数为目标符号地址

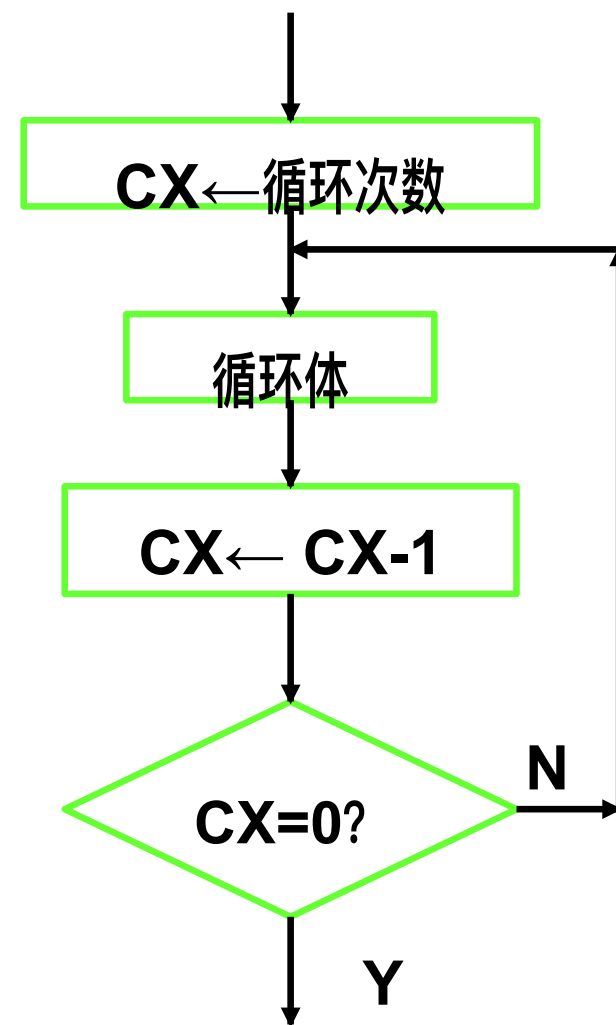
短转移: -128 ~ +127

语句格式: **LOOP** 短标号

执行过程:

1. $(CX) = (CX) - 1$ (不改变任何标志位)
- 2.

如果 $(CX) \neq 0$, 转向“标号”所指向的指令; 否则, 终止循环, 执行该指令下面的指令。



例：求首地址为 ARRAY 的 M
个字之和（不考虑溢出），结果存入 TOTAL 中。

```
MOV  CX, M           ;循环次数
MOV  AX, 0
MOV  SI, 0
```

AGAIN:

```
ADD  AX, ARRAY[SI]
ADD  SI, 2
LOOP AGAIN
MOV  TOTAL, AX
```

相等/为零循环指令 LOOPE/LOOPZ

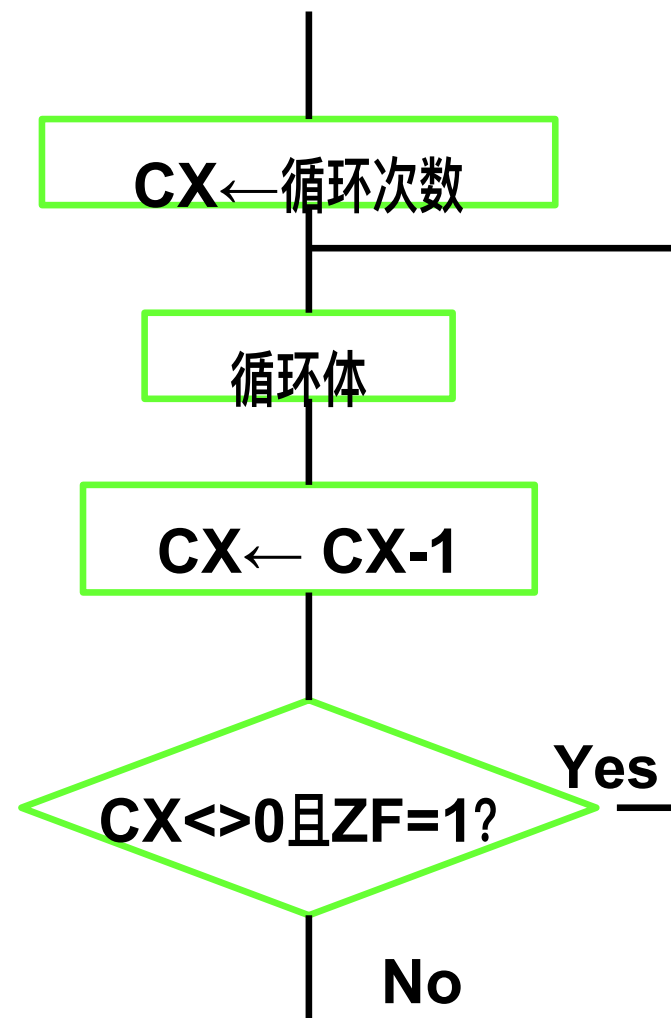
语句格式: **LOOPE/ LOOPZ** 短标号

执行过程:

1. $(CX) = (CX) - 1$ (不改变任何标志位)

2.

如果 $CX \neq 0$ 且 $ZF = 1$, 则程序转到循环体的第一条指令; 否则, 程序将执行该循环指令下面的指令。



不相等/不为零循环指令LOOPNE/LOOPNZ

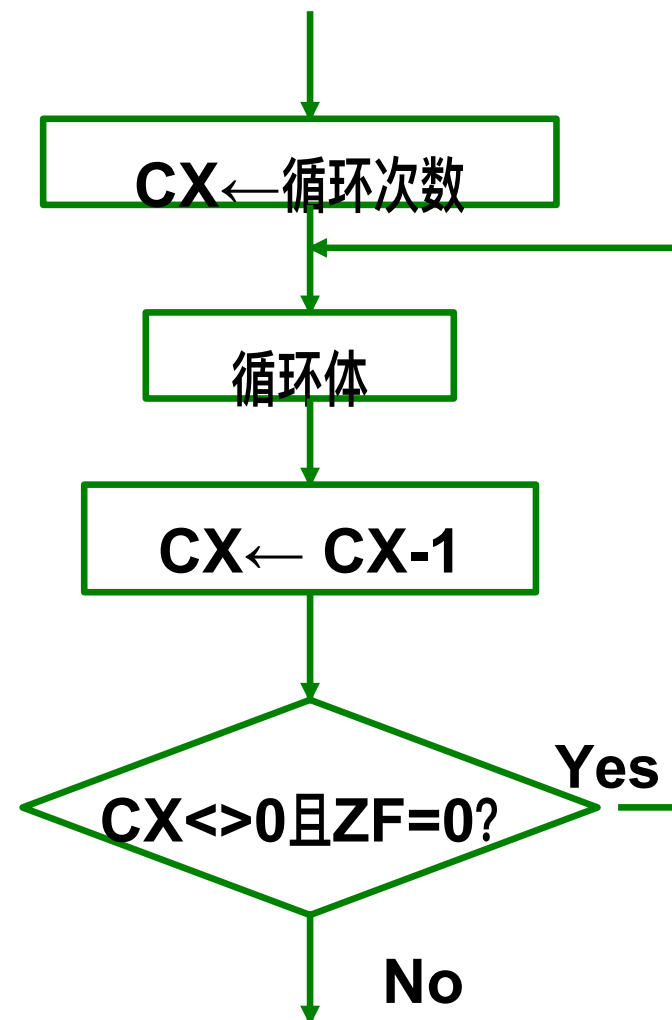
语句格式: LOOPNE/LOOPNZ 短标号

执行过程:

1. $(CX) = (CX) - 1$ (不改变任何标志位)

2.

如果 $CX \neq 0$ 且 $ZF = 0$, 则程序转到循环体的第一条指令; 否则, 程序将执行该循环指令下面的指令。



例：记录附加段一个长度为count的字符串中的空格个数到 RESULT单元。

MOV CX, COUNT ; 设置循环次数

MOV SI, OFFSET STRING

XOR BX, BX ; BX清0, 用于记录空格数

MOV AL, 20H ; 空格的ASC码为20H

AGAIN: CMP AL, ES:[SI]

