汇编语言程序设计

Assembly Language Programming

主讲：徐娟

计算机与信息学院　计算机系 分布式控制研究所

E-mail: xujuan@hfut.edu.cn,

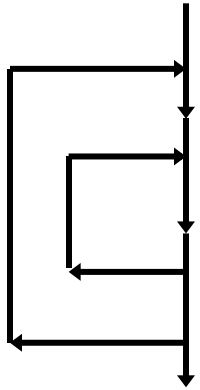
Mobile: 18055100485

第四章
基本汇编语言程序设计

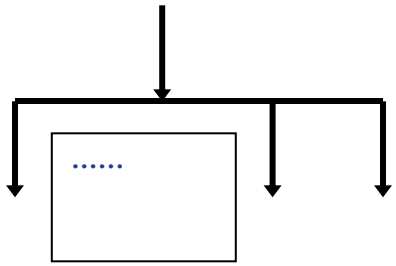
程序结构

顺序结构

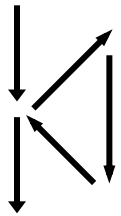
循环结构



分支结构



子程序结构



复合结构：多种程序结构的组合

4.1. 顺序程序设计

；查表法，实现一位16进制数转换为ASCII码显示

```
        .model small
        .stack
        .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
```

4.1. 顺序程序设计

```
.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
```

4.2 分支程序设计

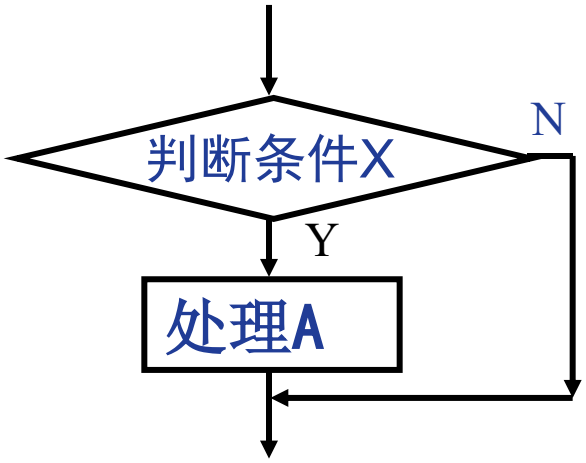
分支程序

- 使用条件转移指令来完成分支。一个JCC产生两分支。
- JCC在条件成立发生转移，条件不成立顺序执行。
- JMP不会产生分支。

分支程序基本结构

- 单分支、双分支、多分支

1 单分支结构




```
    JX  Next
    JMP  Done
Next:
    ; Handle A
Done：
; Switch has done
```

```
    JNX  Next
     ; Handle A
      Next:

; Switch has done
```

1 单分支结构

；计算AX的绝对值

Good

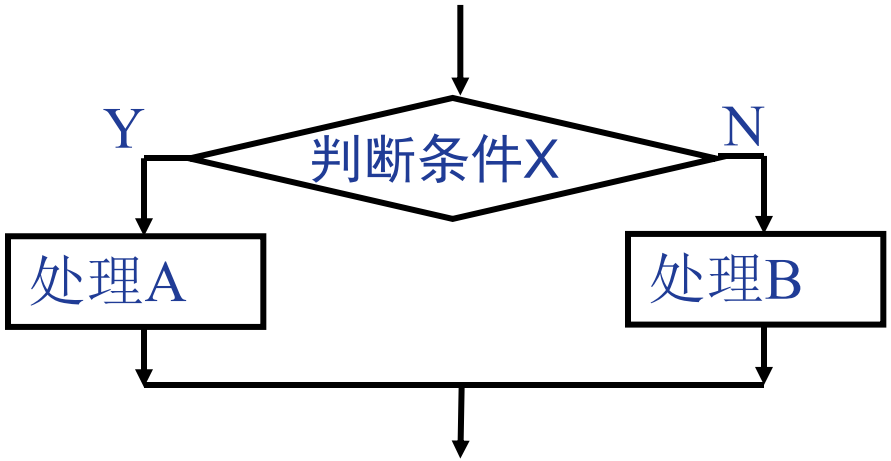
```
        cmp  ax,0
        jns  nonneg       ;分支条件：AX≥0
        neg  ax           ;条件不满足，求补
nonneg: mov  result,ax    ;条件满足
```

；计算AX的绝对值

Bad

```
        cmp  ax,0
        jl   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
```

2 双分支结构




```
        JX  Next
        ; Handle B
        JMP Done
Next:
         ;Handle A
Done:
        ; Switch has done
```

注意：顺序执行的分支语句体1不会自动跳过分支语句体2，所以分支语句体1最后必须要有JMP语句跳过分支体2

2 双分支结构

例2 显示BX最高位 P92

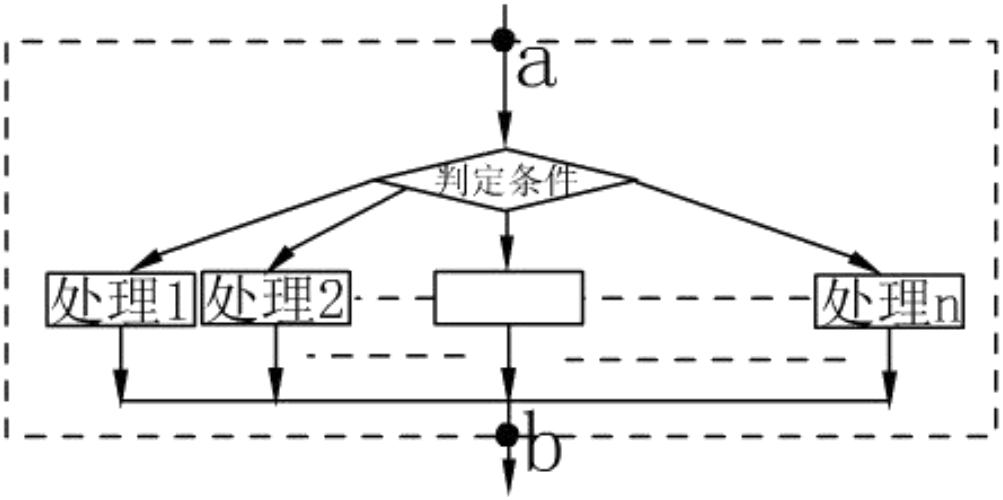
```
        shl bx,1            ;BX最高位移入CF
        jnc one             ;CF＝0，即最高位为0，转移
        mov dl,'1'
        ;CF＝1，即最高位为1，DL←'1'
        jmp two             ;一定要跳过另一个分支体
one:    mov dl,'0'          ;DL←'0'
two:    mov ah,2
        int 21h             ;显示
```

2 双分支结构

例3 判断有无实根 P93

```
        mov al,_b
        imul al              ;al=al*al
        mov bx,ax            ;BX中为b²
        mov al,_a
        imul _c
        mov cx,4
        imul cx              ;AX中为4ac（DX无有效数据?）
        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，tag←0
        jmp done             ;跳过第二个分支体
yes:    mov tag,1            ;第二分支体：条件满足，tag←1
done:       .exit 0
```

3 多分支结构



- **多分支程序处理方法：**
  - 1. 多条件转移指令实现（if … else if …else if …）
  - 2. 地址表 （Switch … Case…）

