第五章　循环与分支程序设计

清華大學
Tsinghua University

主要内容：

- 循环程序设计
- 分支程序设计

编写汇编语言程序的步骤：

(1) 分析题意，确定算法

(2) 根据算法画出程序框图

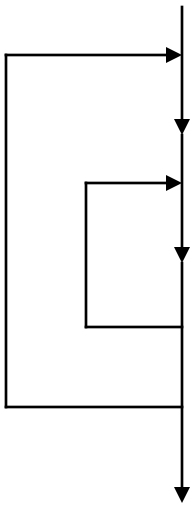
(3) 根据框图编写程序

(4) 上机调试程序

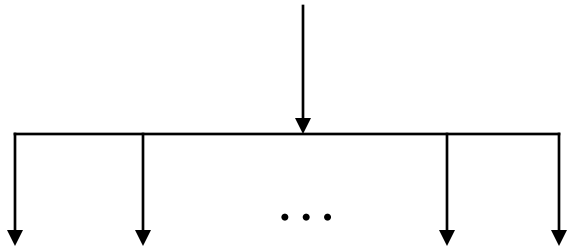
程序结构：

顺序结构

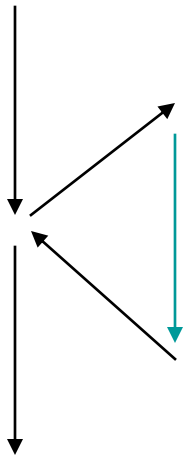
循环结构



分支结构

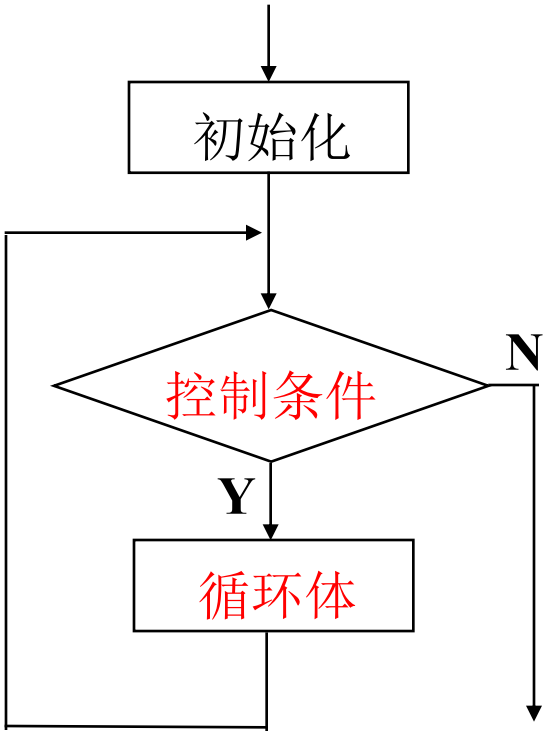


子程序结构

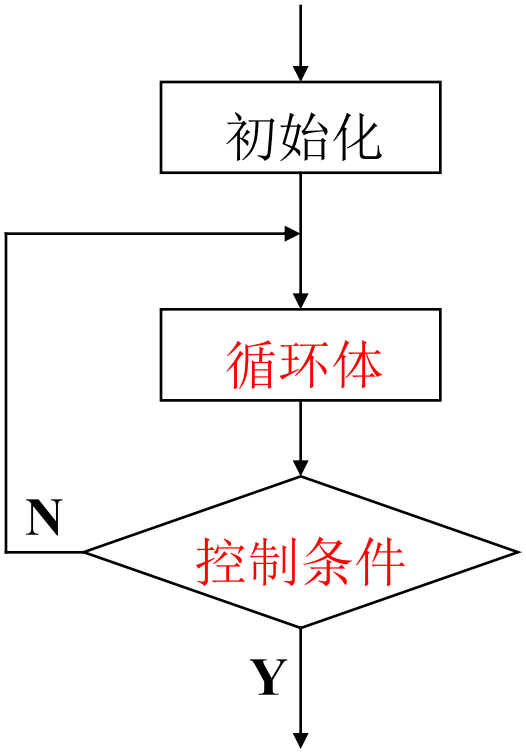


复合结构：多种程序结构的组合

1. 循环程序设计



DO-WHILE 结构



DO-UNTIL 结构

初始化：设置循环的初始状态

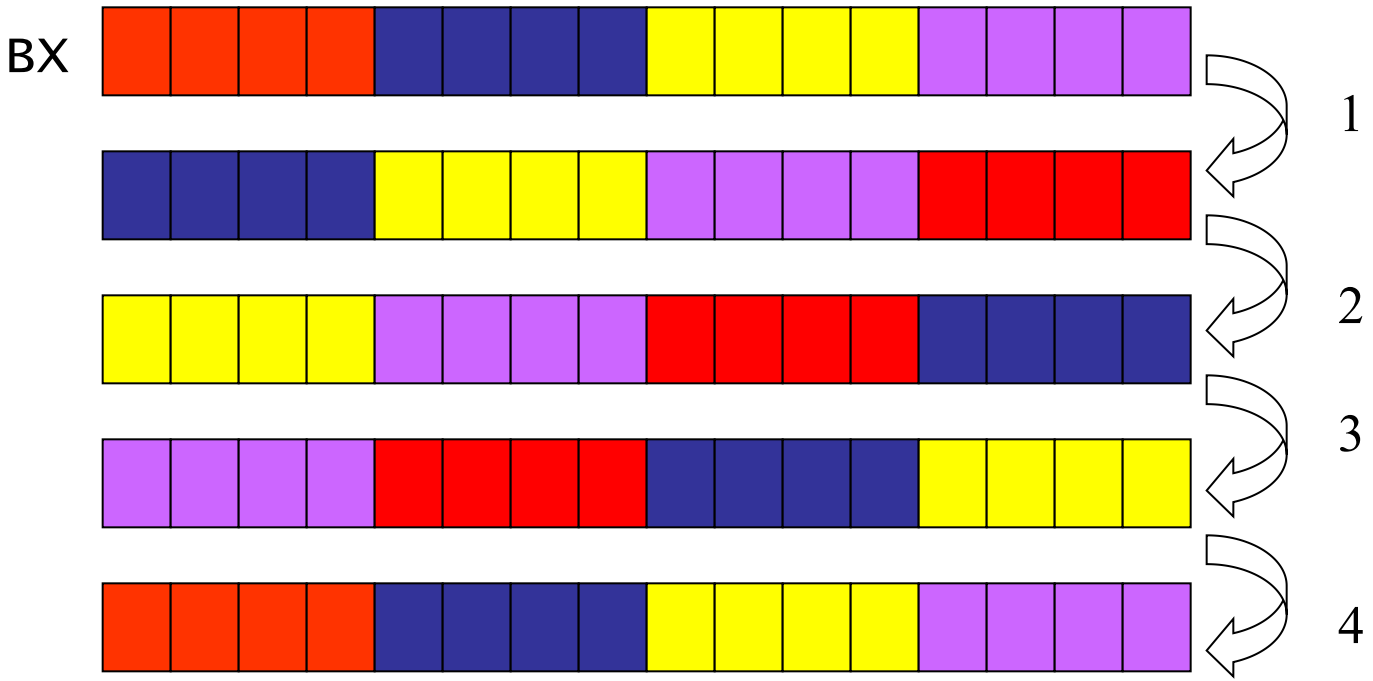
循环体：循环的工作部分及修改部分

控制条件：计数控制

特征值控制

地址边界控制

例：把BX中的二进制数以十六进制的形式显示在屏幕上



```
            ……
            mov   ch, 4
rotate:     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              ; 显示单个字符
            dec   ch
            jnz   rotate
            ……
```

例：从键盘接收十进制数并存入BX

```
            ……
            mov    bx, 0
newchar:    mov    ah, 1
            int    21h          ; 接收单个字符
            sub    al, 30h