立即数不能做目的操作数

JNZ NEXT ; ZF=0, 非空格, 转移

INC BX ; ZF=1, 是空格, 个数加1

NEXT: INC SI

LOOP AGAIN ; 字符个数减1, 不为0继续循环

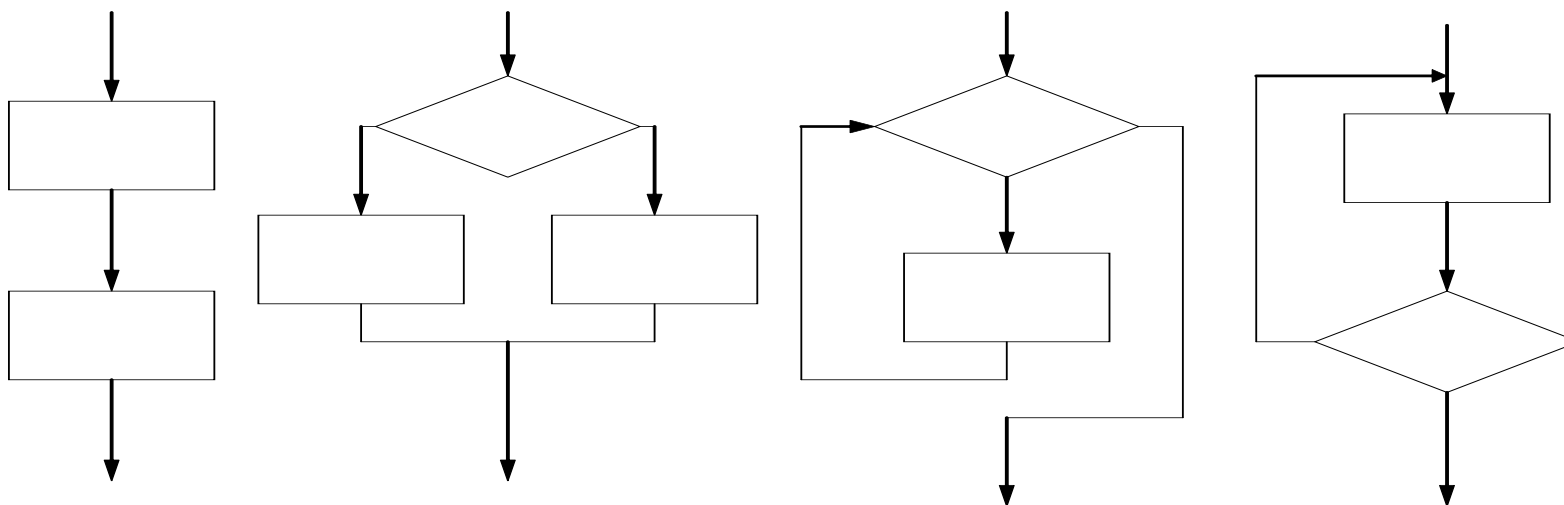
MOV RESULT, BX ; 保存结果

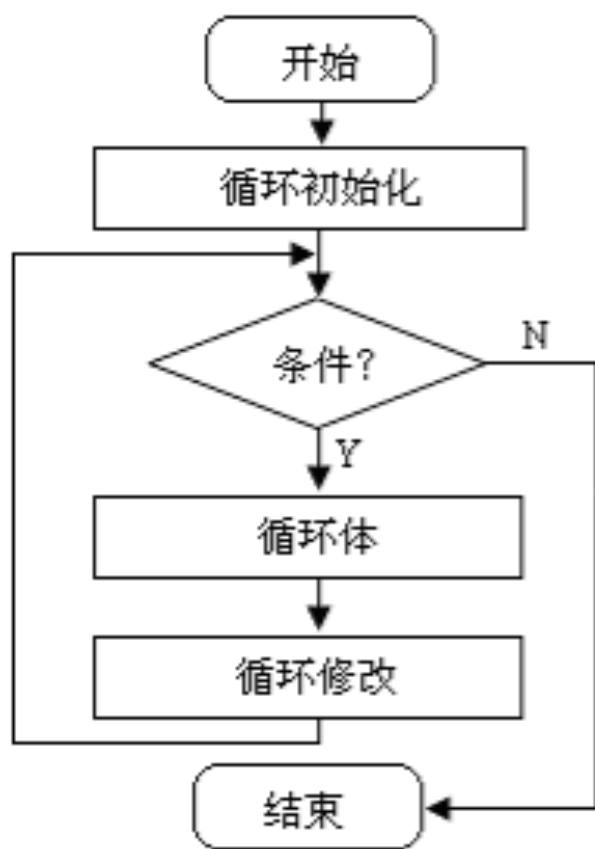
控制转移指令

循环结构设计

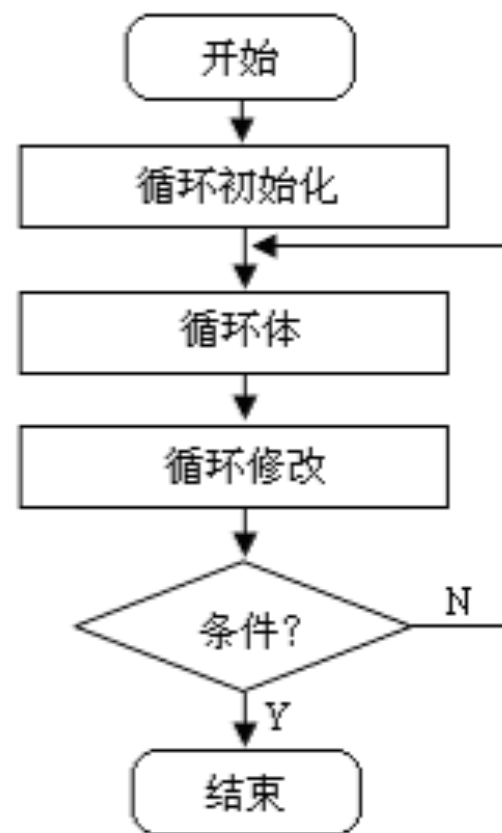
分支结构设计

三种程序构件





DO-WHILE 结构



DO-UNTIL 结构

基本循环结构示意图

循环结构程序设计

循环初始化部分

这是循环准备工作阶段，如建立地址指针、设置循环次数、必要的保护以及为循环体正常工作而建立的初始状态等。

循环体

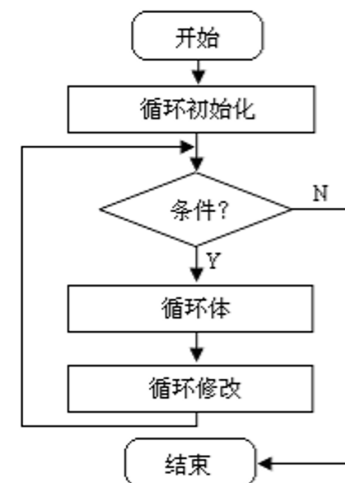
循环体是在循环过程中反复执行的部分。它是循环的核心部分，是循环程序所要完成的若干操作的全部指令。

循环修改部分

循环修改主要是指对一些运算控制单元（变量、寄存器）的修改，如修改操作数地址、修改循环计数器、改变变量的值等。

循环控制部分

根据给定的循环次数或循环条件，判断是否结束循环。若未结束，则转去重复执行循环工作部分。



DO-WHILE 结构

循环结构程序设计

23/64

重点：循环控制部分。

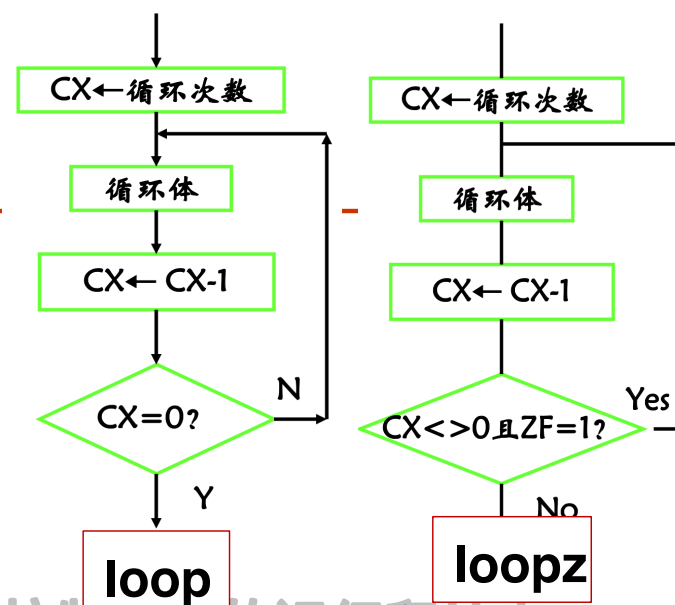
◆ 循环控制是循环体的一部分，它是循环程序设计的关键。

◆ 每个循环程序必须选择一个循环控制条件来控制循环的运行和结束。

有时循环次数已知，此时可以用循环次数作为控制条件，loop指令使得这种循环程序设计很容易实现。

有时循环次数已知，但有可能使用其他特征或条件使循环提前结束，此时可用loopz或loopnz指令。

有时循环次数未知，那就需要根据具体情况找出控制循环结束的条件，采用转移指令来实现循环。



程序设计基本步骤

一般说来，编制一个汇编语言程序需要完成以下步骤：

(1)

分析题意，建立数学模型，确定数据结构及算法。这一步是能否编制出高质量程序的关键，因此不应该一拿到题目就急于写程序，而是应该仔细地分析和理解题意，找出合理的算法及适当的数据结构。

(2)

根据算法画出程序流程图。这一步对初学者尤其重要，这样做可以减少出错的可能性。画流程图时可以从粗到细把算法逐步地具体化。

(3) 编写汇编语言源程序，根据算法及数据结构分配内存单元和寄存器。

(4) 使用汇编程序调试工具上机调试程序。

【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。

如: 1011 0010 1111 1010 B **0B2FAH**

分析题目：

(1)程序结构的确定

由题意应该把BX的内容从左到右每4位为一组在屏幕上显示出来，显然这可以用循环结构来完成，每次显示一个十六进制数位，因而循环次数是已知的，计数值为4。

(2)循环体的构成（算法确定）

循环体应该包括：
二进制到所显示字符的ASCII之间的转换，以及每个字符的显示。

(3)需要了解相关知识

◆字符和其ASCII码之间的关系？

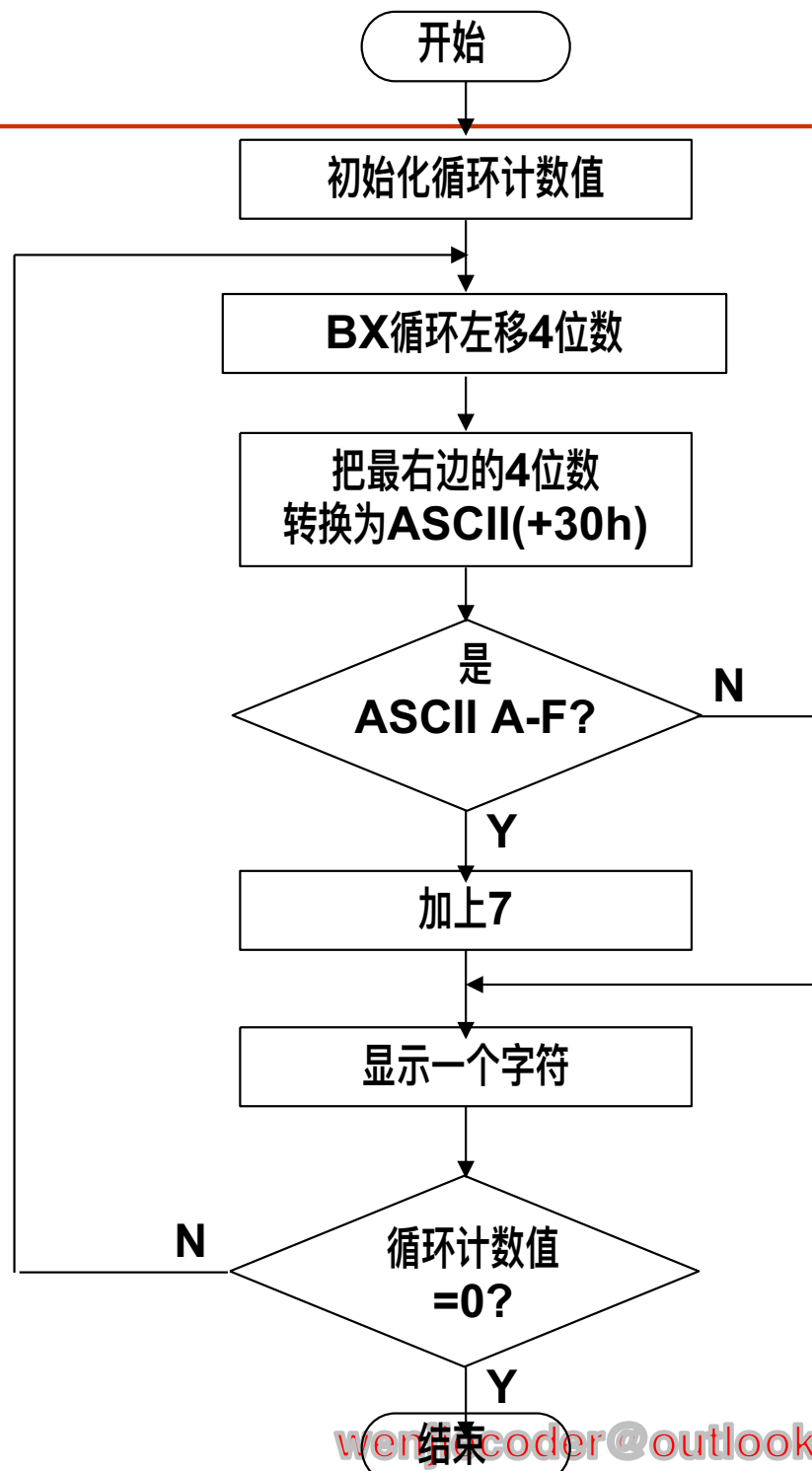
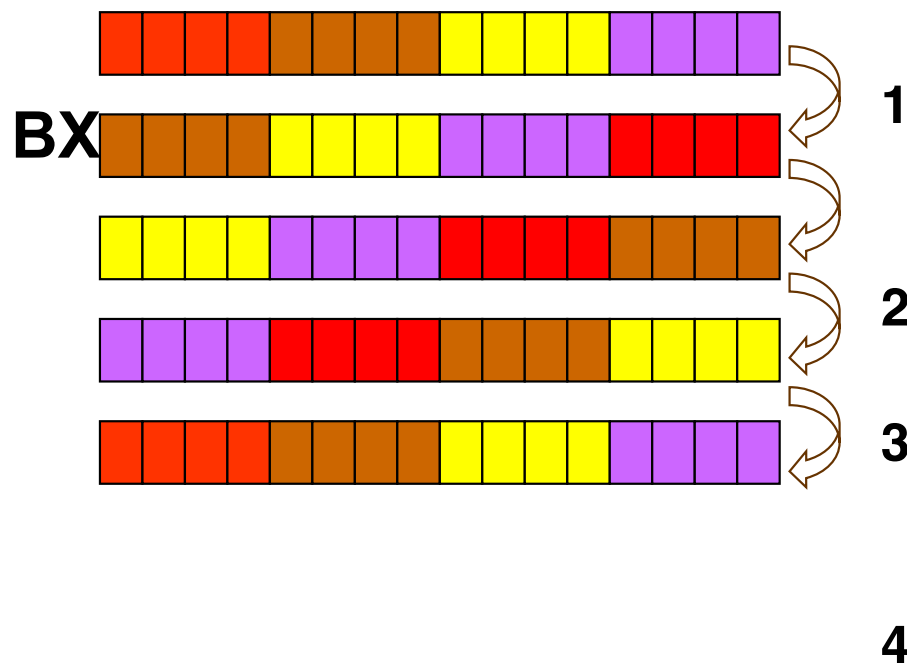
“0”~“9” [?] 30H~39H, “A”~“F” [?] 41H~46H

◆如何显示一个字符？

(a) 将显示字符的ASCII码放入DL寄存器；(b)将AH的内容置为2（功能号）；（
c）执行INT 21H（DOS 功能调用）。

十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符
48	0	64	@	80	P	96	`	112	p
49	1	65	A	81	Q	97	a	113	q
50	2	66	B	82	R	98	b	114	r
51	3	67	C	83	S	99	c	115	s
52	4	68	D	84	T	100	d	116	t
53	5	69	E	85	U	101	e	117	u
54	6	70	F	86	V	102	f	118	v
55	7	71	G	87	W	103	g	119	w
56	8	72	H	88	X	104	h	120	x
57	9	73	I	89	Y	105	i	121	y
58	:	74	J	90	Z	106	j	122	z
59	;	75	K	91	[107	k	123	{
60	<	76	L	92	\	108	l	124	
61	=	77	M	93]	109	m	125	}
62	>	78	N	94	^	110	n	126	~
63	?	79	O	95	_	111	o	127	△

