

第 5 章

基本汇编语言程序设计

教学重点

综合应用指令和伪指令，本章从程序结构角度展开程序设计，重点掌握：

- ✓ 分支结构程序设计
- ✓ 循环结构程序设计



基本程序结构



一、顺序程序

二、分支程序

三、循环程序

四、子程序



一、顺序程序设计

- 顺序程序完全按指令书写的前后顺序执行每一条指令，是最基本、最常见的程序结构



例1 计算



例2 移位



例3 代码转换



例1 计算 $w=x+y+z$

```
.model small  
.stack  
.data  
X    dw 5  
Y    dw 6  
Z    dw 7  
W    dw ?
```

```
.code  
.startup  
mov ax, X  
add ax, Y  
add ax, Z  
mov W, ax  
.exit 0  
end
```

例2 移位：64位数据左移8位

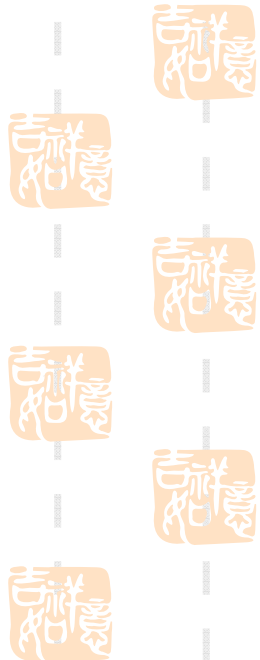
12 34 56 78 87 65 43 21h

移位后

34 56 78 87 65 43 21 00h

64位数据左移8位

qvar[0]	21	00
qvar[1]	43	
qvar[2]	65	
qvar[3]	87	
qvar[4]	78	
qvar[5]	56	
qvar[6]	34	
qvar[7]	12	



例2 移位：64位数据左移8位
- 1/2

.data

qvar dq 1234567887654321h

.code

mov al, byte ptr qvar[6]

mov byte ptr qvar[7], al

mov al, byte ptr qvar[5]

mov byte ptr qvar[6], al

mov al, byte ptr qvar[4]

mov byte ptr qvar[5], al

mov al, byte ptr qvar[3]

mov byte ptr qvar[4], al


```
mov al,byte ptr qvar[2]
mov byte ptr qvar[3],al
mov al,byte ptr qvar[1]
mov byte ptr qvar[2],al
mov al,byte ptr qvar[0]
mov byte ptr qvar[1],al
mov byte ptr qvar[0],0
```

例2 移位 - 2/2

12 34 56 78 87 65 43 21h

移位后

34 56 78 87 65 43 21 00h

例3 代码转换

查表法，实现一位16进制数转换为ASCII码显示

ASCII db 30h, 31h, 32h, 33h, 34h, 35h

db 36h, 37h, 38h, 39h ; 0~9的ASCII码

db 41h, 42h, 43h, 44h, 45h, 46h

; A~F的ASCII码

hex db 0bh

; 任意设定了一个待转换的一位16进制数

例3 代码转换 - 1/2

；查表法，实现一位16进制数转换为ASCII码显示

```
.model small
```

```
.stack 256
```

```
.data
```

ASCII db 30h, 31h, 32h, 33h, 34h, 35h

db 36h, 37h, 38h, 39h ; 0~9的ASCII码

db 41h, 42h, 43h, 44h, 45h, 46h

; A~F的ASCII码

hex db 0bh

; 任意设定了一个待转换的一位16进制数

例题 代码转换 - 2/2

.code

.startup

mov bx, offset ASCII ;BX指向ASCII码表

mov al, hex

;AL取得一位16进制数，正是ASCII码表中位移

and al, 0fh ;只有低4位是有效的，高4位清0

xlat ;换码：AL←DS:[BX + AL]

mov dl, al ;入口参数：DL←AL

mov ah, 2 ;02号DOS功能调用

int 21h ;显示一个ASCII码字符

.exit 0

end

基本程序结构



一、顺序程序

二、分支程序

三、循环程序

四、子程序



二、分支程序设计

- 分支程序根据条件是真或假决定执行与否
- 判断的条件是各种指令，如CMP、TEST等执行后形成的状态标志
- 转移指令Jcc和JMP可以实现分支控制

二、分支程序设计

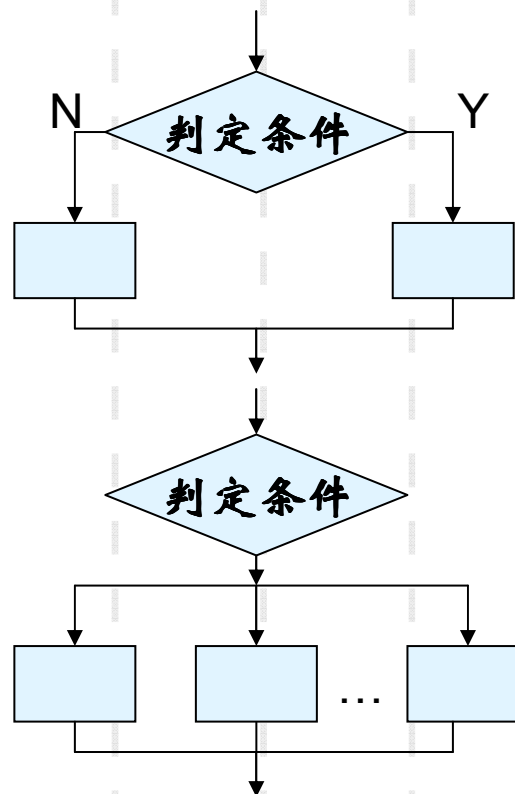
- 分支程序特点：运行方向是向前的，在某一种特定条件下，只能执行多个分支中的一个分支。



if then else

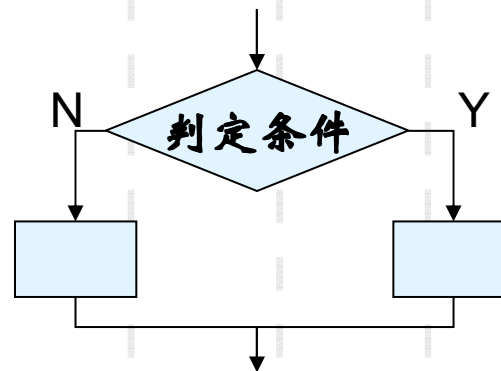


case



1、 if then else 结构

- 条件成立执行一个分支语句体，否则执行另一分支语句体；注意选择正确的条件转移指令和转移目标地址



例1 求AX的绝对值

; 计算AX的绝对值

例1 求绝对值

; 计算AX的绝对值

 **Bad**

cmp ax, 0

j1 yesneg ;分支条件: $AX < 0$

jmp nonneg

yesneg: neg ax ;条件不满足, 求补

nonneg: mov result, ax ;条件满足

 **Good**

cmp ax, 0

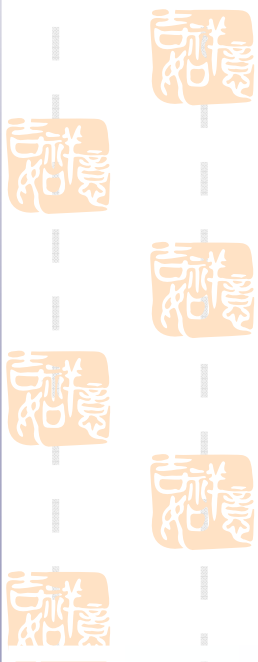
jns nonneg ;分支条件: $AX \geq 0$

neg ax ;条件不满足, 求补

nonneg: mov result, ax ;条件满足

例2 无符号数除以2

将AX中存放的无符号数除以2，如果是奇数，则加1后除以2



例2 无符号数除以2

；将AX中存放的无符号数除以2，如果是奇数，则加1后除以2

test ax, 01h ；测试AX最低位

jz even ；最低位为0：AX为偶数

add ax, 1

；最低位为1：AX为奇数，需要加1

even: rcr ax, 1 ；AX←AX÷2

例3 显示BX最高位 (1)