3 多分支结构

例4：在内存Score缓冲区中存放有100个学生的成绩数据，为无符号字节数。分别统计各个分数段的人数，分别存储在NOTPASSED 、PASSED、GOOD 、BETTER 、BEST。

```
    CMP SCORE[BX],90
    JB NEXT
    INC BEST                    ;if >= 90 , Best!
    JMP DONE
NEXT:
    CMP SCORE[BX],80            ;If got here, must <90!
    JB NEXT1
    INC BETTER                  ;if >=80 , Better
    JMP DONE
        …
```

3 多分支结构

Swich case

switch(表达式）

{

case:常量1： do sth1;break;

……

case:常量n： do sthn;break;

}

分析问题

多分支条件

转化为表达式

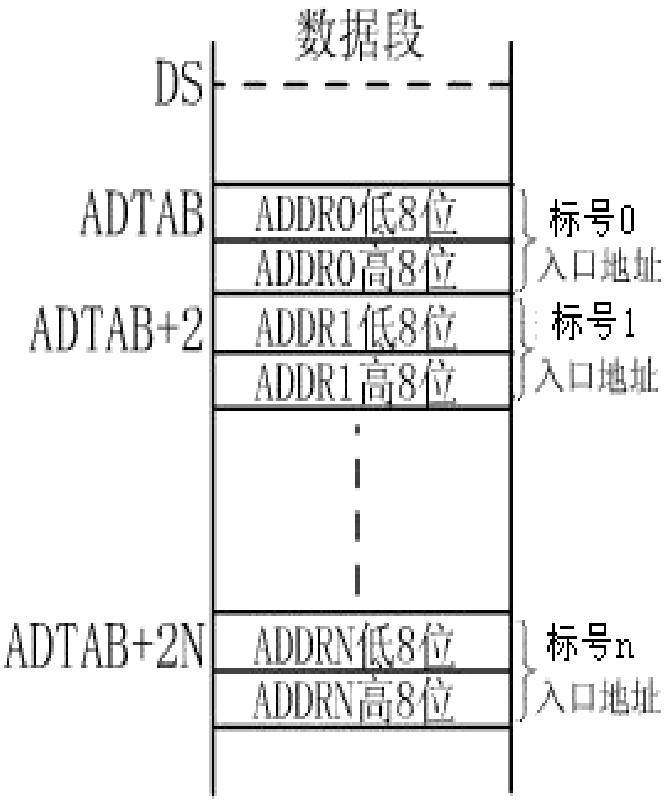
离散常量

地址表（Swich case）

- 设计分支条件，使第n个分支映射为数n
- 在存储器的数据段中定义一张入口地址表，把若干程序段的16位偏移地址按顺序放在地址表

```
AddressTable DW s1,s2,s3,……
```

程序段标号



- 间接寻址 JMP AddressTable[2*n]

根据条件转入n分支：

n号分支地址＝[入口地址表首地址 ＋ n×2]

3 多分支结构（地址表）

例5：根据输入的数字1－7，分别显示相应的英文星期名。

等同于 offset L4

```
Data segment
ADDRTABLE DW L1,L2,L3,L4, … ;
S1 DB 'MONDAY$'
S2 DB 'TUESDAY$'
S3 DB 'WEDNESDAY$'
S4 DB 'THURSDAY$'
;.....
Data ends
```

```
……
MOV AH,1
INT 21H
SUB AL,31H
SHL AL,1   ;AL×2
MOV AH,0
MOV BX,AX
JMP ADDRTABLE[BX]
L1: LEA DX,S1 ……
L2: LEA DX,S2 ……
```

例4.4 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

3 多分支结构（地址表）

例4.4 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码转换成数值
```

3 多分支结构（地址表）

例4.4 代码段-3/3

```
        dec ax          ; ax从0开始
        shl ax,1        ;等效于add ax,ax
        mov bx,ax
        jmp table[bx]   ;（段内）间接转移：IP←[table+bx]
start2: mov ah,9
        int 21h
        .exit 0
disp1:  mov dx,offset msg1      ;处理程序1
        jmp start2
disp2:
        ...
```

对应修改为 ret

Jmp word ptr table[bx]
或call table[bx]

循环程序的一般结构

初始化

- 建立循环计数器,例如：

  MOV CX, n

- 初始化地址指针,例如：

  LEA BX, Buffer

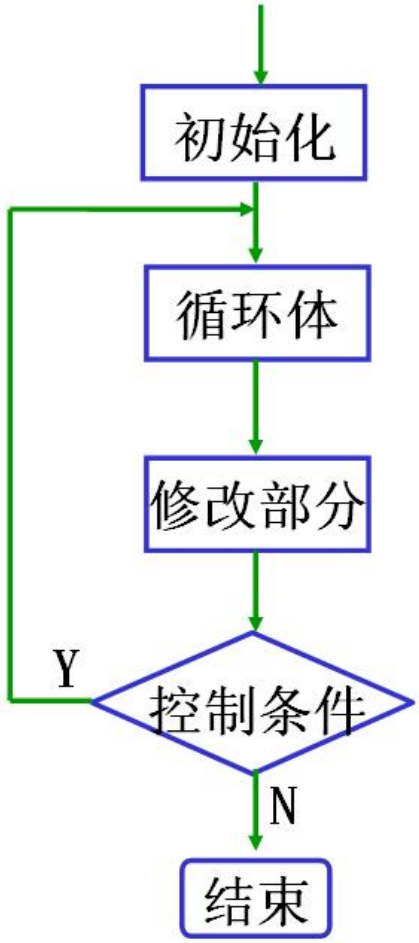
  MOV BX, offset Buffer

- 建立下标计数器,例如：

  MOV SI, 0

- 清空或设置某些寄存器,例如：

  MOV AX, 0

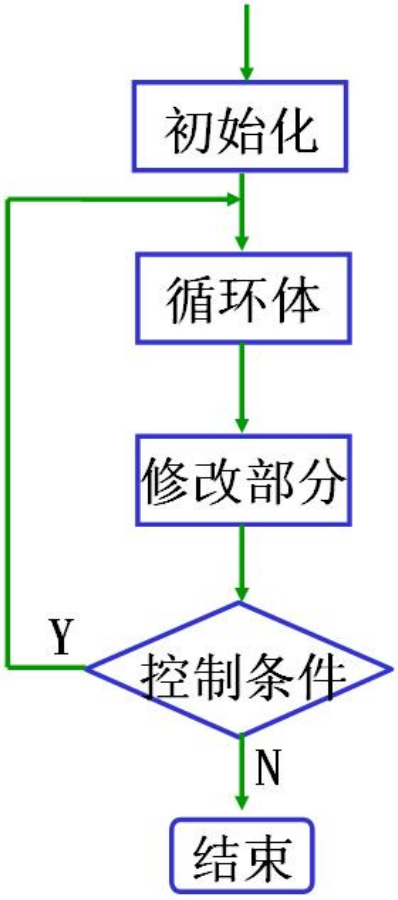


4.3 循环程序设计

- 循环体部分的编写
- 循环控制/触发下一次循环的代码
  - 对地址指针或者下标计数器进行加（注意步长）

    例如： INC SI; ADD BX, 2
  - 循环计数器减

    例如： LOOP AGAIN
- 循环退出的确定
  - 正常退出
    - 计数结束
  - 中途退出
    - 条件退出



循环程序的一般形式

- **一重循环/多重循环/循环＋（多）分支**
  - While——DO——（先判断再循环）
  - DO——While——（先循环再判断）
  - FOR
- **循环控制方式**
  - 计数控制 (LOOP/LOOPE/LOOPZ/LOOPNE/LOOPNZ)
  - 条件控制 (JCC+标号内重复执行的语句体)

4.3 循环程序设计

例4.5 求和1～100，结果存入变量SUM