            jl     exit         ; <0 退出
            cmp    al, 9
            jg     exit         ; >9 退出
            cbw

            xchg   ax, bx       ; 是否必须?
            mov    cx, 10
            mul    cx
            xchg   ax, bx       ; 是否必须?
            add    bx, ax

            jmp    newchar
exit:       ……
```

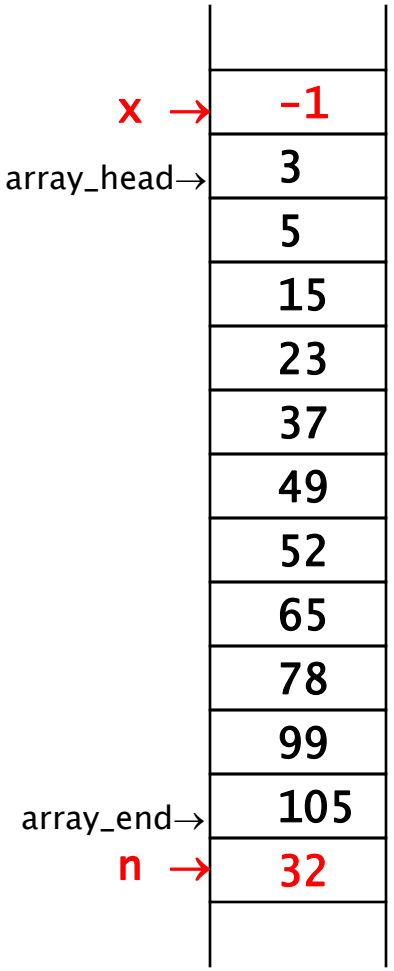
例：从键盘接收十六进制数并存入 BX

```
            ……
            mov   bx, 0
newchar:    mov   ah, 1
            int   21h
            sub   al, 30h
            jl    exit              ; <0 退出
            cmp   al, 10
            jl    add_to
            sub   al, 27h           ; ‘a’~‘f’
            cmp   al, 0ah
            jl    exit              ; < ‘a’  退出
            cmp   al, 10h
            jge   exit              ; >’ f’  退出
add_to:     mov   cl, 4
            shl   bx, cl
            mov   ah, 0
            add   bx, ax
            jmp   newchar
exit:       ……
```