对比

```
shl bx, 1      ;BX最高位移入CF
jc one         ;CF = 1, 即最高位为1, 转移
mov dl, '0'
;CF = 0, 即最高位为0, DL ← '0'
jmp two        ;一定要跳过另一个分支体
one: mov dl, '1' ;DL ← '1'
two: mov ah, 2
int 21h        ;显示
```

例3 显示BX最高位 (2)

对比

```
shl bx, 1          ;BX最高位移入CF
jnc one            ;CF = 0, 即最高位为0, 转移
mov dl, '1'
;CF = 1, 即最高位为1, DL ← '1'
jmp two            ;一定要跳过另一个分支体
one: mov dl, '0'    ;DL ← '0'
two:  mov ah, 2
      int 21h       ;显示
```

例3 显示BX最高位 (3)

```
mov dl, '0'      ;DL←'0'
shl bx, 1         ;BX最高位移入CF
jnc two          ;CF=0, 最高位为0, 转移
mov dl, '1'      ;CF=1, 最高位为1, DL←'1'
two: mov ah, 2
int 21h          ;显示
```

 编写分支程序，需留心分支的开始和结束

例4 判断有无实根 - 1/2

; 有实根: 标记 tag=1; 无实根: 标记 tag=0

.startup

mov al, _b

imul al

mov bx, ax

; BX 中为 b^2

mov al, _a

imul _c

mov cx, 4

imul cx

; AX 中为 4ac (DX 无有效数据)

例4 判断有无实根 - 2/2

`cmp bx, ax` ;比较二者大小

`jge yes` ;条件满足?

`mov tag, 0`

;第一分支体: 条件不满足, $\text{tag} \leftarrow 0$

`jmp done` ;跳过第二个分支体

`yes:` `mov tag, 1`

;第二分支体: 条件满足, $\text{tag} \leftarrow 1$

`done:` `.exit 0`

例5 判断AL中的字母

; 寄存器AL中是字母Y或y, 则令AH=0; 否则令AH=-1

cmp al, 'Y' ; AL是大写Y否?

jz next ; 是, 转移

cmp al, 'y' ; AL是小写y否?

jz next ; 是, 转移

mov ah, -1 ; 不是Y或y, 则AH=-1, 结束

jmp done ; 一定要跳过另一个分支体

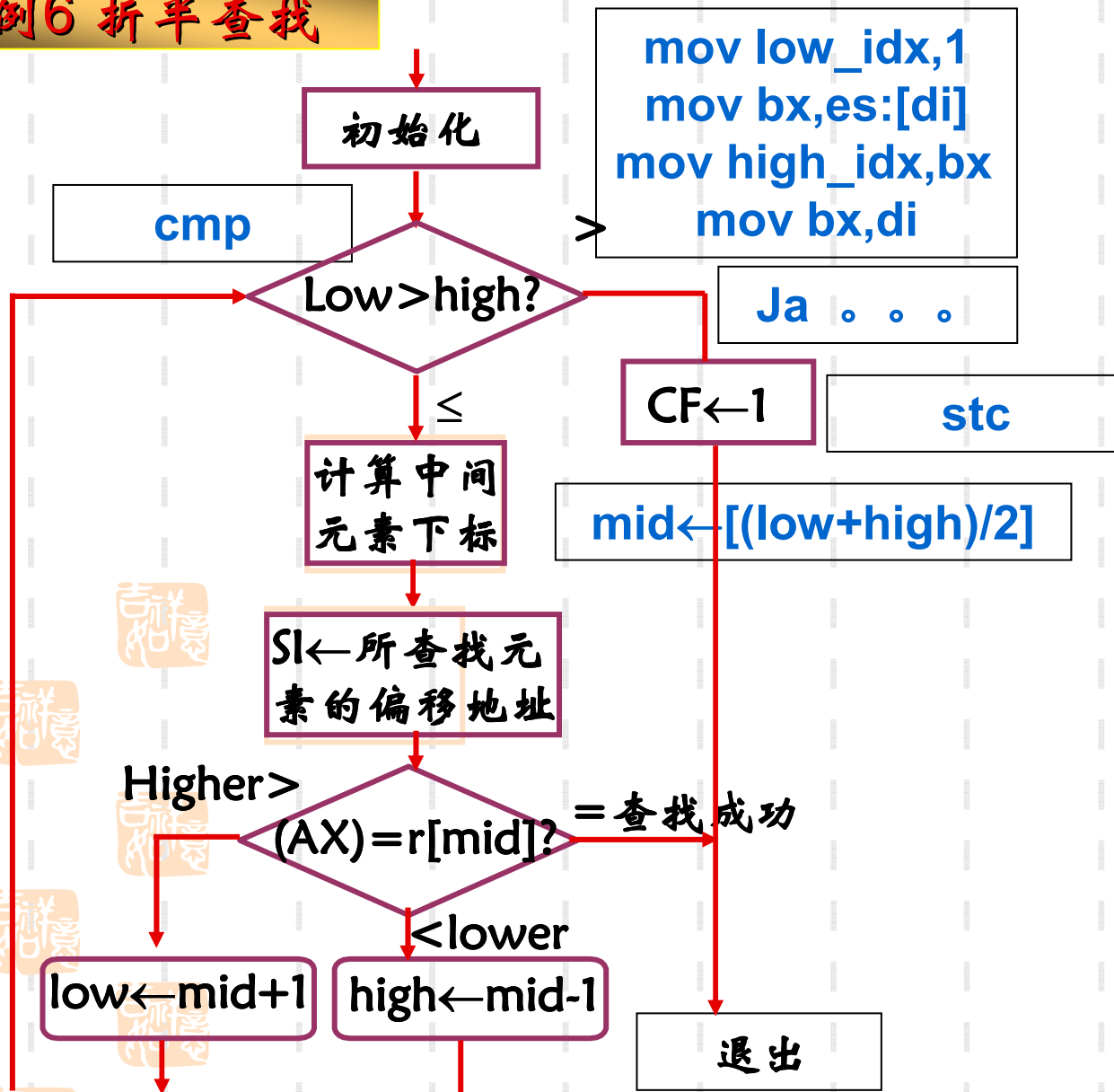
next: mov ah, 0 ; 是Y或y, 则AH=0, 结束

done: ...