```
            .model small
            .stack
            .data
sum         dw ?
            .code
            .startup
            xor ax,ax           ;被加数AX清0
            mov cx,100
again:      add ax,cx           ;从100,99,...,2,1倒序累加
            loop again
            mov sum,ax          ;将累加和送入指定单元
            .exit 0
            end
```

- 计数控制循环
- 循环次数固定

4.3 循环程序设计

例4.7 把一个字符串的所有大写字母改为小写，字符串以‘0’结尾

```
        mov bx,offset string
again:  mov al,[bx]      ;取一个字符
        or al,al         ;是否为结尾符0
        jz done          ;是，退出循环
        cmp al,'A'       ;是否为大写A～Z
        jb next
        cmp al,'Z'
        ja next
        or al,20h        ;是，转换为小写字母（使D5=1）
        mov [bx],al      ;仍保存在原位置
next:   inc bx
        jmp again        ;继续循环
done:   .exit 0
```

- 条件控制循环
- 利用标志退出

大小写字母仅 D5位不同

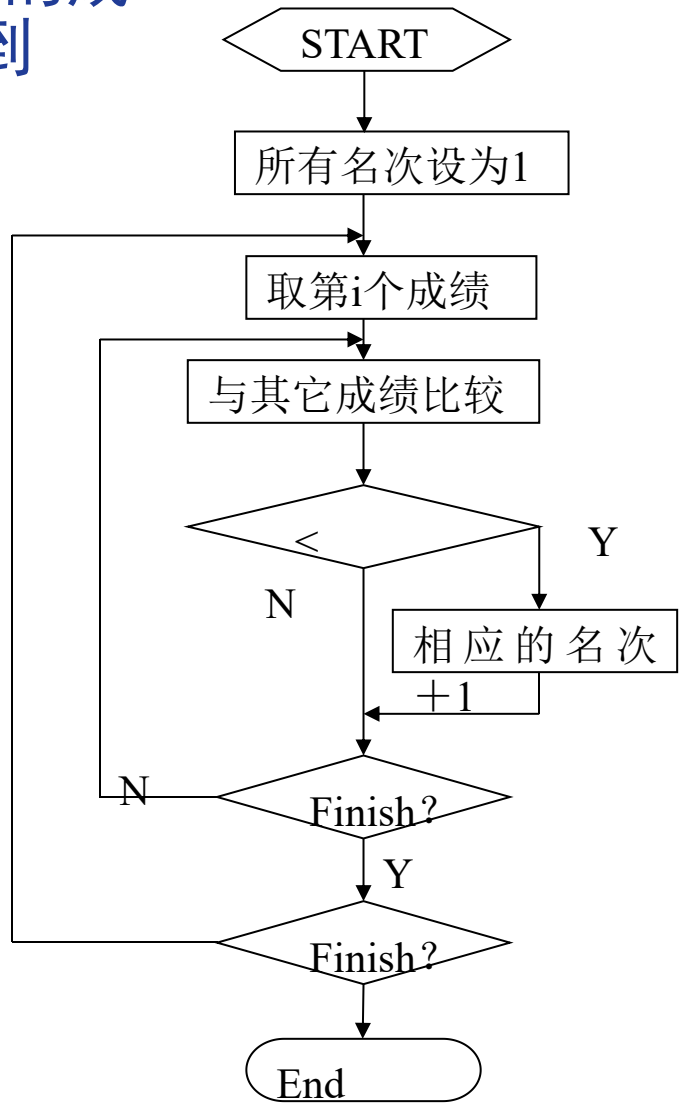
4.3 循环程序设计

例8：数据段中Score缓冲区中有5个学生的成绩（字节型）。将各自的名次算出填充到Rank缓冲区中 。

二重循环

```
        MOV CL,5
        MOV SI,0
        MOV DI,0
AGAIN:
        MOV AL,SCORE[SI]
        MOV DI,0
        MOV CH,5
GOON:
        CMP AL,SCORE[DI]
        JAE NEXT
        INC RANK[SI]
NEXT:
        INC DI
        DEC CH
        JNZ GOON

        INC SI
        DEC CL
        JNZ AGAIN
```



冒泡法

- 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；第一轮比较结束。
- 然后，开始第二轮，除掉最后一个元素，其他元素依上述方法再进行比较，得到次大的元素排在后面；第二轮结束。如此重复，直至完成，就实现了元素从小到大的排序
- 循环次数已知的双重循环程序：外层循环（轮）的次数为数据个数减1，每一轮外循环的内层循环次数等于剩下的外循环次数。
- 如5个数据，外循环次数=4，第一轮中内循环次数也为4；第一轮作完，还剩下3轮，第二轮中内循环次数等于3，以次类推。

冒泡法

排序示意图

须作4轮

第1轮

小数冒泡

序号	数	外循环次数（轮次）			
		(4轮)1	(3轮)2	(2轮)3	(1轮)4
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
		4	3	2	1
		每轮内循环次数			

大数沉底

冒泡法

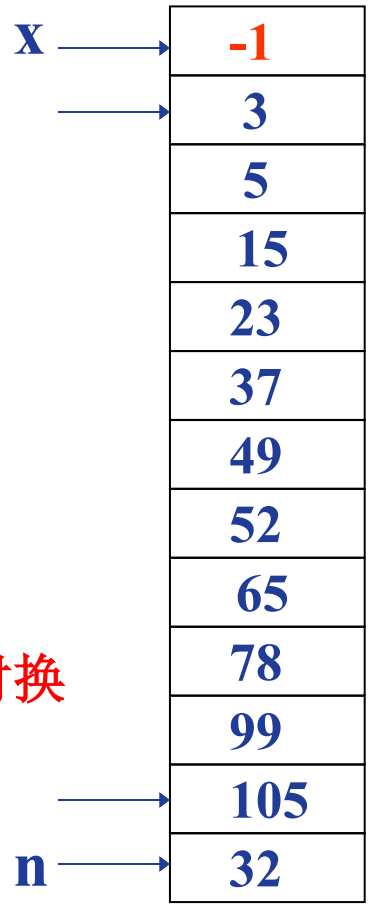
例4.8 已知元素数组元素从小到大排列，数组元素为无符号字节量

```
        mov cx,count        ;CX←数组元素个数
        dec cx              ;元素个数减1为外循环次数
outlp:  mov dx,cx           ;DX←内循环次数
        mov bx,offset array
inlp:   mov al,[bx]         ;取前一个元素
        cmp al,[bx+1]       ;与后一个元素比较
        jna next            ;前一个不大于后一个元素，则不进行交换
        xchg al,[bx+1]      ;否则，进行交换
        mov [bx],al
next:   inc bx              ;下一对元素
        dec dx
        jnz inlp            ;内循环尾
        loop outlp          ;外循环尾
```

- 计数控制
- 双重循环

例9 将正数n插入一个已整序的正字数组的正确位置