画出程序流程图



循环结构程序设计

28/64

datas segment

temp dw 0B2FAH

datas ends

codes segment

assume cs:codes, ds:datas

start:

mov ax, datas

mov ds, ax

mov bx, temp

mov ch, 4 ; 循环次数

rotate: mov cl, 4 ; 移位次数

rol bx, cl ; bx循环左移4位

mov al, bl ; 移位后的低8位送al

and al, 0fh ; 取al的低四位

add al, 30h ; 0-9转ascii码, 加30h

cmp al, 3ah ; 比较是否是a-f

jl printit ; 如果小于3ah, 说明是0-9, 直接输出

add al, 7h ; 反之说明是a-f, 则加7

printit: mov dl, al

mov ah, 2

int 21h

dec ch

jnz rotate

mov ah, 4ch

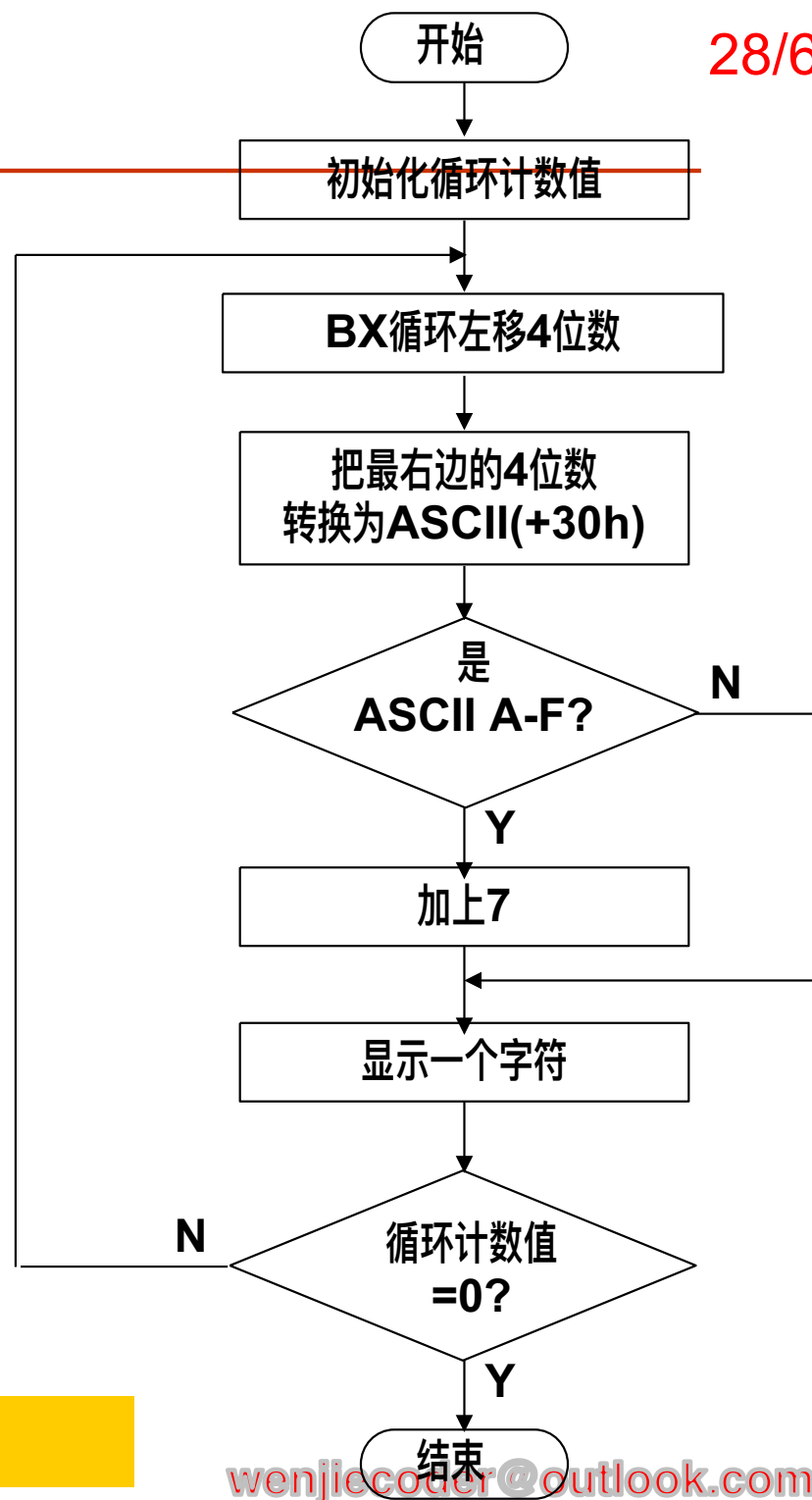
int 21h

codes ends

end start

输出DL寄存器的字符到显示器

; 若zf不为0则循环



思考：为什么不用LOOP指令实现？

```
    mov cx, 4    ; 初始化
rotate: push cx
        mov cl, 4
        rol bx, cl
        mov al, bl
        and al, 0fh
        add al, 30h ; '0'~'9' ASCII 30H~39H
        cmp al, 3ah
        jl printit
        add al, 7h ; 'A'~'F' ASCII 41H~46H
printit: mov dl, al
        mov ah, 2
        int 21h
        pop cx
        loop rotate
```

.....

方法2 (LOOP)

【例5.5】有数组 $x(x_1, x_2, \dots, x_{10})$ 和 $y(y_1, y_2, \dots, y_{10})$, 编程计算 $z(z_1, z_2, \dots, z_{10})$