例6 折半查找

在附加段中，有一个按从小到大顺序排列的无符号数组，其首地址存放在DI寄存器中，数组中的第一个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

例6 折半查找



例6 折半查找P177 5.9 -1/5

dseg segment ; 定义数据段

low_idx dw ?

high_idx dw ?

dseg ends

cseg segment ; 定义数据段

b_search proc near

assume cs:cseg, ds:dseg, es:dseg

push ds

push ax

mov ax, dseg

mov ds, ax

mov es, ax

pop ax

例6 折半查找 - 2/5

```
    cmp ax, es:[di+2]    ; search value<or=first el.
    ja  chk_last        ;no, go check last el.
    lea si, es:[di+2]    ;yes, fetch addr or first el.
    je  exit            ;if value=1st el., exit
    stc                 ;if value<1st el., set CF
    jmp exit            ;and then exit
chk_last:
    mov si, es:[di]      ;point to last el.
    shl si, 1
    add si, di
    cmp ax, es:[si]      ; search value>or=last el.
    jb  search           ;no, go search list
    je  exit            ;yes, exit if value=last el.
    stc                 ;if value>last el., set CF
    jmp exit            ;and then exit
```



例6 折半查找 - 3/5

search:

mov low_idx, 1 ;fetch index

mov bx, es:[di]

mov high_idx, bx

mov bx, di

mid: mov cx, low_idx ;calculate middle point

mov dx, high_idx

cmp cx, dx

ja no_match

add cx, dx

shr cx, 1

mov si, cx

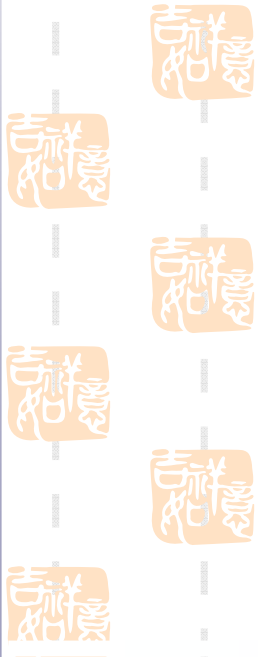
shl si, 1 ;calculate next search addr

例6 折半查找 - 4/5

```
compare: cmp    ax, es:[bx+si] ;search value found?
          je     exit           ;if so, exit
          ja     higher        ;otherwise, find correct half
          dec    cx
          mov     high_idx, cx
          jmp     mid
higher:   inc     cx
          mov     low_idx, cx
          jmp     mid
no_match: stc                  ;set CF
```

例6 折半查找 - 5/5

```
exit:    pop    ds
         ret
b_search endp
cseg     ends
         end
```



2、case 结构

图示

- 多个条件对应各自的分支语句体，哪个条件成立就转入相应分支体执行。多分支可以化解为if then else结构的组合，例如：

cmp ah, 0

jz function0

;ah = 0, 转向function0

cmp ah, 1

jz function1

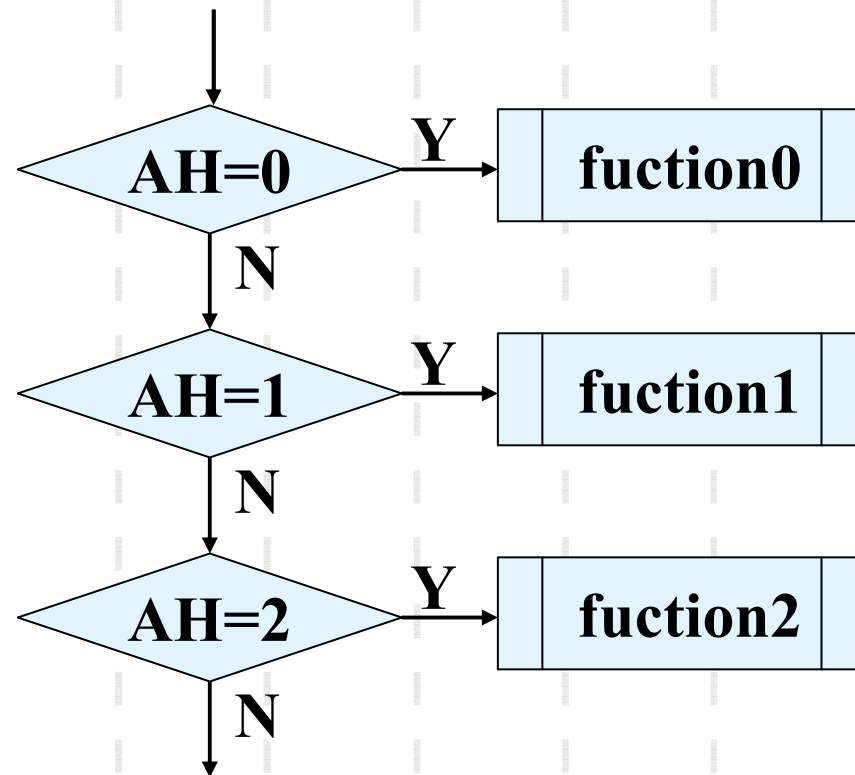
;ah = 1, 转向function1

cmp ah, 2

jz function2

;ah = 2, 转向function2

case 结构



地址表形成多分支

- 需要在数据段事先安排一个按顺序排列的转移地址表
- 输入的数字作为偏移量。因为有2个字节16位偏移地址，所以偏移量需要乘2
- 关键是要理解间接寻址方式JMP指令

Table dw disp1, disp2, disp3, disp4, ...

地址表	分支1地址	分支2地址	...
-----	-------	-------	-----

转移到分支地址: **JMP Table[BX]** ; BX存放偏移量

例7 数据段 - 1/3

```
.data  
msg      db 'Input number (1~8):', 0dh, 0ah, '$'  
msg1     db 'Chapter 1 : ...', 0dh, 0ah, '$'  
msg2     db 'Chapter 2 : ...', 0dh, 0ah, '$\  
...  
msg8     db 'Chapter 8 : ... ', 0dh, 0ah, '$'  
table    dw disp1, disp2, disp3, disp4  
          dw disp5, disp6, disp7, disp8
```

;取得各个标号的偏移地址

此处等同于 offset disp1

例7 代码段 - 2/3

```
start1:  mov dx, offset msg    ;提示输入数字
         mov ah, 9
         int 21h
         mov ah, 1            ;等待按键
         int 21h
         cmp al, '1'          ;数字 < 1?
         jb start1
         cmp al, '8'          ;数字 > 8?
         ja start1
         and ax, 000fh         ;将ASCII码转换成数字
```

例7 代码段 - 3/3

```
dec ax
shl ax, 1      ; 等效于 add ax, ax
mov bx, ax
jmp table[bx]
```

; (段内) 间接转移: $IP \leftarrow [table+bx]$

```
start2:  mov ah, 9
```

```
         int 21h
```

```
         .exit 0
```

```
displ:   mov dx, offset msg1      ; 处理程序1
```

```
         jmp start2
```

```
         ...
```