例：将正数 n 插入一个已整序的正数字数组

```
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：将首地址为 a 的字数组从大到小排序

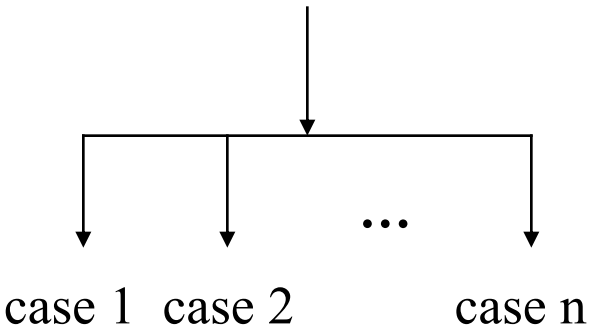
（起泡排序算法，多重循环）

100	100	100	100	100	100	100	100	189	256
30	78	99	99	99	99	99	189	256	189
78	99	78	78	78	78	189	256	100	100
99	30	30	66	66	189	256	99	99	99
15	15	66	45	189	256	78	78	78	78
-1	66	45	189	256	66	66	66	66	66
66	45	189	256	45	45	45	45	45	45
45	189	256	30	30	30	30	30	30	30
189	256	15	15	15	15	15	15	15	15
256	-1	-1	-1	-1	-1	-1	-1	-1	-1

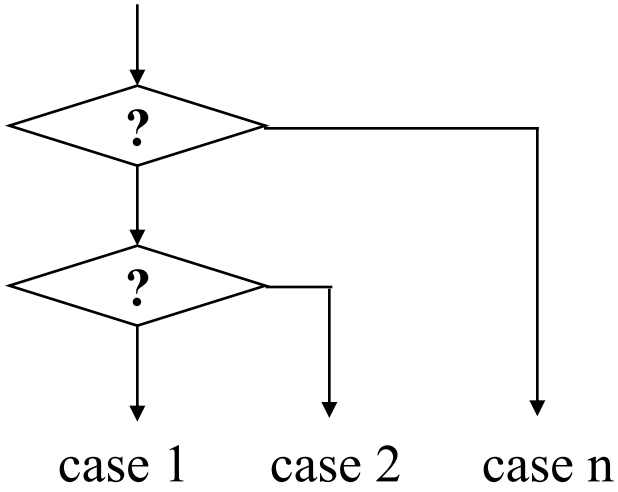
```
a    dw   100,30,78,99,15,-1,66,45,189,256

             ……
             mov    cx, 10
             dec    cx
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
             ……
```

2. 分支程序设计



CASE 结构



IF-THEN-ELSE 结构

(1) 逻辑尺控制

(2) 条件控制

(3) 地址跳跃表（值与地址有对应关系的表）

例：有数组 x(x1,x2,……,x10) 和 y(y1,y2,……,y10)，
编程计算 z(z1,z2,……,z10)

```
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
z10= x10 + y10
```

逻辑尺：0 0 1 1 0 1 1 1 0 0

1 减法

0 加法

```
            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   00dch

            ……
            mov    bx, 0
            mov    cx, 10
            mov    dx, logic_rule
next:       mov    ax, x[bx]
            shr    dx, 1
            jc     subtract
            add    ax, y[bx]
            jmp    short  result          ; 向前引用
subtract:
            sub    ax, y[bx]
result:     mov    z[bx], ax
            add    bx, 2
            loop   next
            ……
```

例：折半查找算法

```
data   segment
       array      dw   12,11,22,33,44,55,66,
                       77,88,99,111,222,333
       number     dw   55
       low_idx    dw   ?
       high_idx   dw   ?
data   ends
```

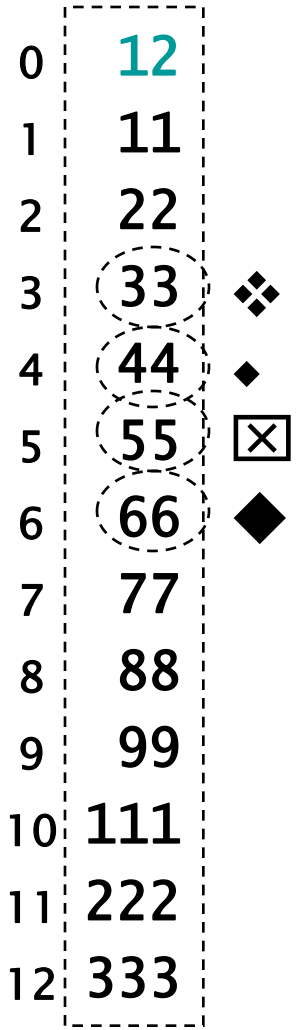
算法1