```
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 ;数组最后一个数≤ax, 则insert
        mov   bx, array_end[si]；否则最后一个数和ax对换
        mov   array_end[si+2], bx
        sub   si, 2
        jmp   short compare
insert: mov   array_end[si+2], ax ;ax放在数值最后
```



串操作指令

- 串操作指令是8086指令系统中比较独特的一类指令，采用比较特殊的数据串寻址方式，常用在操作主存连续区域的数据时

主要熟悉：　MOVS　STOS　LODS

　　CMPS　SCAS　REP

一般了解：　REPZ/REPE　REPNZ/REPNE

串数据类型

- 串操作指令的操作数

  主存中连续存放的数据串（String）/数组（Array）

  即在连续的主存区域中，字节或字的序列

  如以字节为单位的ASCII字符串

- 串操作指令的操作对象

  以字（W）为单位的字串，或是以字节（B）为单位的字节串

串寻址方式

- 源操作数用寄存器SI寻址，默认在数据段DS中，但允许段超越：DS:[SI]
- 目的操作数用寄存器DI寻址，默认在附加段ES中，不允许段超越：ES:[DI]
- CX：存放串的长度
- 每执行一次串操作指令，SI和DI将自动修改：
  - ±1（对于字节串）或±2（对于字串）
  - 执行指令CLD后，DF = 0，地址指针+1或2
  - 执行指令STD后，DF = 1，地址指针 -1或2

串操作指令

- **MOVS**
  - **格式：** MOVSB / MOVSW
  - **功能：** ES:DI ←源地址DS:SI处的一个字节/字
- **LODS**
  - **格式：** LODSB / LODSW
  - **功能：** AL/AX← DS:SI处的一个字节/字
- **STOS**
  - **格式：** STOSB / STOSW
  - **功能：** ES:DI← AL/AX

串操作指令

CMPS

- **格式：** CMPSB/W
- **功能：** DS:SI- ES:DI/ SI←SI±1, DI←DI±1

将主存中的源操作数减去目的操作数，以便设置标志，进而比较两操作数之间的关系

SCAS(scan)

- **格式：** SCASB/W
- **功能：** AL/AX- ES:DI
- 比较AL/AX与目的操作数之间的关系，设置标志，

重复操作前缀

- **REP　串指令；**
  - 若(CX)≠0，重复执行串指令；
  - CX－－＝1；

REP只能用在MOVS，LODS和STOS之前

串操作指令

例4.10：字符串传送　　dstmsg的主存区←srcmsg的字符串

```
mov ax,data
mov ds,ax
mov es,ax                ;设置附加段ES＝DS
mov si,offset srcmsg
mov di,offset dstmsg
mov cx,lengthof srcmsg ;数据存储单元个数
cld                      ；地址增量传送
rep movsb                ；重复传送字符串
mov ah,9                 ；显示字符串
mov dx,offset dstmsg
int 21h
```

Movs需要实现设置DS,ES,SI,DI和DF,CX,然后一条指令完成全部传送

重复操作前缀

- **REPZ/REPE　串指令；**
  - 若(CX)≠0且(ZF)=1，重复执行串指令；
  - 当数据串没有结束（CX≠0），并且串相等（ZF＝1），则继续比较
  - CX－－＝1；
- **REPNZ/REPNE串指令；**
  - 若(CX)≠0且(ZF)=0，重复执行串指令；
  - 当数据串没有结束（CX≠0），并且串不相等（ZF＝0），则继续比较
  - CX　－－＝1

重复前缀指令

- **REPZ和REPNZ只能用在CMPS或者SCAS之前；**
  - REPZ CMPS
    - ——找两个字符串不同的地方；
  - REPZ SCAS
    - ——在字符串中查找某一个不同的字符；
  - REPNZ CMPS
    - ——找两个字符串第一个相同的字符；
  - REPNZ SCAS
    - ——在字符串中查找某一个的字符；

串操作指令-Example

在BLOCK缓冲区中间存放有100个学生的名字，每个名字占用8个字节。名称用一个空格隔开。试查找是否有“JackChen”这个名字，找到置AL为1，否则置AL为0

```
string1 db 'abcdefgh JackChen lalalala...'
string2 db 'JackChen'
        LEA SI,STRING1
        LEA DI,STRING2
        MOV DX,SI
        MOV BX,DI
        MOV AH, 100
AGAIN:
        MOV CX,8
        REPZ CMPSB ;  (CX)≠0且zf=1继续比较
        JZ FOUND
        ADD DX,9
        MOV SI,DX
        MOV DI,BX
        DEC AH
        JNZ AGAIN
;  不是最后一名字跳转again
        MOV AL,0
;  是最后一名字，al=0
        JMP OVER
FOUND:
        MOV AL,1
OVER:
```

解答1

习题2.24(6)已知字符串string包含有32KB内容，将其中的'$'符号替换成空格。

32KB=32×2^{10}B = 2^{15}B

```
;不使用串操作指令
        mov si,offset string
        mov cx,8000h
again:  cmp byte ptr [si], '$'        ;   '$' =24h
        jnz next
        mov byte ptr [si], ' '        ;   ' '  =20h
next:   inc si
        loop again                    ; dec cx
                                      ; jnz again
```

习题2.24(6)已知字符串string包含有32KB内容，将其中的'$'符号替换成空格。

解答2

```
;使用串操作指令
        mov di,offset string
        mov al,'$'
        mov cx,8000h
        cld                 ;地址增量传送
again:  scasb               ;  al-ES:[DI],DI=DI+1
        jnz next
        mov byte ptr es : [di-1], ' '
next:   loop again
```

4.4 子程序设计

- 把功能相对独立的程序段单独编写和调试，作为一个相对独立的模块供程序使用，就形成子程序。
- 使用子程序：简化源程序结构；提高编程效率。

4.4 子程序设计

过程定义伪指令

```
过程名    PROC    [NEAR|FAR]
          过程体
          RET（RET N）
过程名    ENDP
```

- 过程名：符合语法的标识符；同模块唯一性。
- 距离属性：可省略，由汇编程序判断。

4.4 子程序设计

- **关于“距离属性”**
  - NEAR属性（段内近调用）的过程只能被相同代码段的其他程序调用
  - FAR属性（段间远调用）的过程可以被相同或不同代码段的程序调用
- 对简化段定义格式，在微型、小型和紧凑存储模式下，过程的缺省属性为near；在中型、大型和巨型存储模式下，过程的缺省属性为far
- 对完整段定义格式，过程的缺省属性为near
- 用户可以在过程定义时用near或far改变缺省属性

4.4 子程序设计

调用程序和子程序在同一代码段中

```
code  segment
assume ……
Start:
        ……
        call subr1
        ……
Mov ah,4ch
int 21h

subr1   proc near
        ……
        ret
subr1   endp

code  ends
```

```
code  segment

main  proc  far
        ……
        call subr1
        ……
        ret
main   endp

subr1   proc near
        ……
        ret
subr1   endp

code   ends
```

4.4 子程序设计

调用程序和子程序不在同一代码段中

```
subseg   segment
subt   proc  far
       ......
       ret
subt   endp

subseg   ends

mainseg   segment
       ......
       call  subt
       ......
mainseg  ends
```

子程序的调用和返回

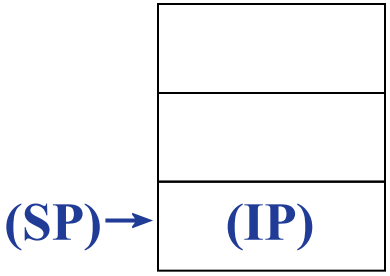
子程序调用（中断调用）：隐含使用堆栈保存返回地址

call near ptr subp

(1) 保存返回地址 PUSH IP

(2) 转子程序

(IP) ← subp的偏移地址



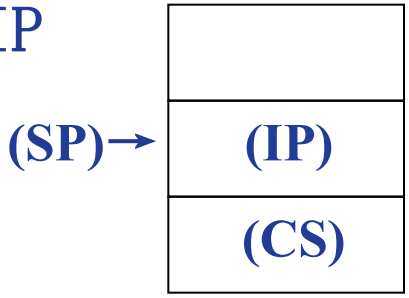
call far ptr subp

(1) 保存返回地址 PUSH CS;PUSH IP

(2) 转子程序

(CS) ← subp的段地址

(IP) ← subp的偏移地址



无参数传递的子程序

实现光标回车、换行功能