例8 用变址寻址方式实现跳跃表-1/4

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个不同的程序分支中去。

```
branch_addresses segment
branch_table  dw  routine_1
                dw  routine_2
                dw  routine_3
                dw  routine_4
                dw  routine_5
                dw  routine_6
                dw  routine_7
                dw  routine_8
branch_addresses ends
```



例8 用变址寻址方式实现跳跃表 - 2/4

```
procedure_select segment
main proc far
assume cs:procedure_select, ds:branch_addresses
start:
    push ds
    sub bx, bx
    push bx
    mov bx, branch_addresses
    mov ds, bx
```


例8 用变址寻址方式实现跳跃表 - 3/4

```
    cmp    al, 0
    je     continue_main_line
    mov    si, 0
1:    shr    al, 1
    jnc    not_yet
    jmp    branch_table[si]
not_yet: add    si, type branch_table
        jmp    1
continue_main_line:
.....
```

例8 用变址寻址方式实现跳跃表-4/4

```
routine_1:
```

```
.....
```

```
routine_2:
```

```
.....
```



```
ret
```



```
main endp
```



```
procedure_select ends  
end start
```



例8 用间接寻址方式实现跳跃表

```
    cmp    al, 0
    je     continue_main_line
    lea     bx, branch_table
1:    shr    al, 1
    jnb     not_yet
    jmp     word ptr [bx]
not_yet: add    bx, type branch_table
          jmp    1
continue_main_line:
```

.....

例8 用基址变址寻址方式实现跳跃表

```
    cmp    al, 0
    je     continue_main_line
    lea     bx, branch_table
    mov     si, 7*type branch_table
    mov     cx, 8
1:    shl    al, 1
    jnb     not_yet
    jmp     word ptr[bx][si]
not_yet:  sub     si, type branch_table
          jmp     1
continue_main_line:
```

基本程序结构



一、顺序程序

二、分支程序

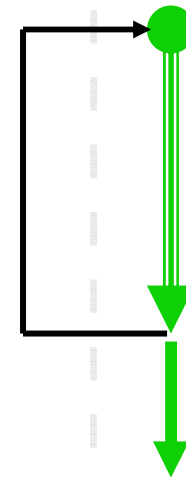
三、循环程序

四、子程序

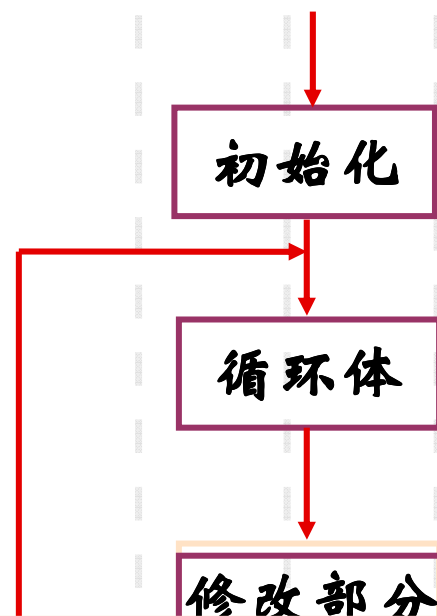


吉祥慶
真或假

-



循环结构



循环的初始状态

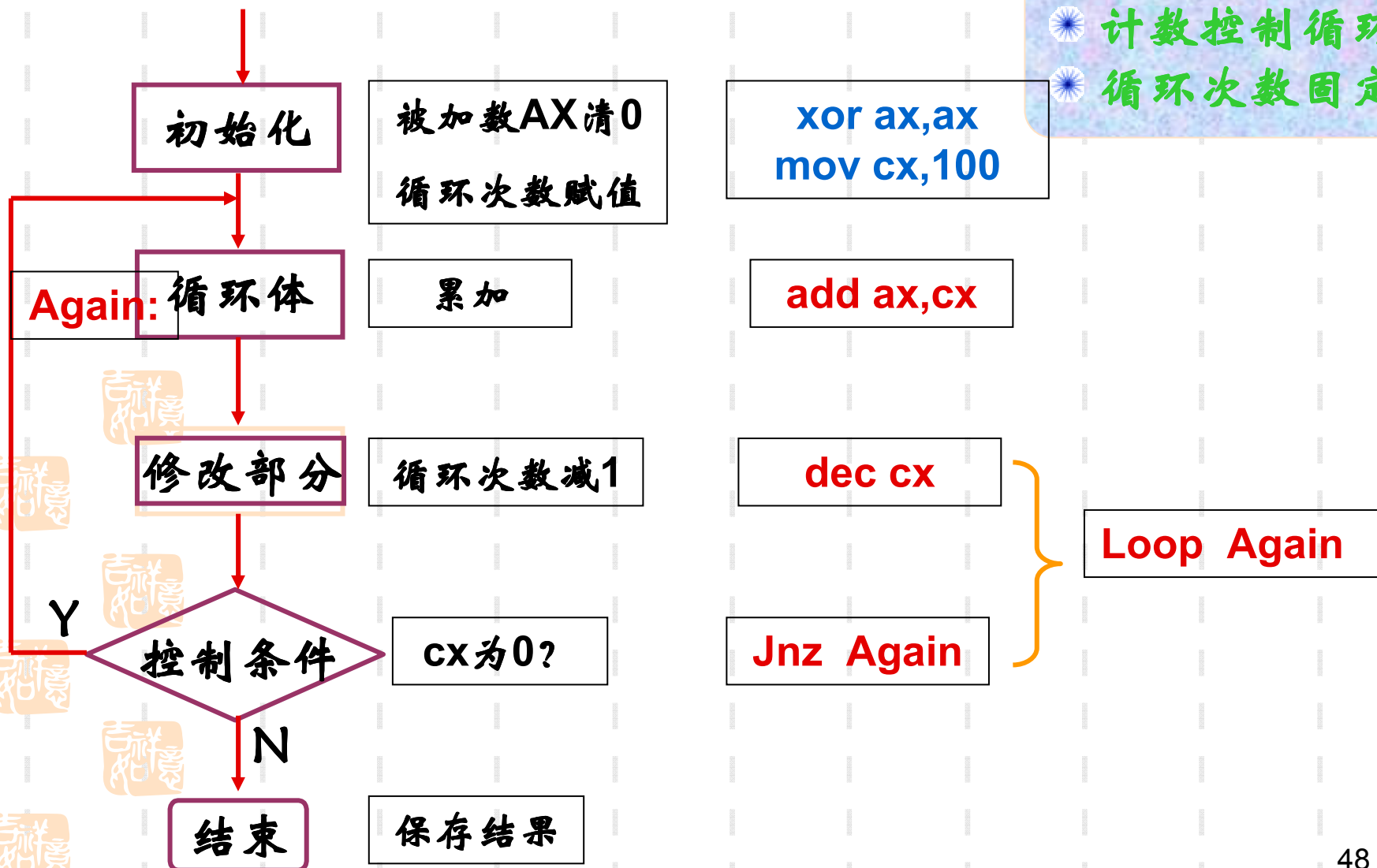
循环的工作部分
及修改部分

注意

循环控制条件是循环程序设计的关键，必须合理选择，同时必须仔细考虑边界情况出现的可能性

结束

例1 求和100, 99, ..., 2, 1倒序累加1/2



sum

```
.model small
.stack
.data
dw ?
.code
.startup
xor ax, ax
mov cx, 100
again: add ax, cx
loop again
mov sum, ax
.exit 0
end
```