```
        ……
        lea   di, array
        mov   ax, number

        cmp   ax, [di+2]    ; (ax)与第一个元素比较
        ja    chk_last
        lea   si, [di+2]    ; si存放位置
        je    exit          ; (ax)=第一个元素,找到退出
        stc                 ; cf=1 查找失败
        jmp   exit          ; (ax)<第一个元素,未找到退出
chk_last:
        mov   si, [di]
        shl   si, 1
        add   si, di
        cmp   ax, [si]      ; (ax)与最后一个元素比较
        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
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:
    ……
```



(ax)=55

low_idx	high_idx	
1	12	
1	5	
4	5	(si)=0ah
5	5	cf=0

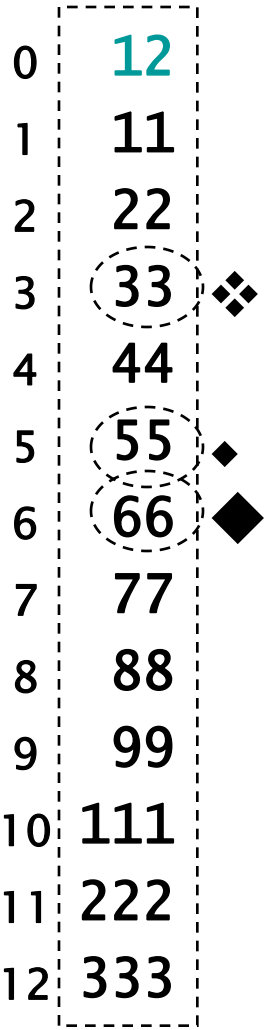
(ax)=90

low_idx	high_idx	
1	12	
7	12	
7	8	(si)=10h
8	8	
9	8	cf=1

算法2

```
search:
    mov  si, [di]
even_idx:
    test si, 1
    jz   add_idx
    inc  si
add_idx:
    add  di, si
compare:
    cmp  ax, [di]
    je   all_done
    ja   higher
    cmp  si, 2
    jne  idx_ok
no_match:
    stc
    jmp  exit
```

```
idx_ok:
    shr  si, 1
    test si, 1
    jz   sub_idx
    inc  si
sub_idx:
    sub  di, si
    jmp  short compare
higher:
    cmp  si, 2
    je   no_match
    shr  si, 1
    jmp  short even_idx
all_done:
    mov  si, di   ; si存放位置
exit:
    ……
```



(ax)=55

di	si	
		(si)=0ah
12	12	
6	6	(di)=0ah
10	4	cf=0

(ax)=90

di	si	
12	12	
18	6	(si)=2
14	4	(di)=10h
16	2	cf=1

例：根据 AL 寄存器中哪一位为 1（从低位到高位），
把程序转移到 8 个不同的程序分支

```
branch_table  dw  routine1
              dw  routine2
              dw  routine3
              dw  routine4
              dw  routine5
              dw  routine6
              dw  routine7
              dw  routine8
```

（寄存器间接寻址）

```
        ……
        cmp   al, 0
        je    continue
        lea   bx, branch_table
L:      shr   al, 1                    ;逻辑右移
        jnb   add1                     ;jnb=jnc
        jmp   word ptr[bx]             ;段内间接转移
add1:   add   bx, type branch_table
        jmp   L
continue:
        ……
routine1:
        ……
routine2:
        ……
```

（寄存器相对寻址）

```
        ……
        cmp   al, 0
        je    continue
        mov   si, 0
L:      shr   al, 1                    ;逻辑右移
        jnb   add1                     ;jnb=jnc
        jmp   branch_table[si]         ;段内间接转移
add1:
        add   si, type branch_table
        jmp   L
continue:
        ……
routine1:
        ……
routine2:
        ……
```

（基址变址寻址）