```
dpcrlf    proc          ;过程开始
          push ax       ;保护寄存器AX和DX
          push dx
          mov dl,0dh    ;显示回车 ASCII=0DH
          mov ah,2
          int 21h
          mov dl,0ah    ;显示换行 ASCII=0AH
          mov ah,2
          int 21h
          pop dx        ;恢复寄存器DX和AX
          pop ax
          ret           ;子程序返回
dpcrlf    endp          ;过程结束
```

程序与子程序之间的参数传递

- **区分**
  - 输入参数(入口)：主程序提供给子程序
  - 输出参数（出口）：子程序返回给主程序
  - 传值（数据本身）和传地址
- **方式**
  - 用寄存器来传递参数
  - 用内存单元传递参数
  - 用堆栈来传递参数

用内存数传递参数

- 适用于传递全局变量的情况

当主程序与子程序在同一个模块时，即为共享数据段的变量；不在同一模块，需要用PUBLIC/EXTERN声明。

适用与参数较多情况。

```
DSEG SEGMENT
  STR DB 'GOOD$'
DSEG ENDS

CSEG    SEGMENT
ASSUME  ……
START:
   ……
   CALL PRINTSTR
   MOV    AH,4CH
   INT    21H
```

```
PRINTSTR       PROC
          LEA DX,STR
          MOV AH,9
          INT 21H
          RET
PRINTSTR       ENDP

CSEG    ENDS
        END    START
```

用寄存器来传递参数(1)

- 传输单个变量：适用于参数较少的情况

例：编写一个子程序，完成求平方的功能，

输入参数通过DL传递，

输出参数通过DX传递

```
SQUARE  PROC   NEAR
        PUSH AX
        MOV AL,DL   ；入口参数
        MUL AL
        MOV DX,AX   ；出口参数
        POP AX
        RET
SQUARE      ENDP
```

用寄存器来传递参数(2)

- 传输指针
  - 传递一个缓冲区
- 例：编写子程序累加长度为100的无符号字节缓冲区buffer，

首地址通过BX传递，

结果通过DX传回

```
SUM  PROC   NEAR
        PUSH CX
        MOV CX, 100
        MOV DX,0
AGAIN:
        ADD DL,[BX]
        ADC DH,0
        INC BX
        LOOP AGAIN
        POP CX
        RET
SUM    ENDP
```

习题2.27：已知AX 、BX存放的是4位压缩BCD表示的十进制数，请说明如下子程序的功能和出口参数。

解答

压缩BCD码加法：

AX←AX＋BX

出口参数：

AX＝BCD码和