例1 求和100, 99, ..., 2, 1倒序累加2/2

计数控制循环
循环次数固定

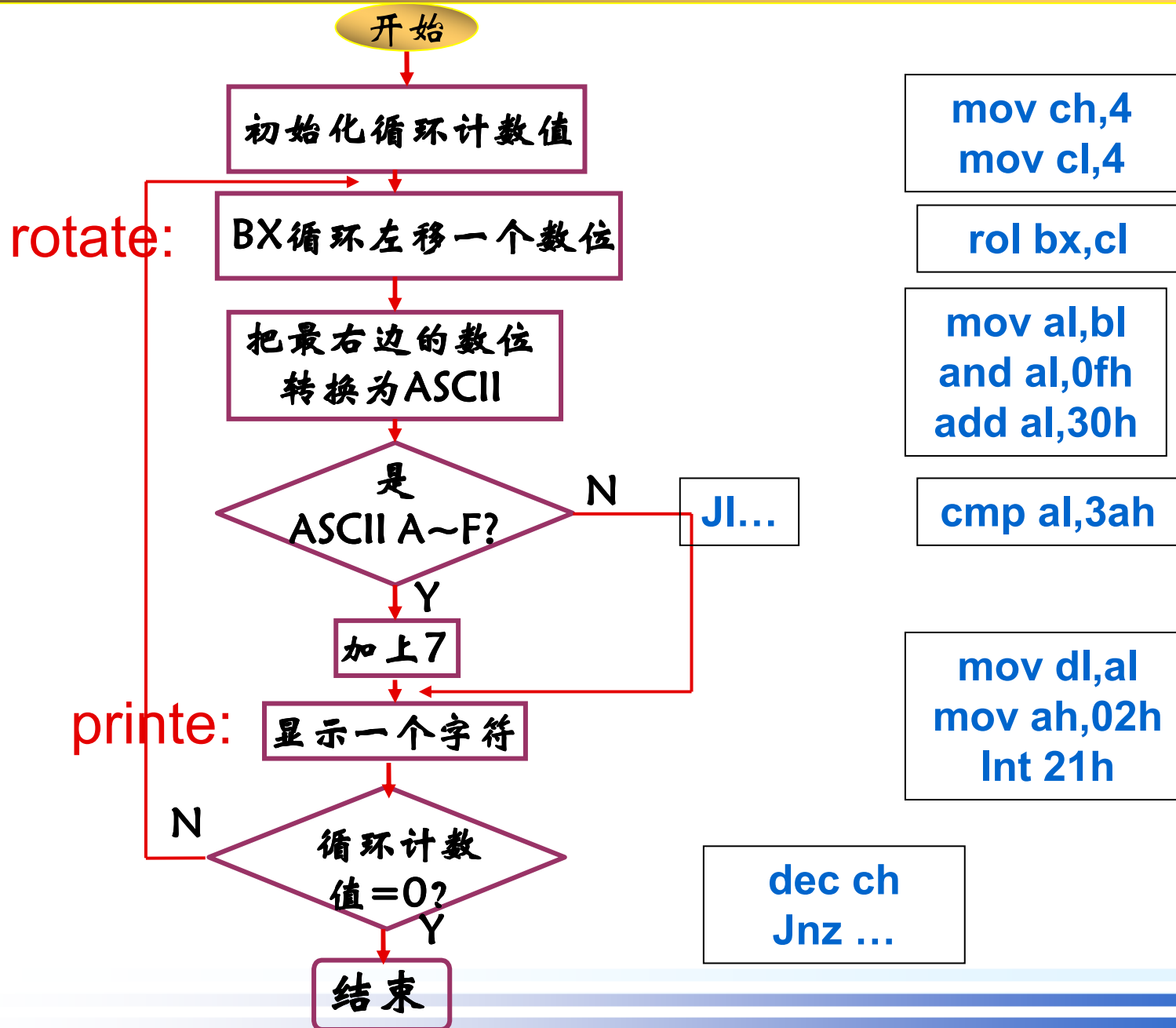
;被加数AX清0

;从100, 99, ..., 2, 1倒序累加

;将累加和送入指定单元

例2 二进制到十六进制转换P161 例5.1 1/2

编制程序把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



例2 二进制到 十六进制转换 2/2

计数控制循环
循环次数固定

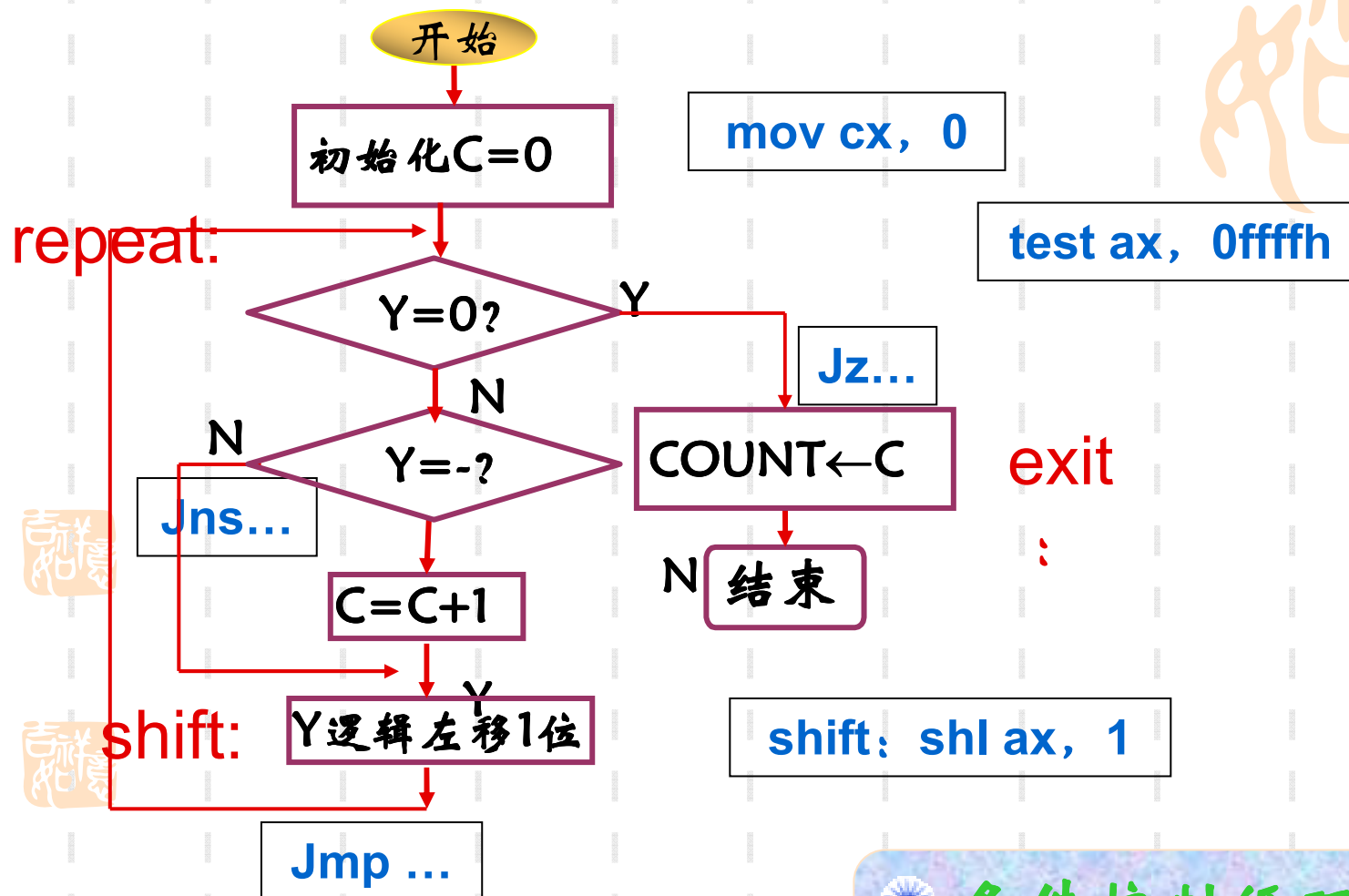
..... 程序初始化

```
mov ch, 4
rotate: mov cl, 4
        rol bx, cl
        mov al, bl
        and al, 0fh
        add al, 30h
        cmp al, 3ah
        jl printit
        add al, 7h
printit: mov dl, al
        mov ah, 2
        int 21h
        dec ch
        jnz rotate
```