```
        ……
        cmp   al, 0
        je    continue
        lea   bx, branch_table
        mov   si, 7 * type branch_table
        mov   cx, 8
L:      shl   al, 1                          ;逻辑左移
        jnb   sub1                           ;jnb=jnc
        jmp   word ptr [bx][si]              ;段内间接转移
sub1:   sub   si, type branch_table
        loop   L
continue:
        ……
routine1:
        ……
routine2:
        ……
```

第五章　循环与分支程序设计

清華大學
Tsinghua University

主要内容：

- 循环程序设计
- 分支程序设计

编写汇编语言程序的步骤：

(1) 分析题意，确定算法

(2) 根据算法画出程序框图

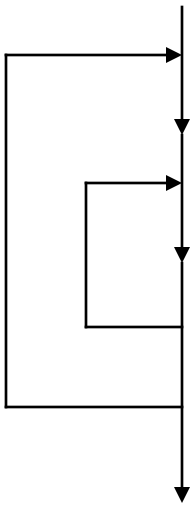
(3) 根据框图编写程序

(4) 上机调试程序

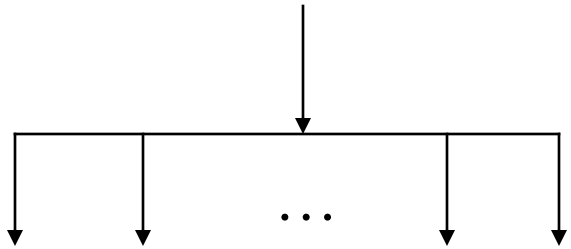
程序结构：

顺序结构

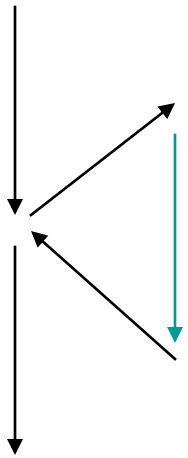
循环结构



分支结构

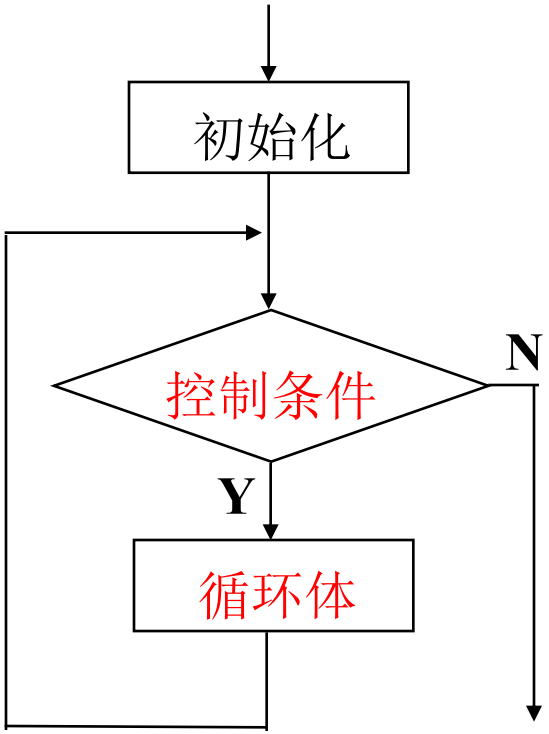


子程序结构

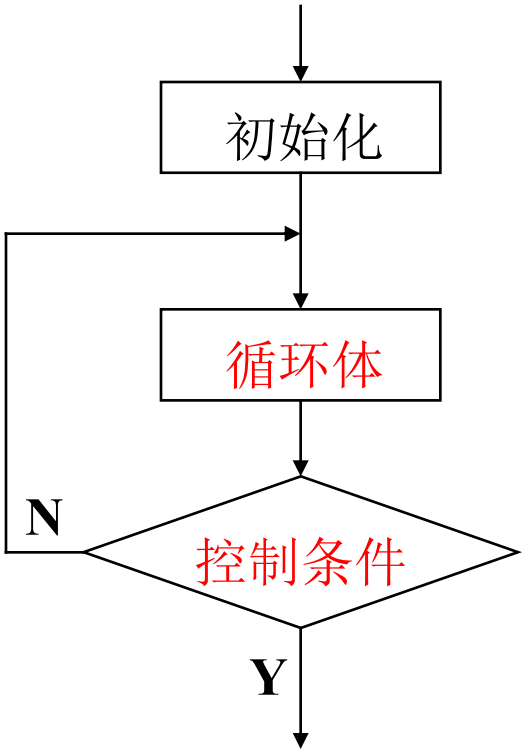


复合结构：多种程序结构的组合

1. 循环程序设计



DO-WHILE 结构



DO-UNTIL 结构

初始化：设置循环的初始状态

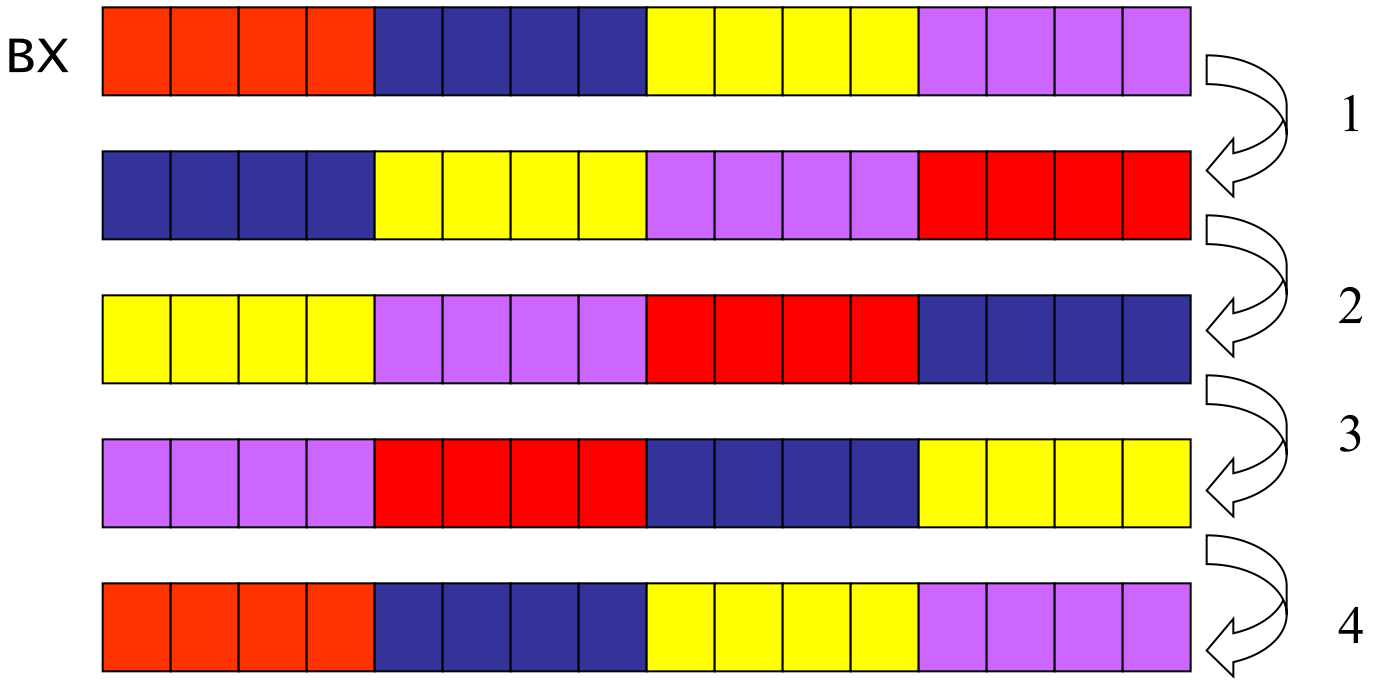
循环体：循环的工作部分及修改部分

控制条件：计数控制

特征值控制

地址边界控制

例：把BX中的二进制数以十六进制的形式显示在屏幕上