```
add al,bl
daa          ；低四位相加，存入al
xchg al,ah   ；低四位存入ah
adc al,bh
daa          ；高四位相加，存入al
xchg al,ah
；高四位存入ah，低四位存入al
ret
```

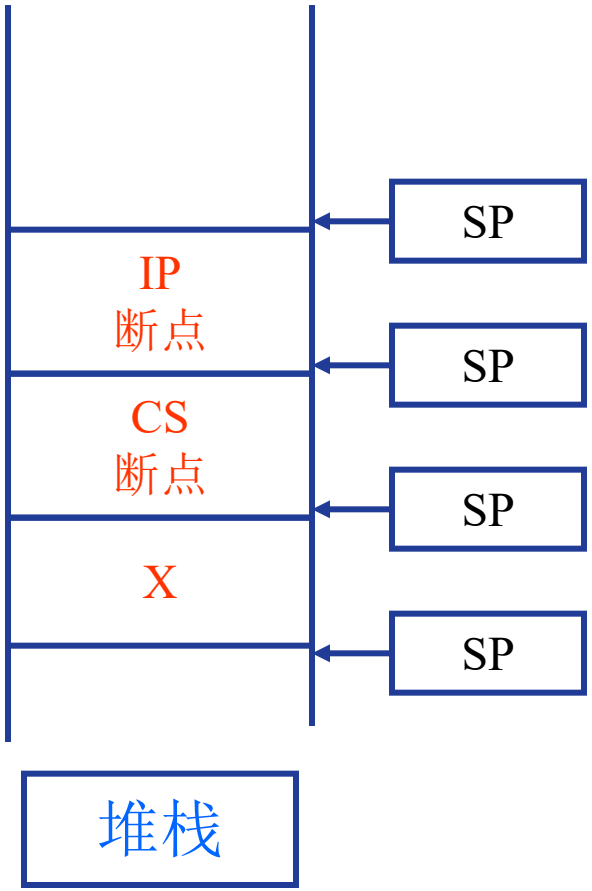
用堆栈来传递参数

- 适用性极大，高级语言均采用此方式
- 主程序把入口参数压入堆栈，子程序从堆栈取出
- 子程序把出口参数压入堆栈，主程序从堆栈取出
- 注意：
  - 段内调用时只有IP入栈
  - 段间调用时有CS，IP入栈

用堆栈来传递参数

主程序通过堆栈传参数到子程序。
设入口参数为X （WORD），子程序为FAR

- 主程序动作
  - PUSH X
  - PUSH CS
  - PUSH IP



用堆栈来传递参数

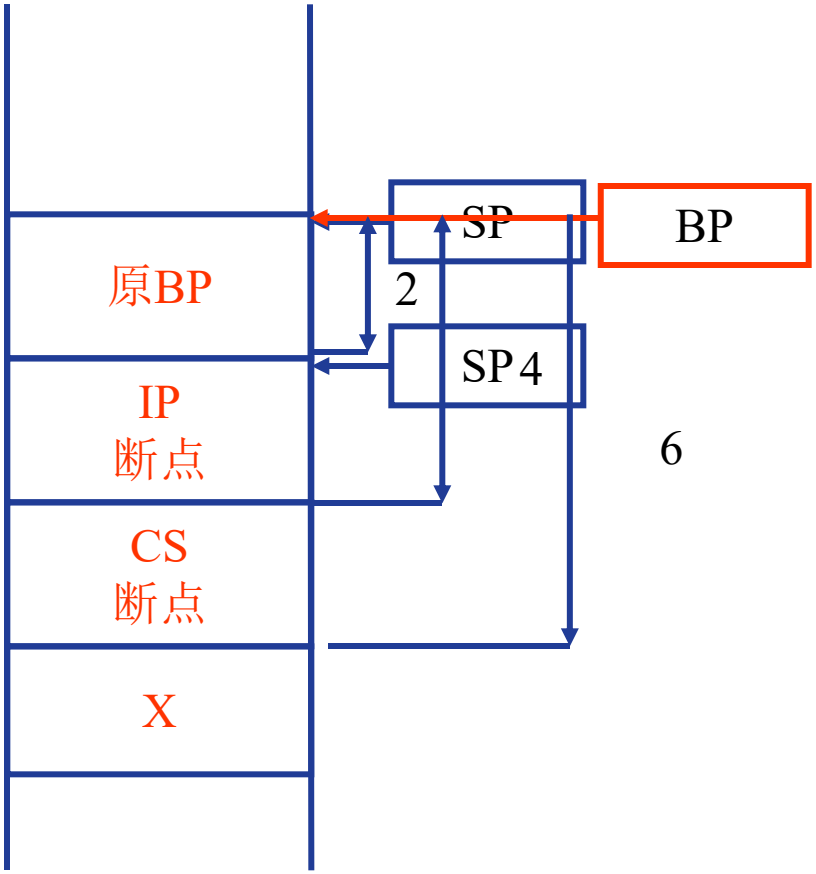
主程序通过堆栈传参数到子程序。
设入口参数为X （WORD），子程序为FAR

- 子程序提取数据X
  - PUSH BP
  - MOV BP,SP

；BP指向当前栈顶，用BP间接寻址存取入口参数

  - WORD PTR [BP+6]

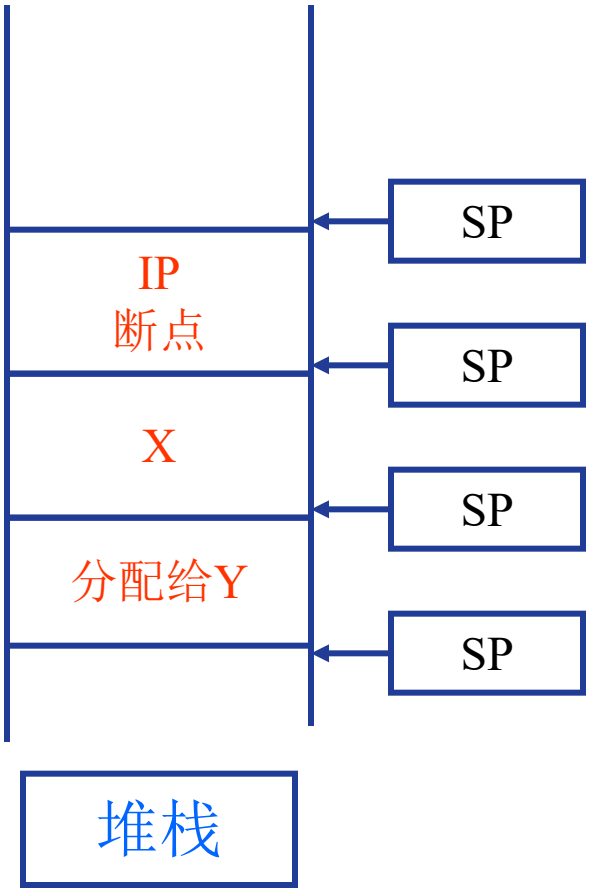
- 如果是Near调用
  - WORD PTR [BP+4]



用堆栈来传递参数

主程序通过堆栈传入和传出参数。设入口参数为X，出口参数为Y （X,Y为WORD），子程序为NEAR

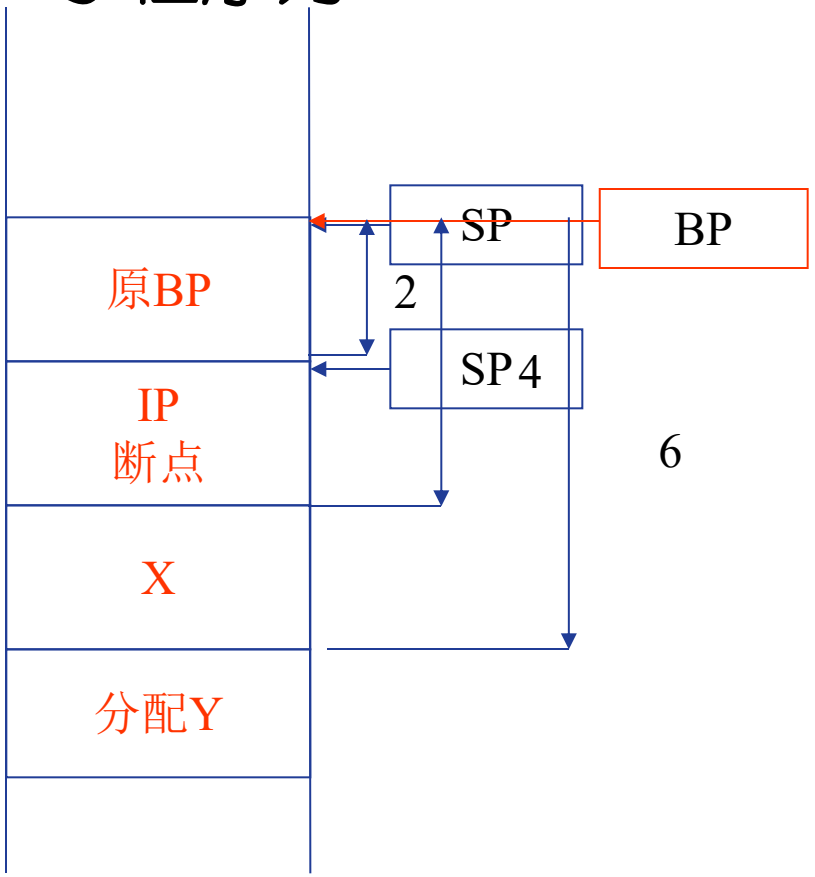
- 主程序动作
  - SUB SP,2
  - PUSH X
  - PUSH IP



用堆栈来传递参数

主程序通过堆栈传入和传出参数。设入口参数为X，出口参数为Y （X,Y为WORD），子程序为NEAR

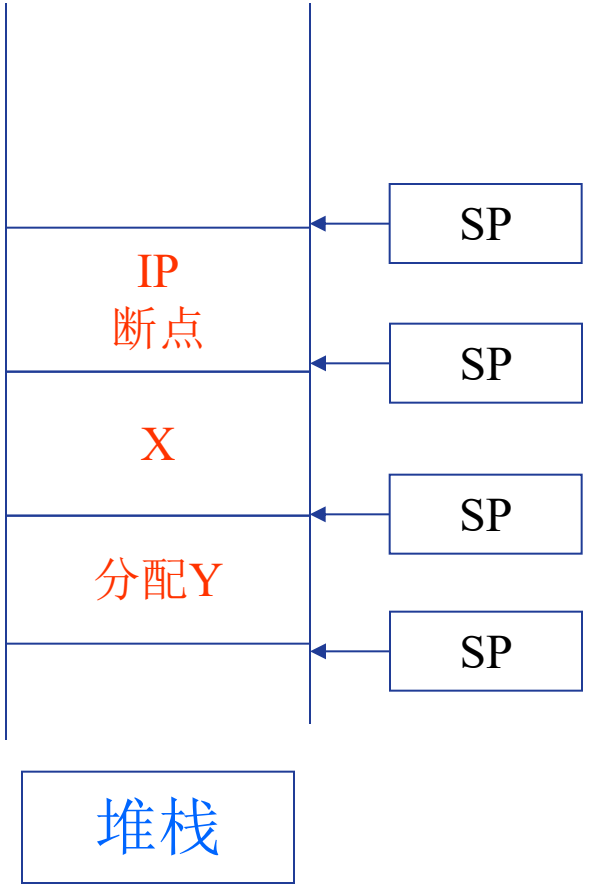
- 子程序提取数据X
  - PUSH BP
  - MOV BP,SP
  - WORD PTR [BP+4]
- 出口数据
  - WORD PTR [BP+6]



用堆栈来传递参数

主程序通过堆栈传入和传出参数。设入口参数为X，出口参数为Y （X,Y为WORD），子程序为NEAR

- RET
- 参数的销毁
  - 主程序完成
    - ADD SP,2
  - 子程序完成
    - RET 2 ; SP←SP+2
  - 取回出口参数的值
    - POP AX



用堆栈来传递参数

例4.16c array是10个元素的数组，每个元素是8位数据。子程序计算数组元素的“校验和”，校验和是指不记进位的累加。

入口参数：

顺序压入偏移地址和元素个数

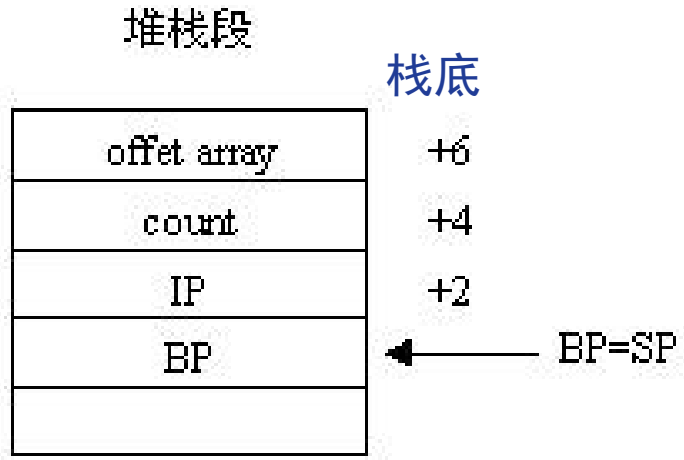
出口参数：

AL＝校验和

用堆栈来传递参数

例4.16c 主程序