$$z_1 = x_1 + y_1$$

$$z_2 = x_2 + y_2$$

$$z_3 = x_3 - y_3$$

$$z_4 = x_4 - y_4$$

$$z_5 = x_5 - y_5$$

$$z_6 = x_6 + y_6$$

$$z_7 = x_7 - y_7$$

$$z_8 = x_8 - y_8$$

$$z_9 = x_9 + y_9$$

$$z_{10} = x_{10} + y_{10}$$

分析题目：

(1) 程序结构的确定

可以用循环结构来完成，且循环次数已知，计数值为10。

(2) 循环体的构成

循环体应该包括：每次顺序取出操作数进行相应的运算。

(3) 难点

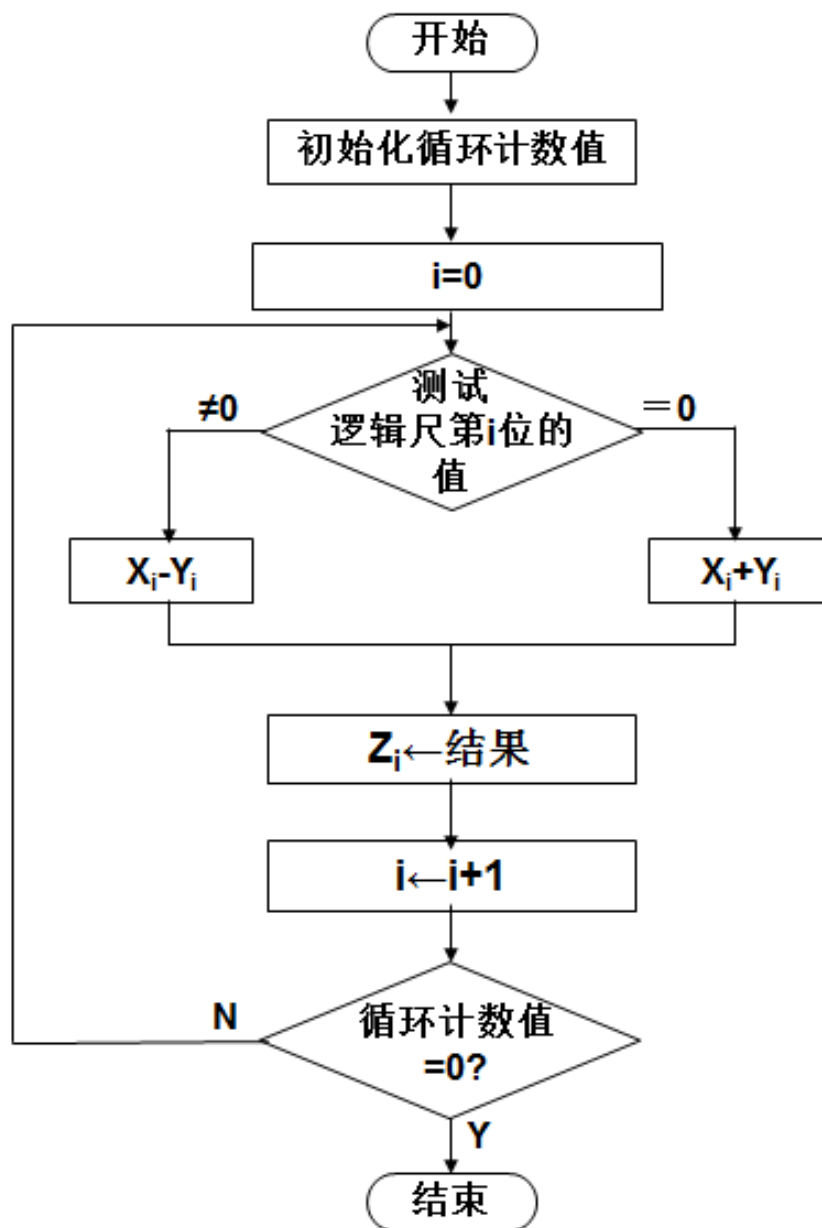
每次的运算不完全一样，可能为加法或者减法。

逻辑尺：0 0 1 1 1 0 1 1 0 0

1 减法

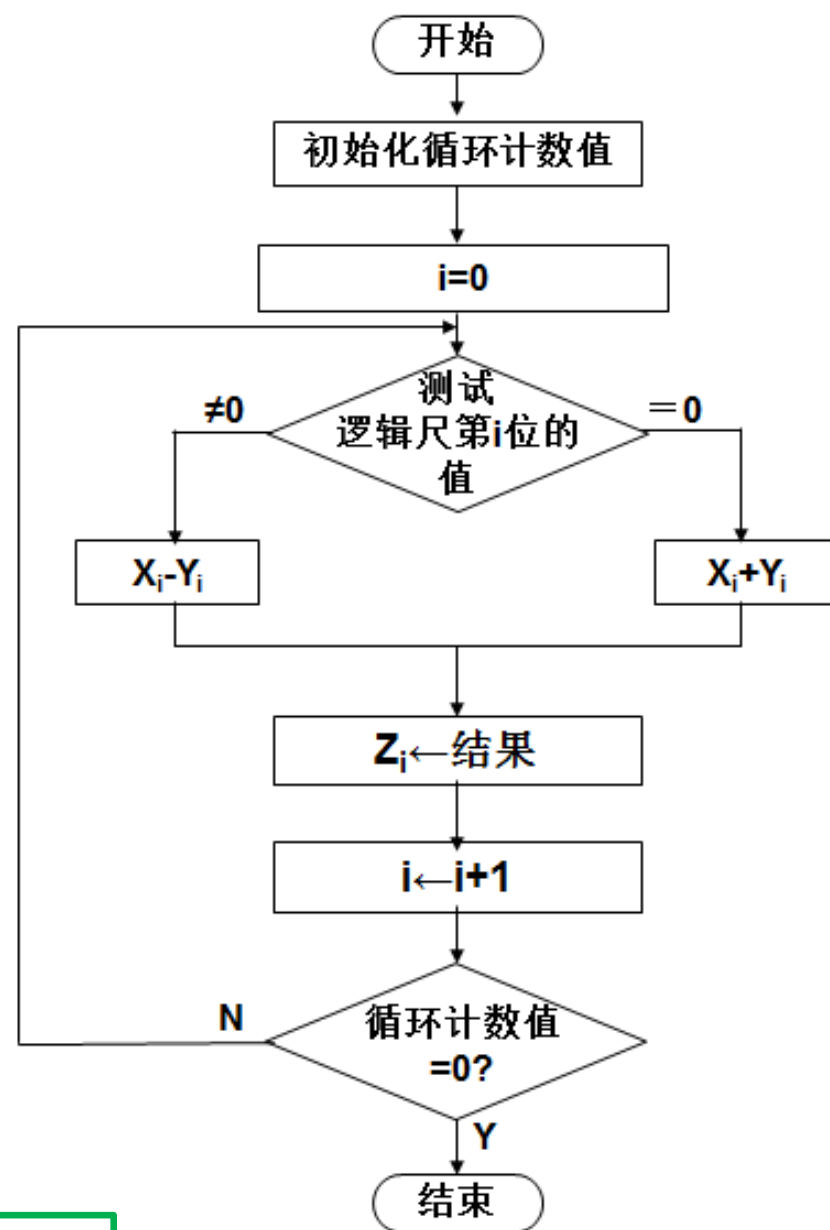
0 加法

画出程序流程图



```

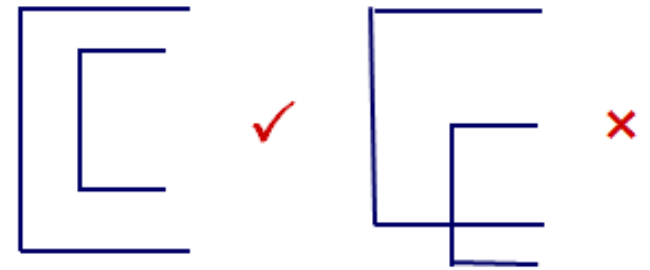
x    dw  x1,x2,x3,x4,x5,x6,x7,x8,x9,x10
y    dw  y1,y2,y3,y4,y5,y6,y7,y8,y9,y10
z    dw  z1,z2,z3,z4,z5,z6,z7,z8,z9,z10
logic_rule dw 0DCH ;0000,0000,1101,1100
               .....
               从右往左
mov   bx, 0      ; 数组索引
mov   cx, 10     ; 循环次数
mov   dx, logic_rule
next: mov ax, x[bx]
      shr dx, 1   ; 逻辑尺逻辑右移
      jc  subtract ; CF=1, 减法
      add ax, y[bx] ; CF=0, 加法
      jmp short result ; 跳转
subtract:
      sub ax, y[bx]
result:
      mov z[bx], ax ; 输出到Z
      add bx, 2     ; 修改数组索引
      loop next     ; CX不为0则循环
               .....
    
```



为什么需要先把一个加数放到寄存器:
两个操作数不能同时为存储器寻址方式。

循环结构程序设计

多重循环程序设计:



循环可以有多重结构。多重循环程序设计的基本方法和单重循环程序设计基本一致的，应分别考虑各重循环的控制条件及其程序实现，相互之间不能混淆。

内循环必须完整地包含在外循环内，内外循环不能相互交叉。

内循环在外循环中的位置可根据需要任意设置，在设计内、外循环时要避免出现混乱。

多个内循环可以拥有一个外循环，这些内循环间的关系可以是嵌套的，也可以是并列的

。当通过外循环再次进入内循环时，内循环中的初始条件必须重新设置。

无论是外循环，还是内循环，注意不要使循环返回到初始部分，以避免出现“死循环”情况。

注意：在多重循环的程序结构中，要注意**CX** 计数器的保存和恢复

```
    MOV CX, M
AGAIN: .....
    PUSH CX
    MOV CX, N
NEXT: .....
    LOOP NEXT
    .....
    POP CX
    LOOP AGAIN
```

利用堆栈保存和恢复**CX**

```
    MOV DI, M
AGAIN: .....
    MOV CX, N
NEXT: .....
    LOOP NEXT
    .....
    DEC DI
    JNZ AGAIN
```

分别用不同的寄存器来进行内外循环计数，避免出错。

【例5.7】有一个首地址为A的N字数组，编写程序采用冒泡排序使该数组中的数按照从大到小的次序排序。

冒泡法排序算法思想：

- 初始时，数组为无序排列。在排序过程中，数组会有一部分元素处于无序状态，称为无序集合，另一部分元素处于有序状态，称为有序集合。
- 在排序过程的每一轮（遍）比较中，从第一个数依次对无序集合中相邻两个数进行比较，如果次序不对，就交换位置。经过一轮排序后，最后的元素便是无序数组中最小的元素。将此元素加入有序集合，而无序集合减少一个元素。
- 依次循环进行，直到所有元素都进入有序集合。

序号	地址	数	比 较 遍 数			
			1	2	3	4
1	A	8	8	16	84	84
2	A+2	5	16	84	32	32
3	A+4	16	84	32	16	16
4	A+6	84	32	8	8	8
5	A+8	32	5	5	5	5

- ◆ N个数排序，第一遍比较了N-1次，把最小的数排到了第N的位置。
- ◆ 第二遍比较了N-2次，把第二小的数排到了第N-1的位置。以此类推
- ◆ 最终只需比较N-1遍，每遍比较次数递减。