```
            ……
            mov   ch, 4
rotate:     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              ; 显示单个字符
            dec   ch
            jnz   rotate
            ……
```

例：从键盘接收十进制数并存入BX

```
            ……
            mov    bx, 0
newchar:    mov    ah, 1
            int    21h          ; 接收单个字符
            sub    al, 30h
            jl     exit         ; <0 退出
            cmp    al, 9
            jg     exit         ; >9 退出
            cbw

            xchg   ax, bx       ; 是否必须?
            mov    cx, 10
            mul    cx
            xchg   ax, bx       ; 是否必须?
            add    bx, ax

            jmp    newchar
exit:       ……
```

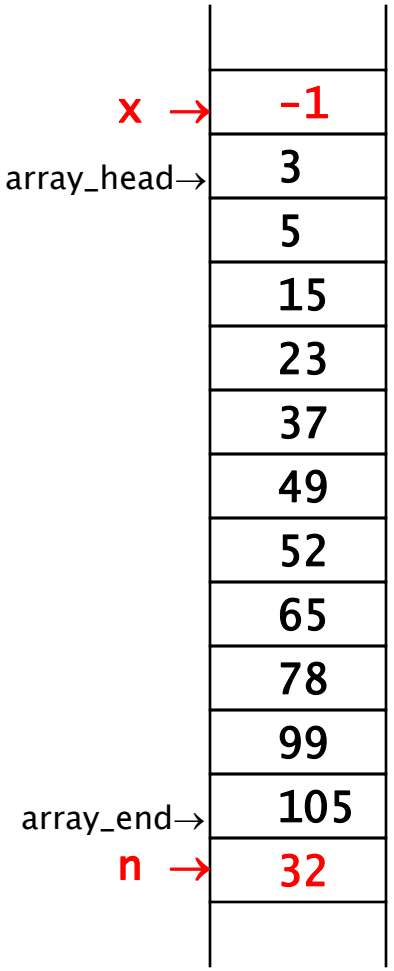
例：从键盘接收十六进制数并存入 BX