..... 程序结束

例3 统计Y中1的个数 P163 例5.2 1/2

在
ADDR
单元中
存放着
数Y的
地址，
试编制
一程序
把Y中
1的个
数存入
COUNT
单元中



条件控制循环
利用标志退出

```

data segment
    addr    dw    number
    number  dw    Y
    cont    dw    ?
data ends
code  segment
    .....
        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
    .....

```

例3 统计Y中1的个数2/2

数Y的地址存放在ADDR中

- 条件控制循环
- 利用标志退出

例4 一维数组运算P167例5.5 - 1/4

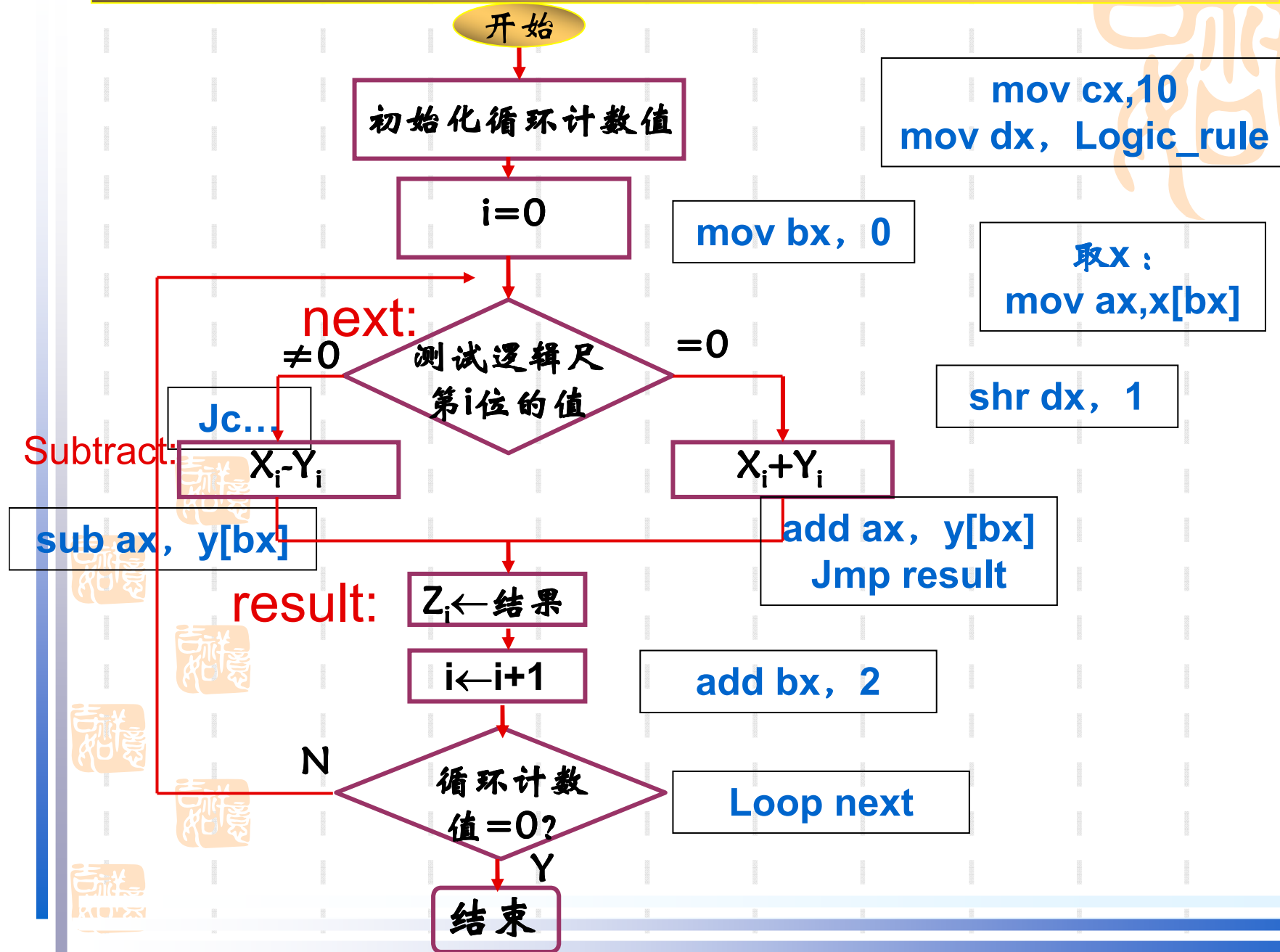
设有数组X和Y。X中有 X_1, \dots, X_{10} ；Y中有 Y_1, \dots, Y_{10} 。编制程序计算

$$\begin{array}{lll} Z_1 = X_1 + Y_1 & Z_5 = X_5 - Y_5 & Z_8 = X_8 - Y_8 \\ Z_2 = X_2 + Y_2 & Z_6 = X_6 + Y_6 & Z_9 = X_9 + Y_9 \\ Z_3 = X_3 - Y_3 & Z_7 = X_7 - Y_7 & Z_{10} = X_{10} + Y_{10} \\ Z_4 = X_4 - Y_4 & & \end{array}$$