```
.startup
mov ax,offset array
push ax
mov ax,count
push ax
call checksumc
add sp,4
mov result,al
.exit 0
```



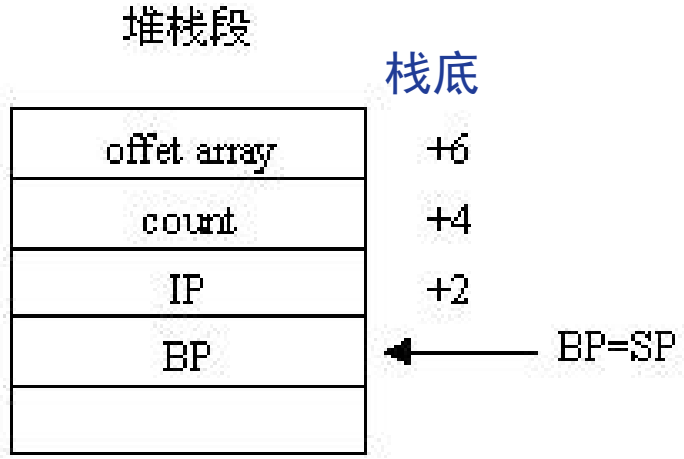
- 主程序实现平衡堆栈： add sp,n
- 子程序实现平衡堆栈： ret n

- 要注意堆栈的分配情况，保证参数存取正确、子程序正确返回，并保持堆栈平衡

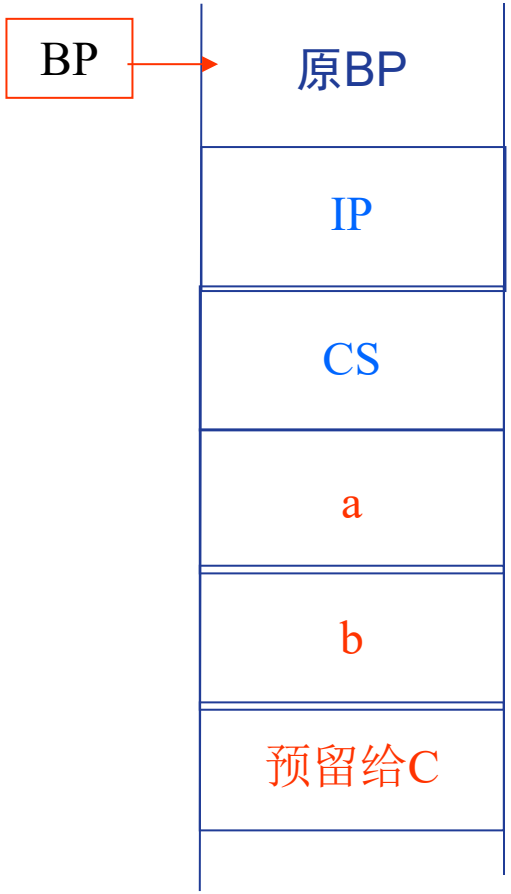
用堆栈来传递参数

例4.16c 子程序

```
checksumc   proc
            push bp
            mov bp,sp          ;利用BP间接寻址存取参数
            push bx
            push cx
            mov bx,[bp+6]      ;SS:[BP+6]指向偏移地址
            mov cx,[bp+4]      ;SS:[BP+4]指向元素个数
            xor al,al
sumc:       add al,[bx]
            inc bx
            loop sumc
            pop cx
            pop bx
            pop bp
            ret
checksumc   endp
```



用堆栈来传递参数



子程序

```
DATA    SEGMENT
        A DW 2
        B DW 1
        SUM DW 0
DATA    ENDS

SUBSEG SEGMENT
A   S   S   U   M   E
CS:SUBSEG,DS:DATA
HTON_F PROC   FAR
        PUSH    BP
        MOV     BP,SP
        PUSH    AX
        PUSH    BX

        MOV     BX,[BP+8]
        MOV     AX,[BP+6]
        SUB     AX,BX
        MOV     [BP+10],AX
        POP     BX
        POP     AX
        POP     BP
        RET     4
HTON_F ENDP
SUBSEG ENDS
```

编写子程序完成a−b=c

主程序

```
CSEG    SEGMENT
ASSUME  CS:CSEG, DS:DATA
START:
        MOV     AX,DATA
        MOV     DS,AX

        SUB     SP,  2
        PUSH    B
        PUSH    A
        CALL    HTON_F
        POP     SUM

        MOV   AH, 4CH
        INT   21H

CSEG    ENDS
        END   START
```

注意事项

- 在过程定义体内，必须有一条RET指令能被执行到。
- 调用时，最好不要强制改变调用类型。
- 子程序保护现场。（不改变寄存器内容，堆栈）
- 堆栈操作指令必须配对。
- 堆栈使用：平衡才能保证RET指令弹出的是断点地址。
- 定义允许嵌套和递归。

❖ 编写子程序时应注意的问题：

①使用简化的段定义格式时，过程定义在程序中的位置：

- 主程序的最后，即“.EXIT 0”之后，END语句之前；
- 放在主程序之前，即“.CODE”之后，“.STARTUP”之前。

②使用寄存器传递参数时，

- 带有入口参数的寄存器可以保护，也可以不保护；
- 带有出口参数的寄存器则一定不可保护和恢复；
- 其他与出口参数无关、而子程序中使用的寄存器，子程序开始处应该保护，子程序结束、返回主程序之前应该恢复。

❖ 子程序规范

一个完整的子程序，特别是供其他编程人员使用的子程序，必须附有一个详细说明：

- 子程序名（过程名）
- 子程序功能介绍
- 子程序的入口参数
- 子程序的出口参数

某子程序的说明

❖子程序名：DTOB
❖功能：完成两位十进制数转换成二进制数
❖入口参数：AL存放待转换的两位BCD码
❖出口参数：CL存放转换后的二进制数
❖占用寄存器：BX
❖示例：输入AL=01010110B （56H）
输出CL=00111000B （38H）

4.22 过程定义的一般格式？子程序入口为什麽常有PUSH指令，出口为什麽POP指令？下面的程序段有无不妥？若有请改正。