```
            ……
            mov   bx, 0
newchar:    mov   ah, 1
            int   21h
            sub   al, 30h
            jl    exit              ; <0 退出
            cmp   al, 10
            jl    add_to
            sub   al, 27h           ; ‘a’~‘f’
            cmp   al, 0ah
            jl    exit              ; < ‘a’  退出
            cmp   al, 10h
            jge   exit              ; >’ f’  退出
add_to:     mov   cl, 4
            shl   bx, cl
            mov   ah, 0
            add   bx, ax
            jmp   newchar
exit:       ……
```

例：将正数 n 插入一个已整序的正数字数组

```
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：将首地址为 a 的字数组从大到小排序

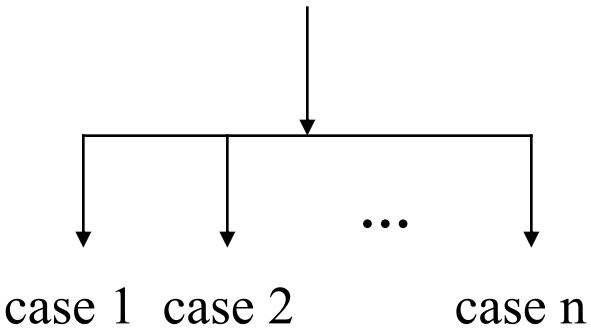
（起泡排序算法，多重循环）

100	100	100	100	100	100	100	100	189	256
30	78	99	99	99	99	99	189	256	189
78	99	78	78	78	78	189	256	100	100
99	30	30	66	66	189	256	99	99	99
15	15	66	45	189	256	78	78	78	78
-1	66	45	189	256	66	66	66	66	66
66	45	189	256	45	45	45	45	45	45
45	189	256	30	30	30	30	30	30	30
189	256	15	15	15	15	15	15	15	15
256	-1	-1	-1	-1	-1	-1	-1	-1	-1

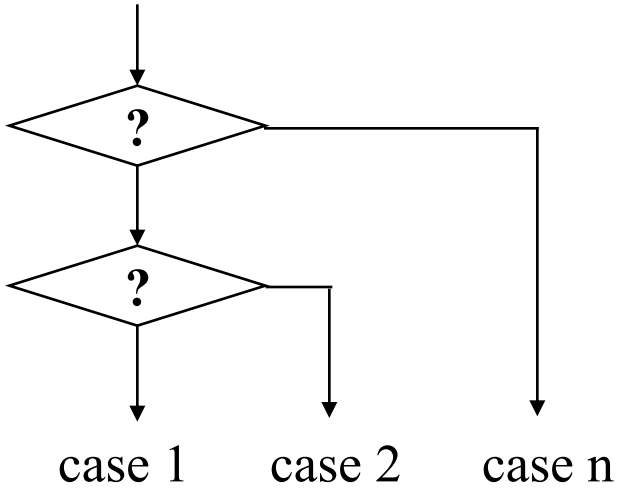
```
a    dw   100,30,78,99,15,-1,66,45,189,256

             ……
             mov    cx, 10
             dec    cx
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
             ……
```

2. 分支程序设计



CASE 结构



IF-THEN-ELSE 结构

(1) 逻辑尺控制

(2) 条件控制

(3) 地址跳跃表（值与地址有对应关系的表）

例：有数组 x(x1,x2,……,x10) 和 y(y1,y2,……,y10)，
编程计算 z(z1,z2,……,z10)

```
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
z10= x10 + y10
```

逻辑尺：0 0 1 1 0 1 1 1 0 0

1 减法

0 加法

```
            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   00dch

            ……
            mov    bx, 0
            mov    cx, 10
            mov    dx, logic_rule
next:       mov    ax, x[bx]
            shr    dx, 1
            jc     subtract
            add    ax, y[bx]
            jmp    short  result          ; 向前引用
subtract:
            sub    ax, y[bx]
result:     mov    z[bx], ax
            add    bx, 2
            loop   next
            ……
```

例：折半查找算法

```
data   segment
       array      dw   12,11,22,33,44,55,66,
                       77,88,99,111,222,333
       number     dw   55
       low_idx    dw   ?
       high_idx   dw   ?
data   ends
```