❁ 循环次数已知，但每次操作不同，用逻辑尺控制分支。

用一个二进制数的两个状态来标志两个分支程序段，将多位二进制数排列在一起，以表示多次循环时执行不同程序段的情况。0000000011011100

例4一维数组运算P167例5.5 - 2/4



例4 一维数组运算P167例5.5-3/4

data segment

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

data ends

prognam segment

..... 程序初始化

mov bx, 0

mov cx, 10

mov dx, logic_rule

例4 一维数组运算P167例5.5 - 4/4

```
next:  mov  ax, x[bx]
        shr  dx, 1
        jc   subtract
        add  ax, y[bx]
        jmp  short result
```

subtract:

```
        sub  ax, y[bx]
result:  mov  z[bx], ax
        add  bx, 2
        loop next
```

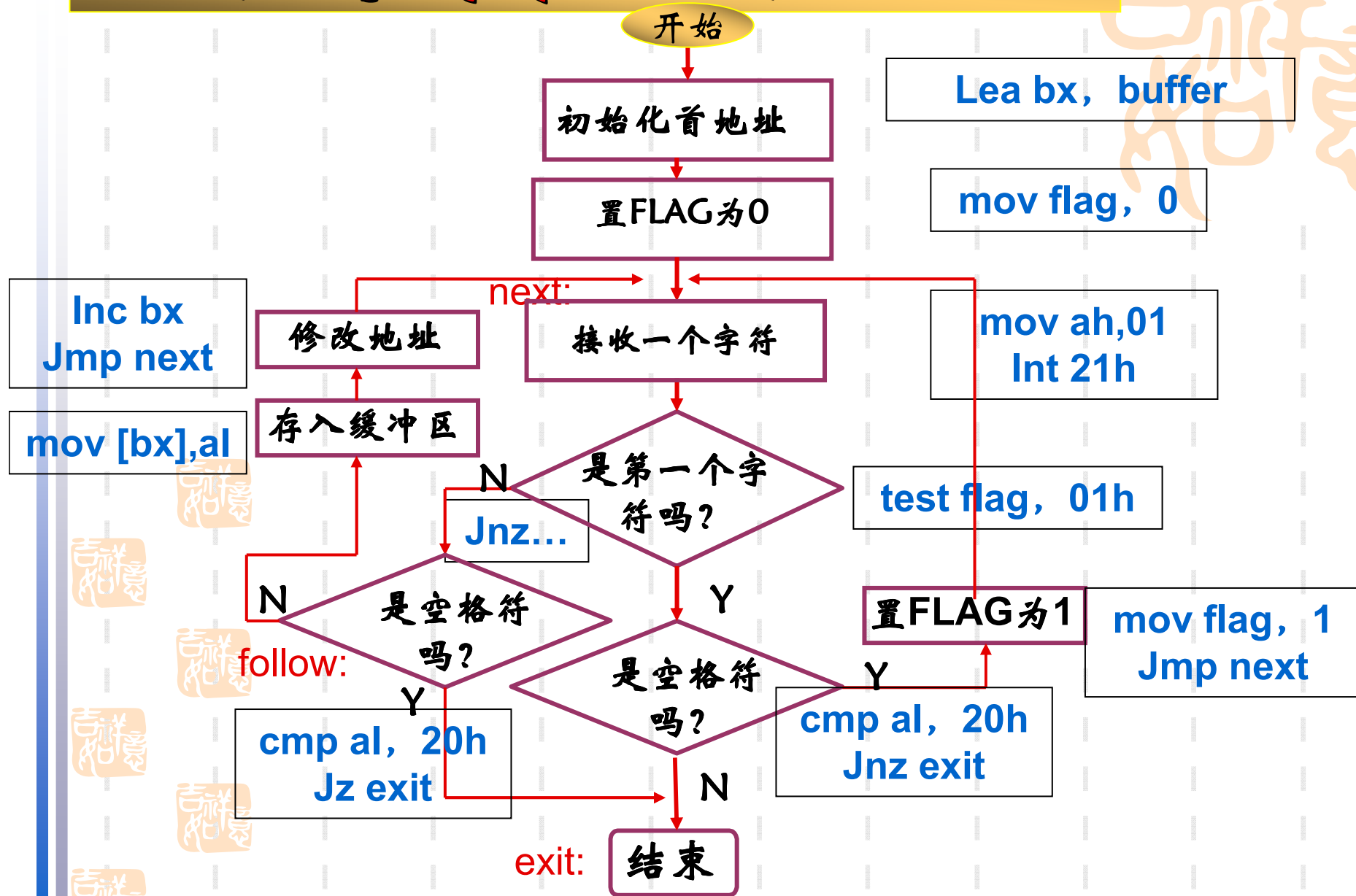
..... 程序结束



例5 键入字符P170 例5.6 - 1/4

试编制一程序：从键盘输入一行字符，要求第一个键入的字符必须是空格符，如不是，则退出程序；如是，则开始接收键入的字符并顺序存放在首地址为BUFFER的缓冲区中（空格符不存入），直到接收到第二个空格符时退出程序。

例5 键入字符P170 例5.6 2/4



例5 键入字符P170 例5.6 - 3/4

dataarea segment

buffer db 80dup(?)

flag db ?

dataarea ends

prognam segment

.....

程序初始化

lea bx, buffer

mov flag, 0

Next: mov ah, 01

int 21h

例5 键入字符P170 例5.6 - 4/4

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: ret

.....