【例5.7】 有一个首地址为A的N数组，编写程序采用冒泡排序使该数组中的数按照从大到小的次序整序。

分析题目：

(1)程序结构的确定

可以用双重循环结构来完成，且循环次数已知：外循环N-1次，内循环次数从N-1递减。

(2)循环体的构成

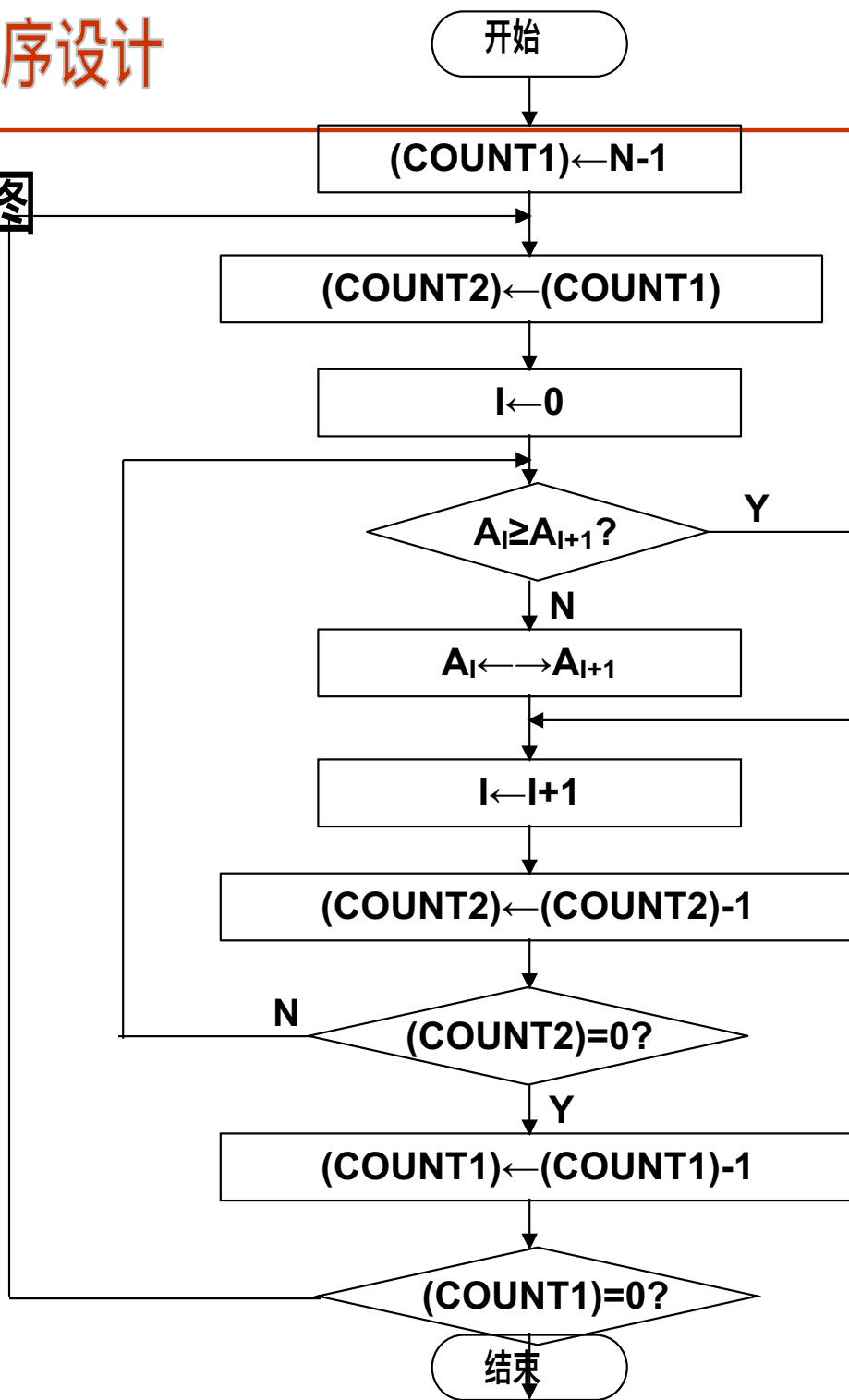
外循环：及时更新循环修改部分，包括内循环次数；

内循环：比较相邻两个元素，如果次序不对，则交换位置。

(3)难点

注意内外循环中循环计数。如使用loop指令，则要注意cx寄存器的保存和恢复。

画出程序流程图

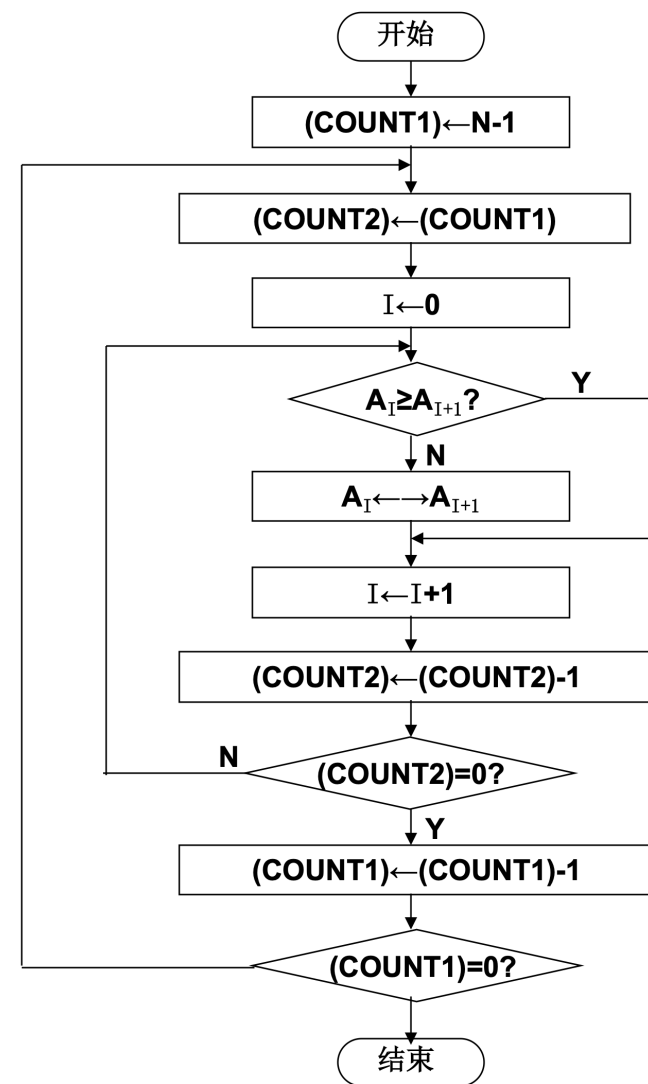


Count1:
比较遍数

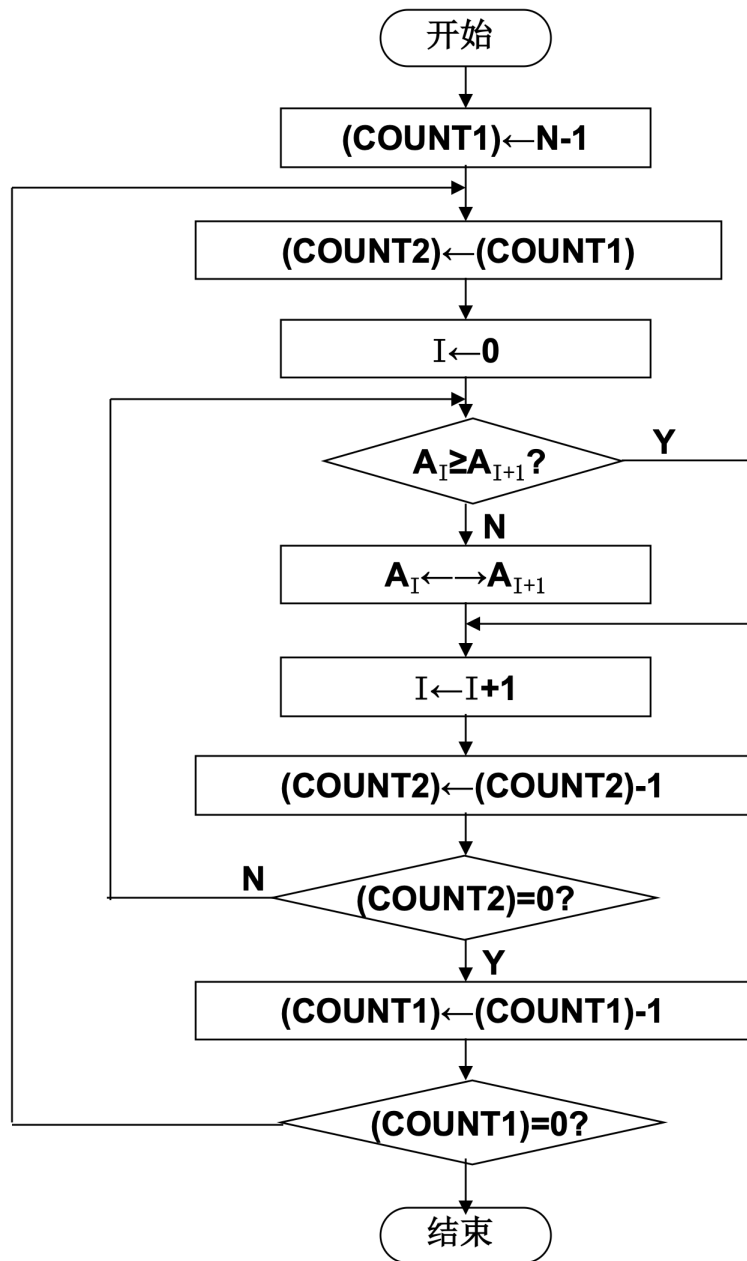
Count2:
每遍比较次数

```
data    segment
        a      dw
        8,16,84,32, 5
        n      equ    ($-
a)/2
data    ends

prognam segment
main    proc    far
        assume cs:prognam,ds:data
start:
        mov     ax, data
        mov     ds, ax
        mov     cx, n
        dec     cx; 比较遍数n-1
```



注意：Loop指令会自动修改CX的值，多重循环的时候要注意CX的保存和恢复。



```

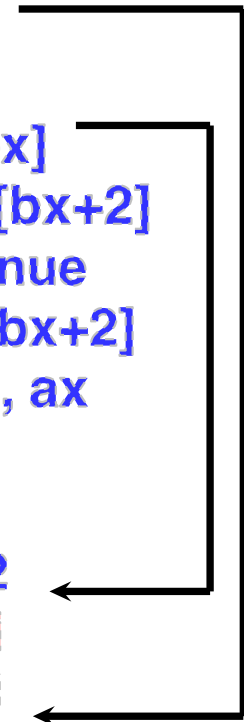
loop1:  mov     di,cx
        mov     bx,0
loop2:  mov     ax,a[bx]
        cmp     ax,a[bx+2]
        jge     continue
        xchg    ax,a[bx+2]
        mov     a[bx],ax
continue:
        add     bx,2
        loop    loop2
        mov     cx,di
        loop    loop1
        ret
main endp
prognam ends
        end     start
    
```

注意：Loop指令会自动修改CX的值，多重循环的时候要注意CX的保存和恢复。


```
data segment
    a dw 8,16,84,32,5
    n equ ($-a)/2
data ends

prognam segment
main proc far
    assume cs:prognam,ds:data
start:
    mov ax, data
    mov ds, ax
    mov cx, n
    dec cx; 比较次数n-1
```

```
loop1: mov di, cx
        mov bx, 0
loop2:  mov ax, a[bx]
        cmp ax, a[bx+2]
        jge continue
        xchg ax, a[bx+2]
        mov a[bx], ax
continue:
        add bx, 2
        loop loop2
        mov cx, di
        loop loop1
        ret
main endp
prognam ends
end start
```



注意：Loop指令会自动修改CX的值，多重循环的时候要注意CX的保存和恢复。

控制转移指令

循环结构设计

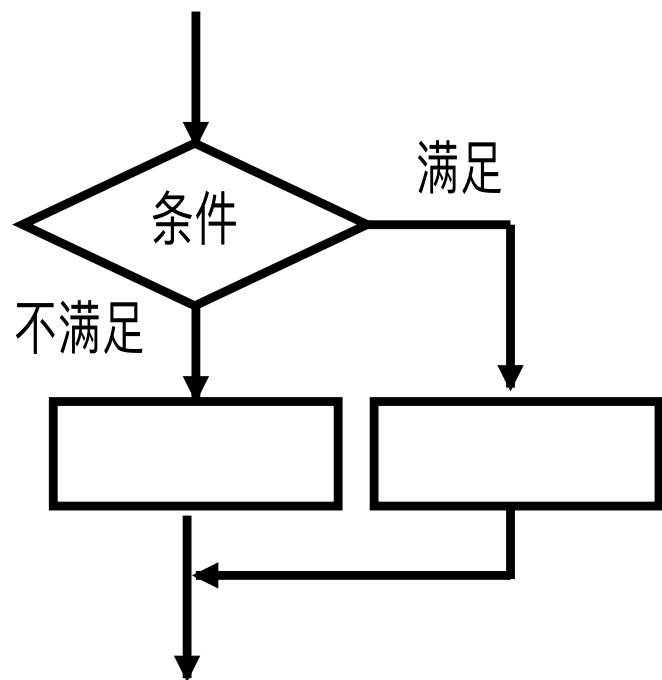
分支结构设计

分枝结构程序设计

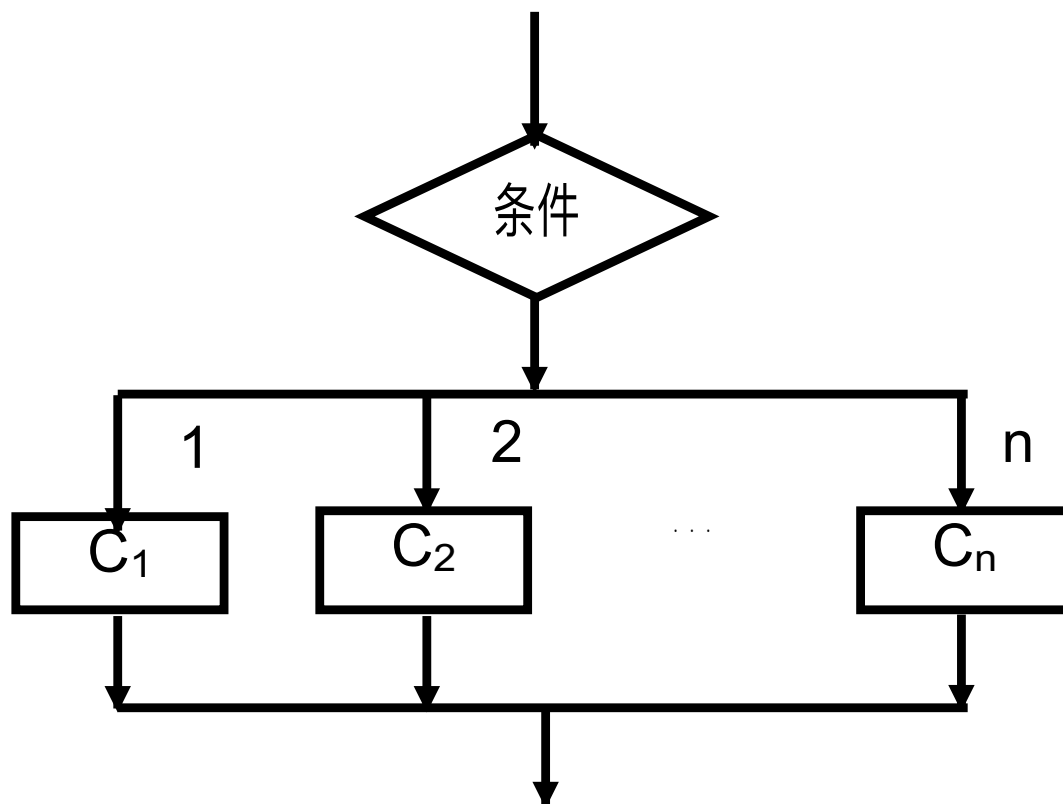
在程序中，往往需要对不同的情况或条件进行不同的处理，这样的程序就不再是简单的顺序结构，而要采用分枝结构。

分支程序结构可以有两种基本形式，即双分支结构和多分支结构。

不论哪种形式，共同点是：运行方向是向前的，在某一特定条件下，只能执行多个分支中的一个分支。



IF_THEN_ELSE结构



CASE结构

双分枝结构:

使用条件转移指令与无条件转移指令JMP来实现分支：一般必须先进行比较或算术、逻辑运算等影响标志位的指令，然后用条件转移指令判断条件，以实现分支转移。

IF-THEN结构:

```
    cmp AX, BX
    JE  EndOfIF
        <THEN 程序段>
```

EndOfIF:

注意：程序隐含是顺序执行的，在THEN分支体执行后，不会自动跳过ELSE分支体，而是继续执行其后的代码。

IF-THEN-ELSE结构:

```
    cmp AX, BX
    JE  ElseCode
        <THEN 程序段>
    jmp EndOfIF
```

ElseCode:

```
        <ELSE 程序段>
```

EndOfIF:

例:

已知在内存中有一个字节单元**NUM**，存有带符号数据，要求计算出它的绝对值后，放入**RESULT**单元中。

```
DATA    SEGMENT
        X          DB    -25
        RESULT DB    ?
DATA ENDS
CODE SEGMENT
        ASSUME DS:DATA,CS:CODE
START:   MOV     AX, DATA
         MOV     DS, AX           ;初始化
         MOV     AL, X           ;X取到AL中
         TEST    AL, 80H         ;测试AL正负
         JZ      NEXT           ;为正，转NEXT
         NEG     AL             ;否则AL求补
NEXT:    MOV     RESULT, AL      ;送结果
         MOV     AH, 4CH
         INT     21H            ;返回DOS
CODE ENDS
        END     START          ;汇编结束
```

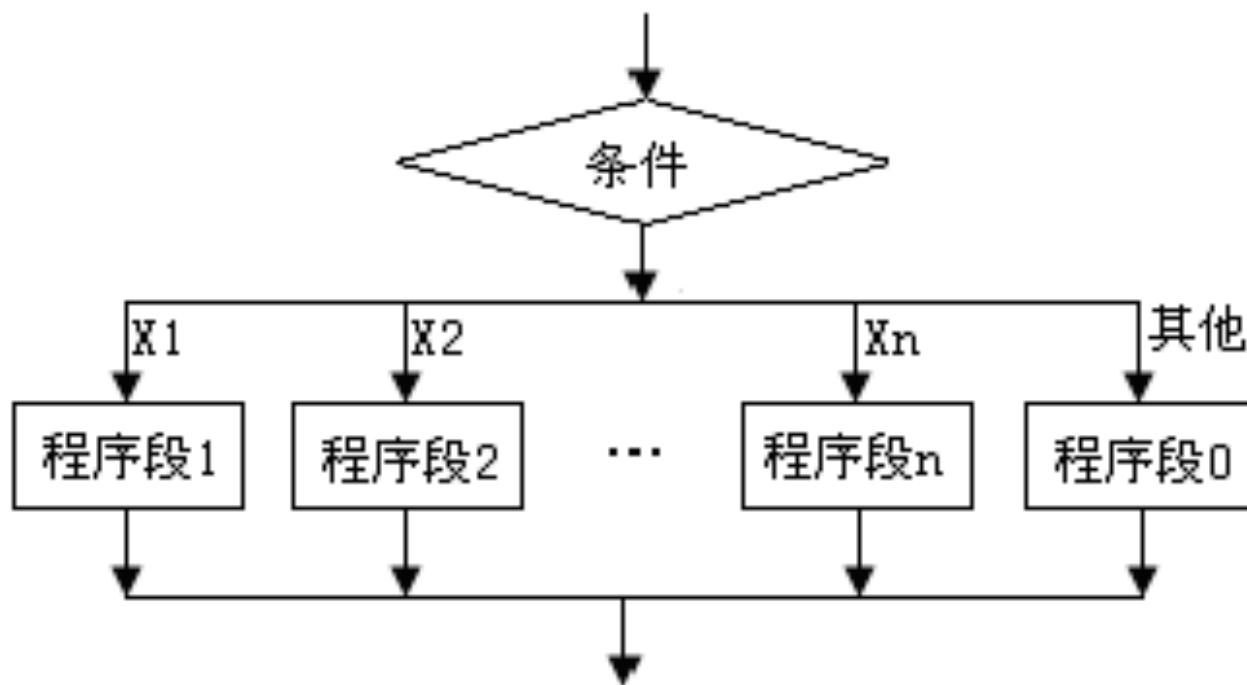
分枝结构程序设计

多分枝结构：

多分支结构是指有两个以上的分支。在程序设计时，有时要求对多个条件同时进行判断，根据判断的结果，可能有多分支要进行处理。

在汇编语言中，多分支只能由多次使用单分支方式予以实现。

设计方法：双分支法、逻辑分解法、跳跃表法、转移表法。



双分支法：由多个双分支实现多分枝程序设计

例.实现符号函数Y的功能。

其中： $-128 \leq X \leq +127$

$$Y = \begin{cases} 1 & \text{当 } X > 0 \text{ 时} \\ 0 & \text{当 } X = 0 \text{ 时} \\ -1 & \text{当 } X < 0 \text{ 时} \end{cases}$$