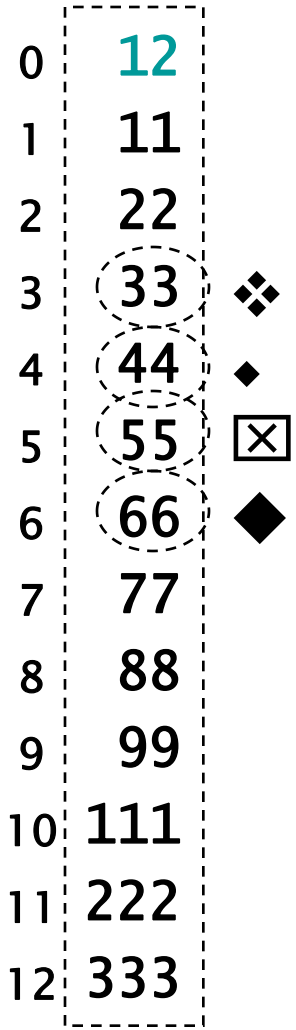
算法1

```
        ……
        lea   di, array
        mov   ax, number

        cmp   ax, [di+2]    ; (ax)与第一个元素比较
        ja    chk_last
        lea   si, [di+2]    ; si存放位置
        je    exit          ; (ax)=第一个元素,找到退出
        stc                 ; cf=1 查找失败
        jmp   exit          ; (ax)<第一个元素,未找到退出
chk_last:
        mov   si, [di]
        shl   si, 1
        add   si, di
        cmp   ax, [si]      ; (ax)与最后一个元素比较
        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
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:
    ……
```



(ax)=55

low_idx	high_idx	
1	12	
1	5	
4	5	(si)=0ah
5	5	cf=0

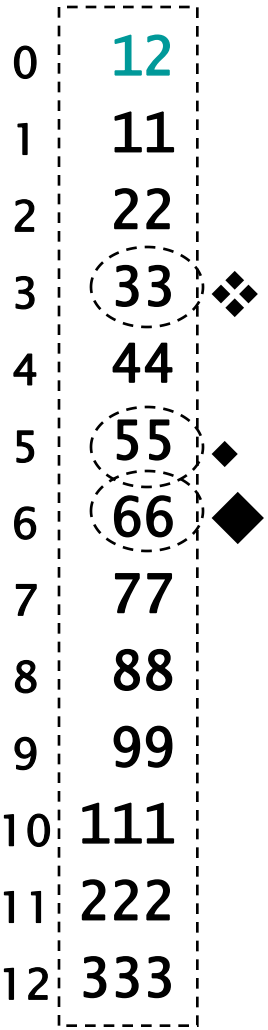
(ax)=90

low_idx	high_idx	
1	12	
7	12	
7	8	(si)=10h
8	8	
9	8	cf=1

算法2

```
search:
    mov  si, [di]
even_idx:
    test si, 1
    jz   add_idx
    inc  si
add_idx:
    add  di, si
compare:
    cmp  ax, [di]
    je   all_done
    ja   higher
    cmp  si, 2
    jne  idx_ok
no_match:
    stc
    jmp  exit
```

```
idx_ok:
    shr  si, 1
    test si, 1
    jz   sub_idx
    inc  si
sub_idx:
    sub  di, si
    jmp  short compare
higher:
    cmp  si, 2
    je   no_match
    shr  si, 1
    jmp  short even_idx
all_done:
    mov  si, di   ; si存放位置
exit:
    ……
```



(ax)=55

di	si	
		(si)=0ah
12	12	
6	6	(di)=0ah
10	4	cf=0

(ax)=90

di	si	
12	12	
18	6	(si)=2
14	4	(di)=10h
16	2	cf=1

例：根据 AL 寄存器中哪一位为 1（从低位到高位），
把程序转移到 8 个不同的程序分支

```
branch_table  dw  routine1
              dw  routine2
              dw  routine3
              dw  routine4
              dw  routine5
              dw  routine6
              dw  routine7
              dw  routine8
```

（寄存器间接寻址）

```
        ……
        cmp   al, 0
        je    continue
        lea   bx, branch_table
L:      shr   al, 1                    ;逻辑右移
        jnb   add1                     ;jnb=jnc
        jmp   word ptr[bx]             ;段内间接转移
add1:   add   bx, type branch_table
        jmp   L
continue:
        ……
routine1:
        ……
routine2:
        ……
```

（寄存器相对寻址）

```
        ……
        cmp   al, 0
        je    continue
        mov   si, 0
L:      shr   al, 1                    ;逻辑右移
        jnb   add1                     ;jnb=jnc
        jmp   branch_table[si]         ;段内间接转移
add1:
        add   si, type branch_table
        jmp   L
continue:
        ……
routine1:
        ……
routine2:
        ……
```

（基址变址寻址）

```
        ……
        cmp   al, 0
        je    continue
        lea   bx, branch_table
        mov   si, 7 * type branch_table
        mov   cx, 8
L:      shl   al, 1                          ;逻辑左移
        jnb   sub1                           ;jnb=jnc
        jmp   word ptr [bx][si]              ;段内间接转移
sub1:   sub   si, type branch_table
        loop   L
continue:
        ……
routine1:
        ……
routine2:
        ……
```