程序结束

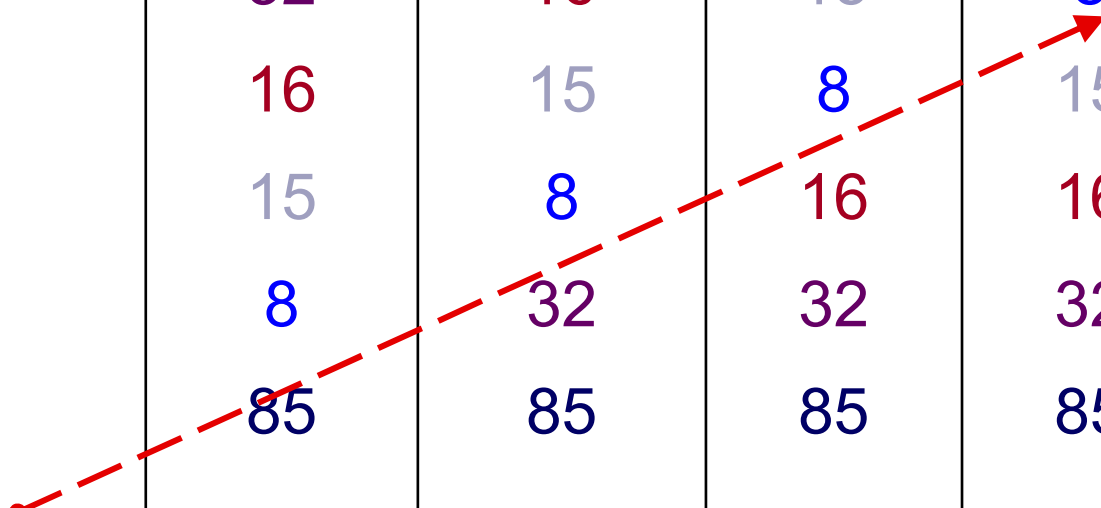
冒泡法

图示

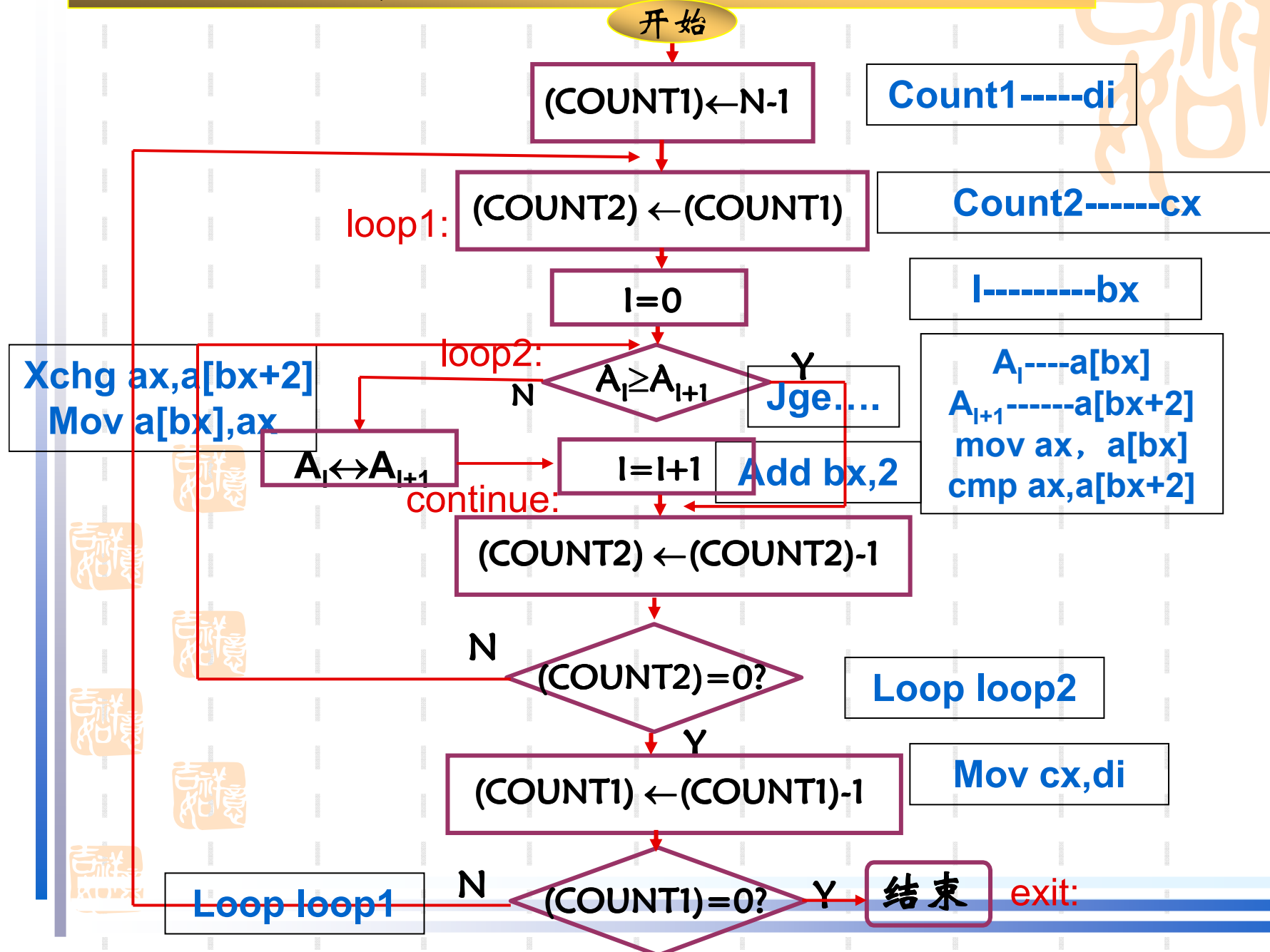
- “冒泡法”是一种排序算法，不是最优的算法，但它易于理解和实现
- 冒泡法从第一个元素开始，依次对相邻的两个元素进行比较，使前一个元素不大于后一个元素；将所有元素比较完之后，最大的元素排到了最后；然后，除掉最后一个元素之外的元素依上述方法再进行比较，得到次大的元素排在后面；如此重复，直至完成就实现元素从小到大的排序
这需要一个双重循环程序结构

冒泡法的排序过程

		比 较 遍 数			
		1	2	3	4
序号	数				
1	32	32	16	15	8
2	85	16	15	8	15
3	16	15	8	16	16
4	15	8	32	32	32
5	8	85	85	85	85



例6 冒泡法P172 例5.7 1/3



冒泡法 2/3

```
mov cx, count      ;CX←数组元素个数
dec cx             ;元素个数减1为外循环次数
loop1: mov di, cx   ;DX←内循环次数
        mov bx, 0
loop2:  mov ax, a[bx]      ;取前一个元素, a为数组名
        cmp ax, a[bx+2]   ;与后一个元素比较
        jge continue
        ;前一个不大于后一个元素, 则不进行交换
        xchg ax, a[bx+2] ;否则, 进行交换
        mov a[bx], ax
continue: add bx, 2      ;下一对元素
        loop loop2
        mov cx, di      ;内循环尾
        loop loop1      ;外循环尾
```

对比

例6 冒泡法 3/3

```
        mov cx, count
        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
```

```
        mov cx, count
        dec cx
outlp:   mov dx, cx
        mov bx, offset array
inlp:    mov al, [bx]
        cmp al, [bx+1]
        jna next
        xchg al, [bx+1]
        mov [bx], al
next:    inc bx
        dec dx
        jnz inlp
        loop outlp
```



计数控制双重循环

例7 剔除空格

；现有一个以\$结尾的字符串，要求剔除其中的空格



例7 剔除空格 - 1/2

; 现有一个以\$结尾的字符串，要求剔除其中的空格

```
.data
string db 'Let us have a try !', '$'

.code
.startup
mov si, offset string
outlp: cmp byte ptr [si], '$'
      ; 外循环，先判断后循环
      jz done      ; 为$结束
      cmp byte ptr [si], ' '
      ; 检测是否是空格
      jnz next     ; 不是空格继续循环
```



例7 剔除空格-2/2

条件控制双重循环

```
        mov di, si        ;是空格, 进入剔除空格分支
                                ;该分支是循环程序段
inlp:    inc di
        mov al, [di]      ;前移一个位置
        mov [di-1], al
        cmp byte ptr [di], '$'
                                ;内循环, 先循环后判断
        jnz inlp
        jmp outlp
next:    inc si            ;继续对后续字符进行处理
        jmp outlp
done:    .exit 0          ;结束
```

第5章 教学要求

1. 掌握基本程序结构——顺序结构、分支结构、循环结构及其汇编语言程序设计

2. 熟悉常见程序设计问题：

多精度运算、查表（查代码、特定值等）

ASCII、BCD及十六进制数据间的代码转换

数据范围判断（0~9、A~Z、a~z）

字母大小写转换；字符串传送、比较等操作

求最小最大值、数据求和、统计字符个数

作业：5.1 5.3 5.18 5.19 5.23



放鬆一下