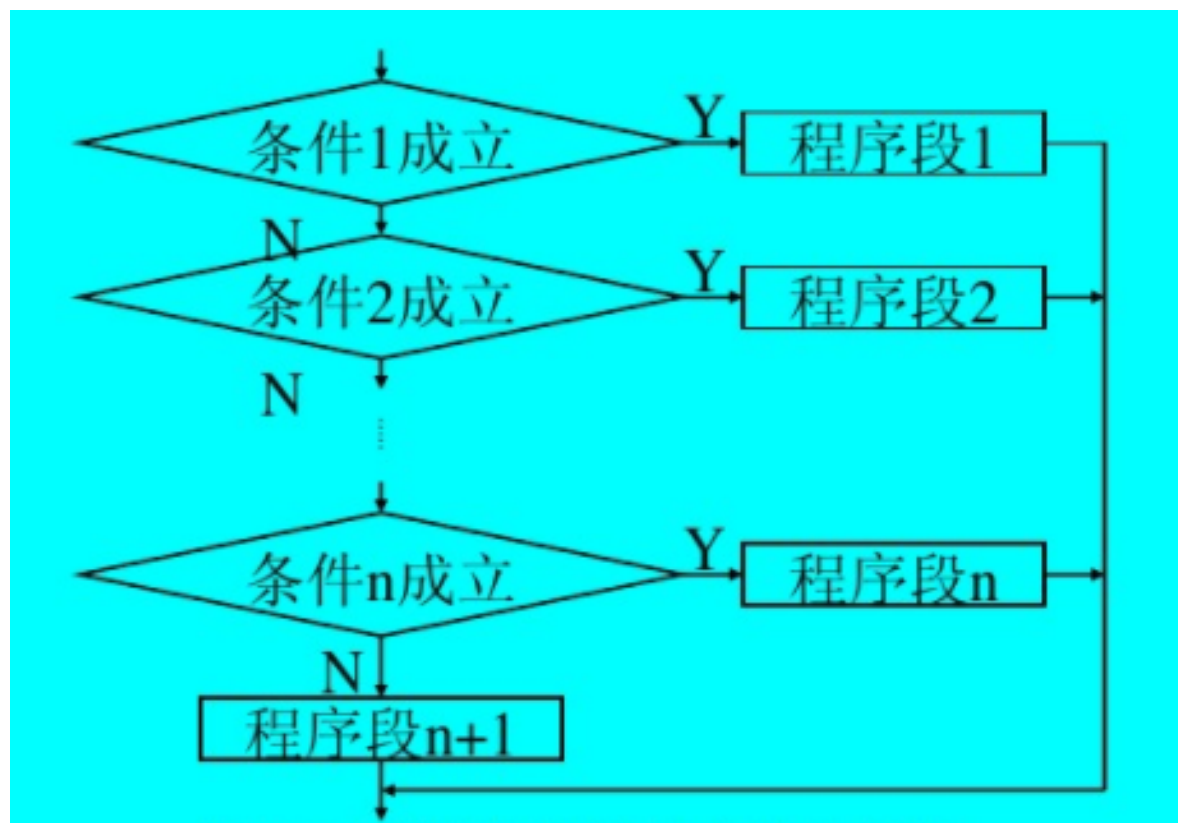
代码：

```
X      DB      ?           ;被测数据
Y      DB      ?           ;函数值单元
...
      MOV      AL, 0
      CMP      X, AL
      JG       BIG
      JZ       SAV          ;等于0
      MOV      AL, 0FFH     ;小于0
      JMP      SAV
BIG:   MOV      AL, 1        ;大于0
SAV:   MOV      Y, AL       ;保存结果
```

分枝结构程序设计

逻辑分解法：

按条件成立的先后，依次逻辑分解成下图所示的一串双分支结构，然后使用双分支的方法来进行程序设计。



例：根据输入值（0-4）的不同，执行不同的操作。

```
DATA SEGMENT
NUM DB 2
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
    MOV DS,AX
    MOV AL,NUM
    CMP AL,0
    JZ NEXT0
    CMP AL,1
    JZ NEXT1
    CMP AL,2
    JZ NEXT2
    CMP AL,3
    JZ NEXT3
    CMP AL,4
    JZ NEXT4
```

```
NEXT0:MOV DL,30H
    JMP EXIT
NEXT1:MOV DL,31H
    JMP EXIT
NEXT2:MOV DL,32H
    JMP EXIT
NEXT3:MOV DL,33H
    JMP EXIT
NEXT4:MOV DL,34H
EXIT:MOV AH,2
    INT 21H
    MOV AH,4CH
    INT 21H
CODE ENDS
END START
```

字符和ASCII码
对应关系：
“0”~“9”
30H~39H

输出DL寄存器的字
符到显示器

补充

5个常用的INT 21H系统功能调用

一般步骤：
入口参数送到指定寄存器
系统功能号送到寄存器AH
用INT 21H执行功能调用

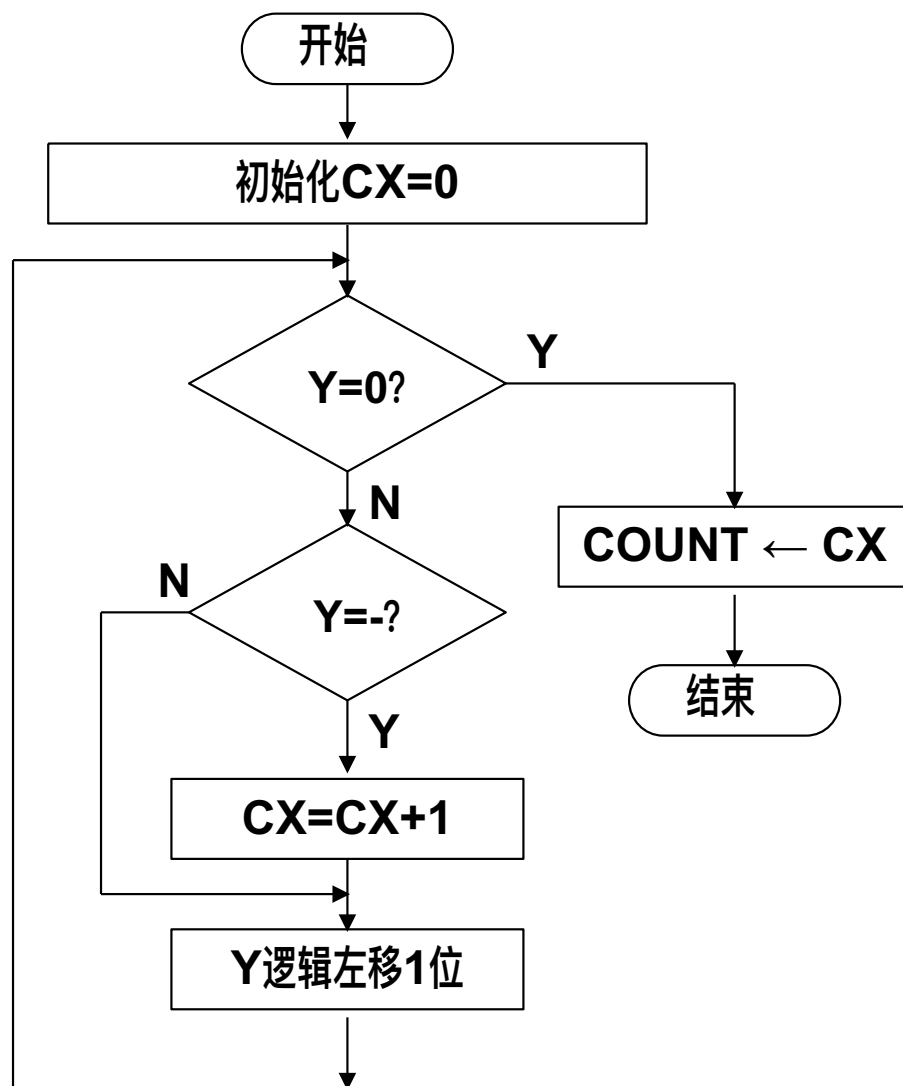
AH	功能	入口参数	出口参数
4CH	返回DOS	无	无
1	键盘输入一个字符到AL中	无	AL=字符
2	输出DL寄存器的字符到显示器	DL (存放一个字符)	无
9	输出一个以"\$"结尾的字符串到显示器	DS:字符串所在的段地址 DX:字符串首地址	无
0AH	从键盘输入一个字符串到指定缓冲区	DS:缓冲区所在的段地址 DX:缓冲区首地址	缓冲区相应位置

第6讲作业:

Page 193 -194: 5.3、5.6

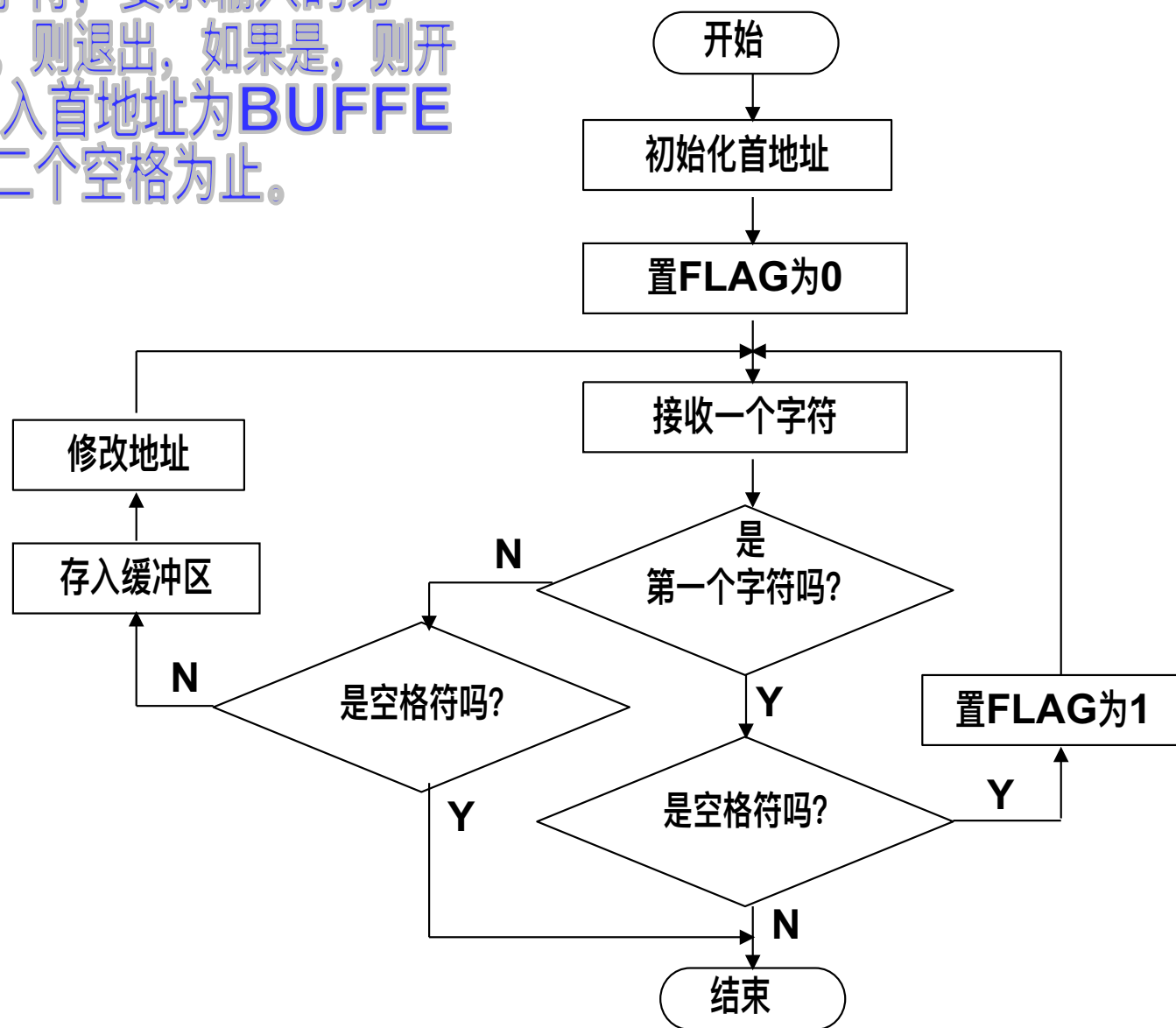
后面的例题感兴趣的同学可自学!

【例5.2】在Y中存放着16位数，试编制一个程序把Y中1的个数存入COUNT单元中。



```
data    segment
        Y      dw      06F3CH
        count  dw      ?
data    ends
prog    segment
        assume  cs: prog, ds: data
start:  mov     ax, data    ;数据段地址送ax
        mov     ds, ax
        mov     cx, 0      ;初始化CX=0
        mov     ax, Y      ;把数放到ax
testing: and    ax, ax     ;测试Y
        jz      exit      ;如果Y=0, 结束
        jns     shift     ;如果为正数, 不变
        inc     cx        ;否则CX=CX+1
shift:  shl     ax, 1      ;左移Y
        jmp     testing
exit:   mov     count, cx
        mov     ah, 4ch
        int     21h
prog    ends
        end     start
```

【例5.6】从键盘输入一行字符，要求输入的第一个字符必须是空格，如果不是，则退出，如果是，则开始接受输入的字符并顺序存入首地址为**BUFFER**的缓冲区，直到接收到第二个空格为止。

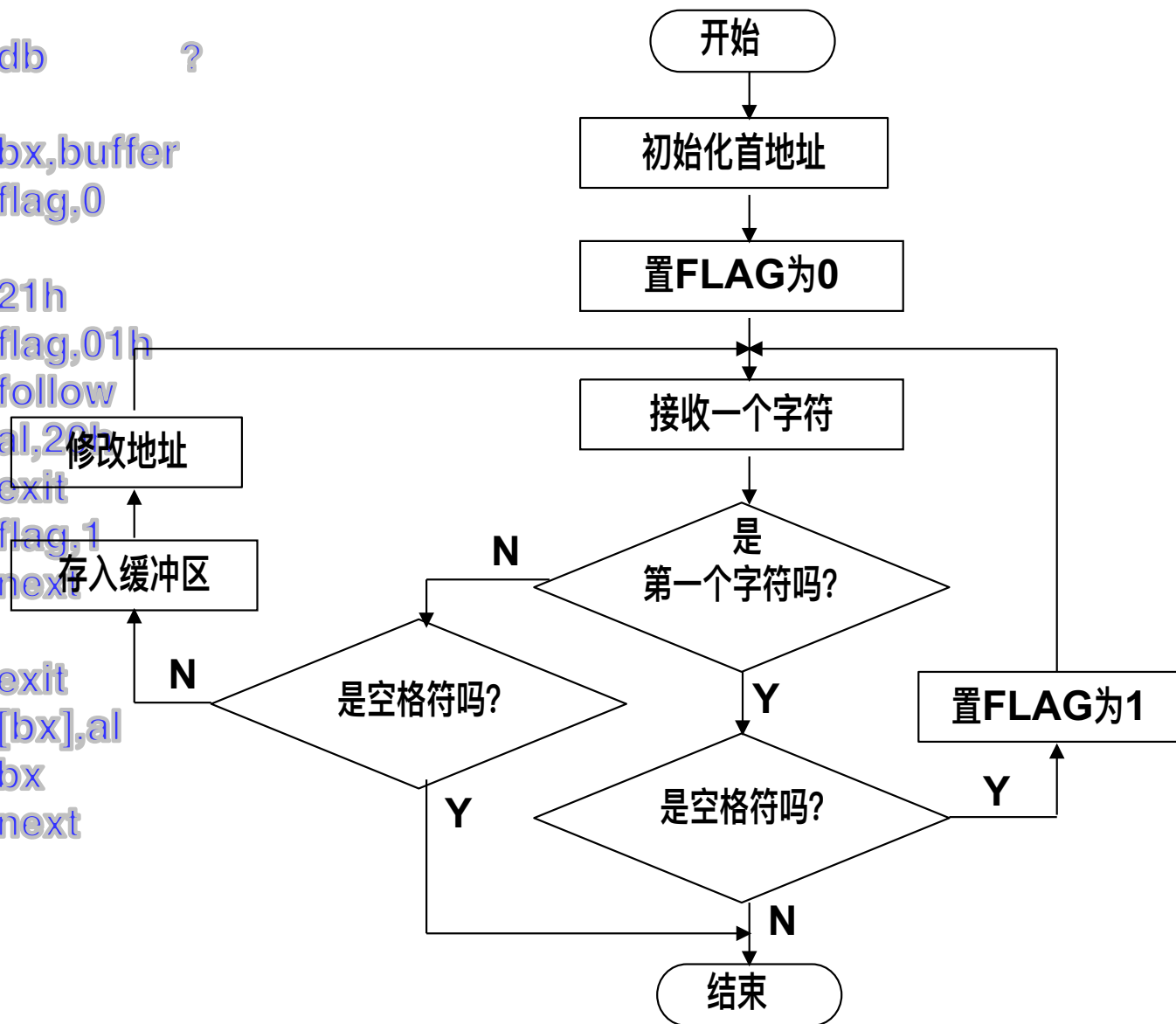


Flag=0: 第1个字符

Flag=1: 不是第1个字符

```

dup(?)      buffer    db      80
            flag      db      ?
            .....
            lea        bx,buffer
            mov        flag,0
next:      mov        ah,1
            int        21h
            test       flag,01h
            jnz        follow
            cmp        al,20h
            jnz        exit
            mov        flag,1
            jmp        next
follow:    cmp        al,20h
            jz         exit
            mov        [bx],al
            inc        bx
            jmp        next
exit:      .....
    
```



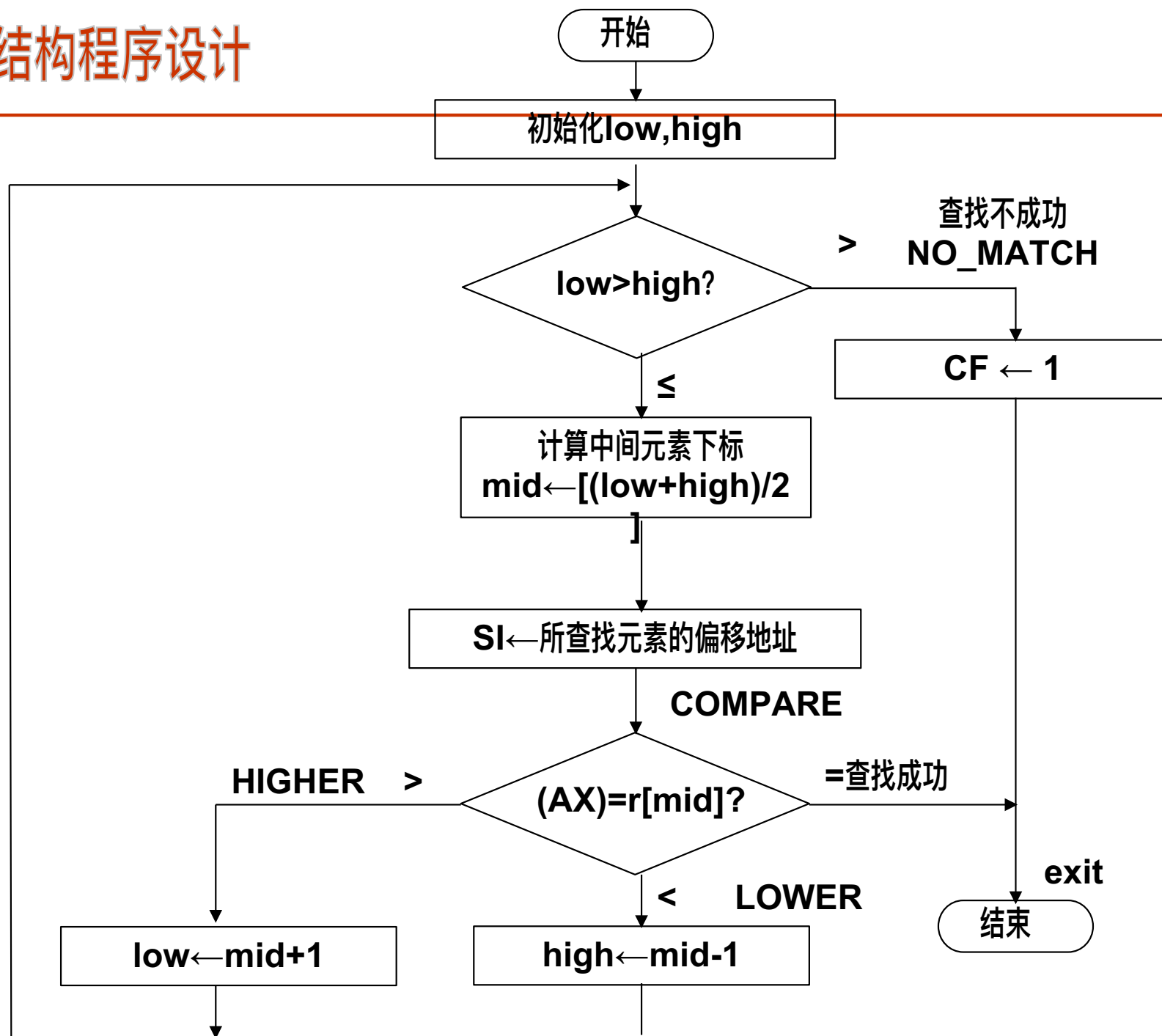
分枝结构程序设计

【例5.9】在数据段中，有一个按从小到大顺序排列的无符号数数组，其首地址存放在DI寄存器中，数组中的第一个字单元存放着数组长度。在AX中有一个无符号数，要求在数组中查找(AX)：如找到，则使CF=0，并在SI中给出该元素在数组中的偏移地址

如未找到，则使CF=1，并使SI中存放最后一次比较的数组元素的偏移地址

折半查找：在一个长度为n的有序数组r中，查找元素k的折半查找算法可描述如下：

- (1) 初始化被查找数组的首尾下标，low \leftarrow 1，high \leftarrow n；
- (2) 若low>high，则查找失败，置CF=1，退出程序。否则，计算中点mid \leftarrow [(low+high)/2]；
- (3) k与中点元素r[mid]比较。若k=r[mid]，则查找成功，程序结束；若k<r[mid]，则转(4)；若k>r[mid]，则转(5)；
- (4) 低半部分查找，high \leftarrow mid-1，返回(2)，继续查找；
- (5) 高半部分查找，low \leftarrow mid+1，返回(2)，继续查找。



折半算法1

			low_idx	high_idx	
0	12				
1	11		1	12	
2	22		1	5	
3	33	(ax)=55	4	5	(si)=0ah
4	44		5	5	Cf=0
5	55				
6	66				
7	77		low_idx	high_idx	
8	88		1	12	
9	99		7	12	
10	111	(ax)=90	7	8	
11	222		8	8	(si)=10h
12	333		9	8	Cf=1

分枝结构程序设计

```

...
    mov ax, number ;要查找数
    cmp ax, [di+2] ; (ax)与第一个元素比较
ja    chk_last
lea   si, [di+2]
je    exit        ; (ax)=第1个元素,找到退出
stc                                ;可以去掉
jmp   exit        ; (ax)<第一个元素,未找到退出
chk_last:
    mov si, [di] ; si=元素个数
shl   si,1
add   si,di      ; 计算最后一个元素的地址
    cmp ax,[si]
jb    search
je    exit        ; (ax)=最后一个元素,找到退出
stc
jmp   exit        ; (ax)>最后一个元素,未找到退出
search:
    mov low_idx,1
    mov bx,[di]
    mov high_idx,bx
    mov bx,di

```

```

mid: mov cx,low_idx
    mov dx,high_idx
    cmp cx,dx
    ja    no_match
    add cx,dx
    shr cx,1
    mov si,cx
    shl si,1
compare:
    cmp ax,[bx+si]
    je    exit
    ja    higher
        dec cx
        mov high_idx,cx
    jmp mid
higher: inc cx
    mov low_idx,cx
    jmp mid
no_match:
    stc
exit:    ...

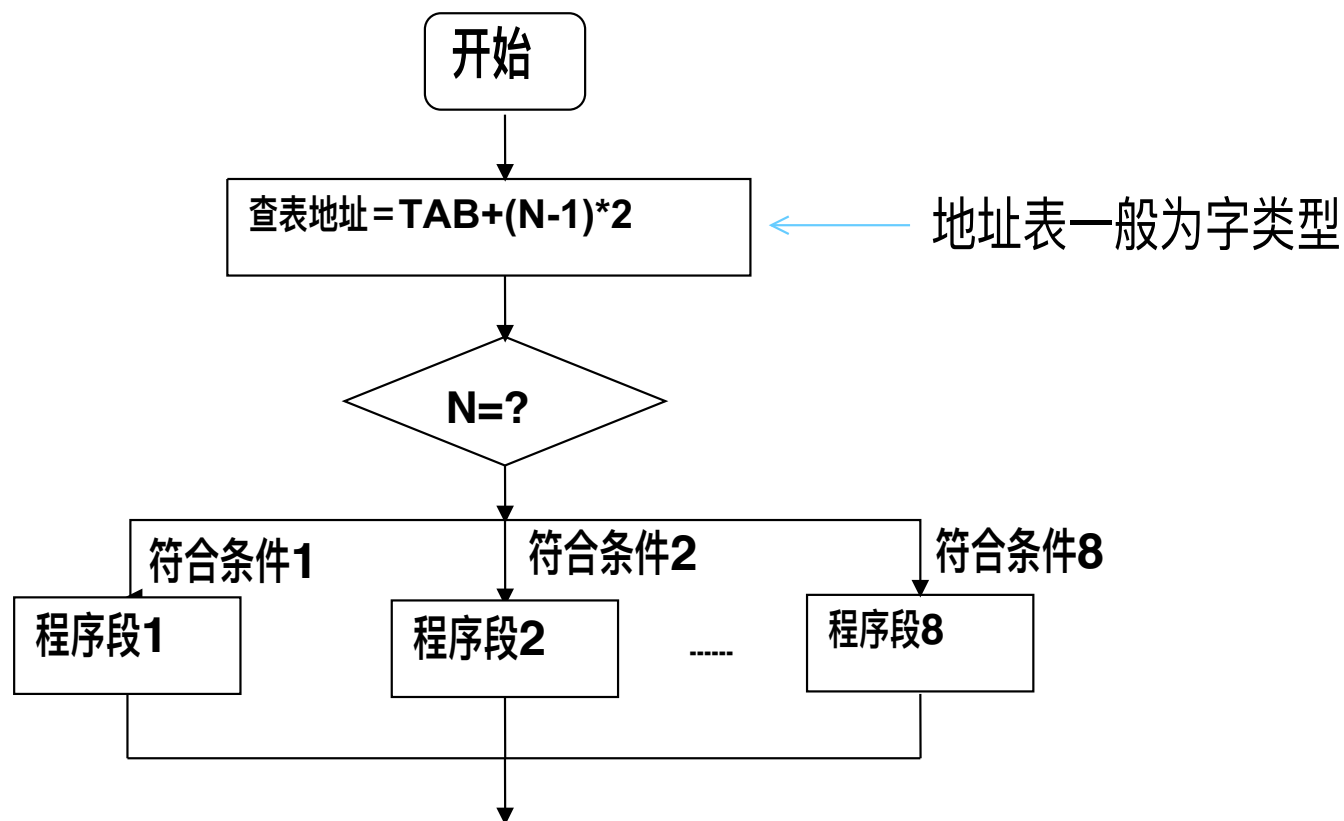
```

分枝结构程序设计

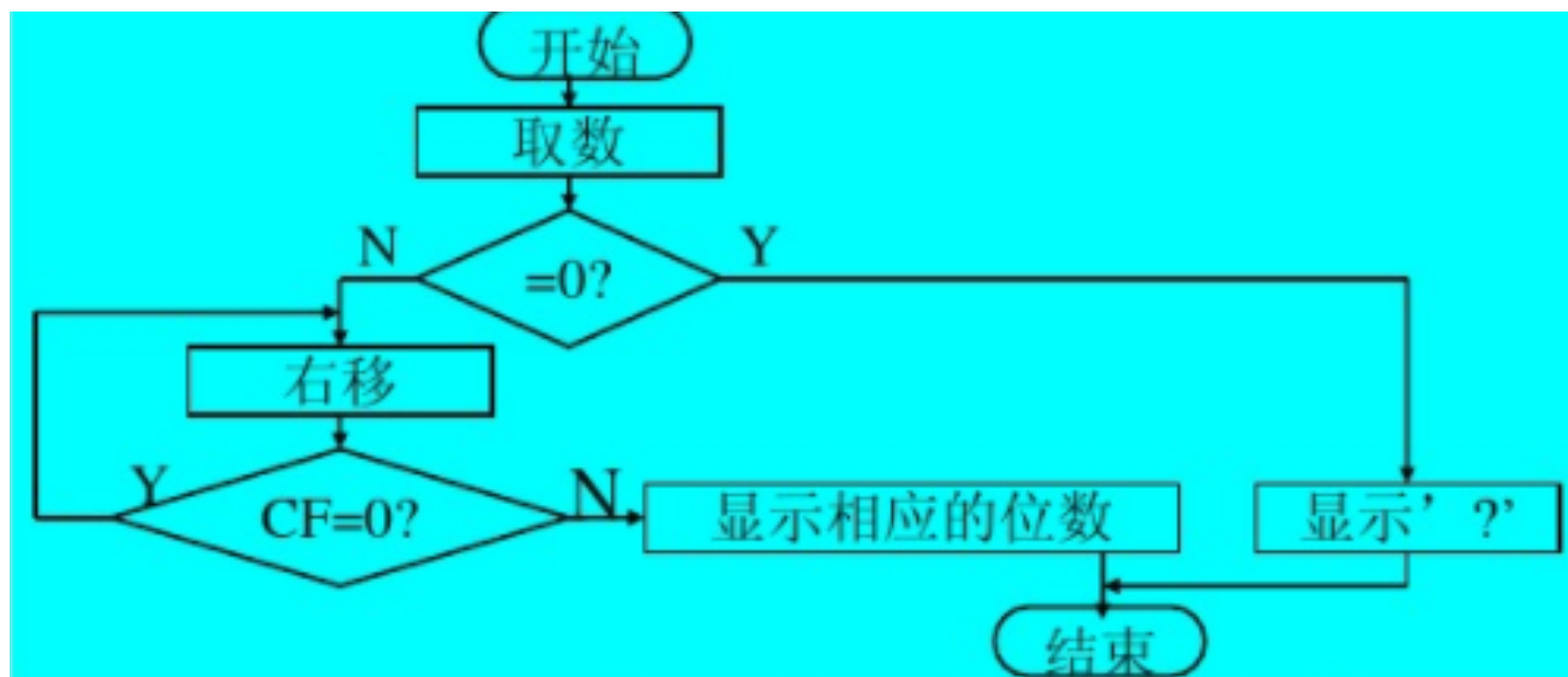
跳跃表法:

设有若干段分支程序，将每段分支程序的入口地址（也称跳跃地址）组成一个连续存放在内存中的表，称为跳跃表。

例：设某程序有8路分支，试根据给定的N值（1~8），将程序的执行转移到其中的一路分支。



例：用跳跃表法编程实现：从低到高逐位检测一个字节数据，找出第一个非0的位数。检测时，为零则继续检测，为1则转移到对应的处理程序段显示相应的位数。若数据本身为0，则显示？号。



分枝结构程序设计

DATAS SEGMENT

num db 78h

adtab dw ad0,ad1,ad2,ad3,ad4,ad5,ad6,ad7

DATAS ENDS

CODES SEGMENT

ASSUME CS:CODES,DS:DATAS

START:

MOV AX,DATAS

MOV DS,AX

mov al,num

mov dl,'?'

cmp al,0

jz disp ;为0则跳转显示?

mov bx,0 ;循环计数初值

again: shr al,1 ;测试最低位是否为1

jc next ;为1则跳转

inc bx

jmp again

next: shl bx,1 ;bx*2, 计算偏移量

jmp adtab[bx] ;跳转到指定分支

ad0: mov dl,'0'

jmp disp

ad1: mov dl,'1'

jmp disp

ad2: mov dl,'2'

jmp disp

ad3: mov dl,'3'

jmp disp

ad4: mov dl,'4'

jmp disp

ad5: mov dl,'5'

jmp disp

ad6: mov dl,'6'

jmp disp

ad7: mov dl,'7'

jmp disp

disp: mov ah,2

int 21h

mov ah,4ch

int 21h

CODES ENDS

END START

注意：每个分支程序的最后要有一条转移语句，以便跳过其他的分支。

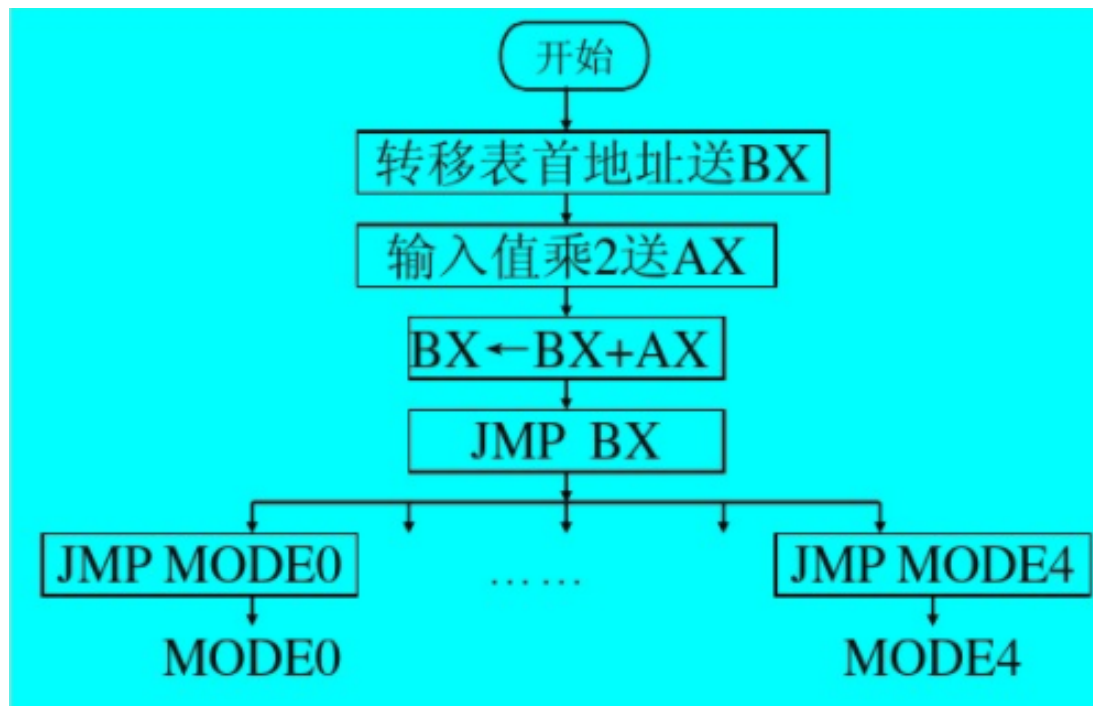
分枝结构程序设计

转移表法:

把转移到各分支程序段的转移指令依次存放在一起，形成转移表。各转移指令在表中的位置——

离表首地址的偏移量作为转移条件，偏移量加上表首地址作为转移地址，转到表的相应位置，执行相应的无条件转移指令。

例：根据输入值（0-4）的不同，执行不同的操作。



<pre>CODE SEGMENT ASSUME CS:CODE START:LEA BX,TAB MOV AH,1 INT 21H SUB AL,30H MOV AH,0 ADD AX,AX ADD BX,AX JMP BX TAB:JMP SHORT MODE0 ; 转移表 JMP SHORT MODE1 JMP SHORT MODE2 JMP SHORT MODE3 JMP SHORT MODE4</pre>	<pre>MODE0:MOV DL,30H JMP EXIT MODE1:MOV DL,31H JMP EXIT MODE2:MOV DL,32H JMP EXIT MODE3:MOV DL,33H JMP EXIT MODE4:MOV DL,34H EXIT:MOV AH,2 INT 21H MOV AH,4CH INT 21H CODE ENDS END START</pre>
--	---

说明：转移表中每条转移指令占用2个字节（段内短转移），所以有以下计算公式：表地址=模式字*2+表首地址