```
CRRAY   PROC
          PUSH AX
          XOR  AX, AX
          XOR  DX, DX
AGAIN:  ADD AX, [BX]
          ADC  DX, 0
          INC  BX
          INC  BX
          LOOP AGAIN
          RET
          ENDP CRRAY
```

如果需要保护AX，缺POP AX。

实际上DX AX为出口参数，两个寄存器都不能保护和恢复；
BX携带入口参数，可保护也可不保护。
有保护，势必有恢复。

位置不对

子程序的嵌套、递归与重入

❖ 子程序的嵌套

子程序又调子程序称为子程序的嵌套，嵌套的层数取决于堆栈空间的大小。嵌套子程序的设计和一般子程序完全相同。

❖ 子程序的递归

子程序直接或间接地嵌套调用自己，称为递归调用。含有递归调用的子程序称为递归子程序。

- 每次调用时不能破坏以前调用所用的参数及中间结果，因此，调用参数及中间结果一般都放在堆栈中。不可放在固定的寄存器或存储单元中。
- 要控制递归的次数，避免陷入死循环。
- 递归深度受堆栈空间的限制。

子程序的递归

例4.17 主程序－1/3

```
        .model small
        .stack
        .data
N       dw 3
result  dw ?
        .code
        .startup
        mov bx,N
        push bx          ;入口参数：N
        call fact        ;调用递归子程序
        pop result       ;出口参数：N!
        .exit 0
```

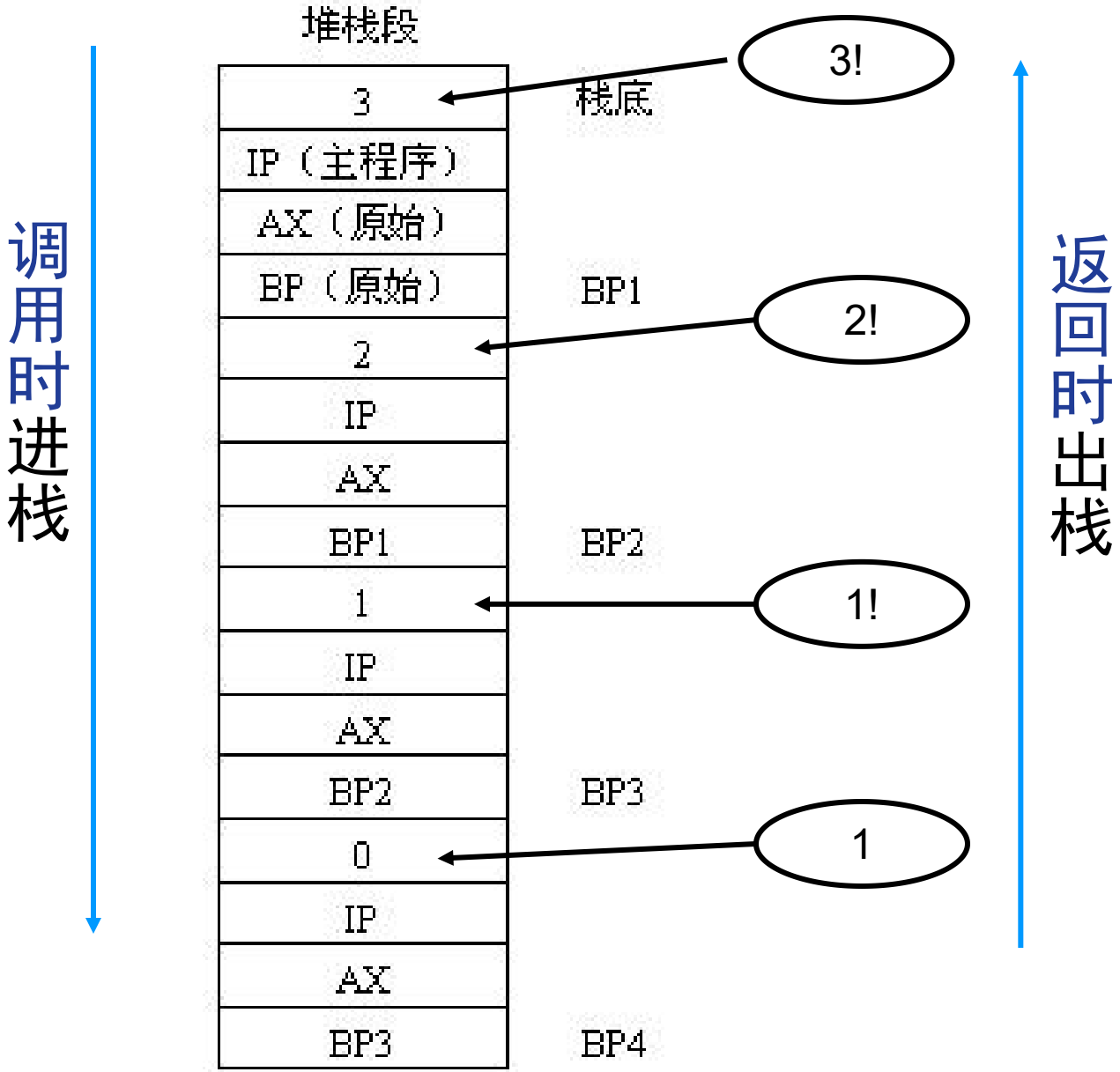
例4.17 递归子程序－2/3

```
;计算N!的近过程
;入口参数：压入 N                ;出口参数：弹出 N!
fact        proc
            push ax
            push bp
            mov bp,sp
            mov ax,[bp+6] ;取入口参数 N
            cmp ax,0
            jne fact1               ;N＞0,N!＝N×(N-1)!
            inc ax                  ;N＝0,N!＝1
            jmp fact2
```

例4.17 递归子程序－3/3

```
fact1:    dec ax          ;N-1
          push ax
          call fact       ;调用递归子程序求(N-1)!
          pop ax
          mul word ptr [bp+6]  ;求 N×(N-1)!
fact2:    mov [bp+6],ax ;存入出口参数 N!
          pop bp
          pop ax
          ret
fact      endp
```

递归子程序


堆栈段
3
IP（主程序）
AX（原始）
BP（原始）
2
IP
AX
BP1
1
IP
AX
BP2
0
IP
AX
BP3
栈底
BP1
BP2
BP3
BP4
3!
2!
1!
1
调用时进栈
返回时出栈

递归次数用N控制，由N=3，子程序共运行4次（主程序调用1次，递归调用3次）；入口参数及中间结果都用堆栈保存。

注释：

- 在进入子程序过程中，不计算阶乘值，只求中间参数。第一次进入求出中间参数2；第二次进入求出中间参数1；第三次进入求出中间参数0；第四次进入后，由于中间参数为0，开始执行返回处理。
- 在返回过程中计算阶乘：在过程3中计算1*1=1，在过程2中计算1*2=2，在过程1中计算2*3=6。
- 递归子程序可设计出效率较高的程序，但是编程较难，编出的程序易读性差，使用不多。

❖作业

4. 5/4. 19/4. 20/4. 22/4. 26

例. 设行李重量为 P，行李托运费为 M，行李托运计费规定如下： 行李重量小于等于50公斤时，单价为2元；超过50公斤时，超出部分按每公斤3元计算。设 P 存于 DATA 单元， M 存于 RESULT 单元。编程，计算行李费。

题意分析：

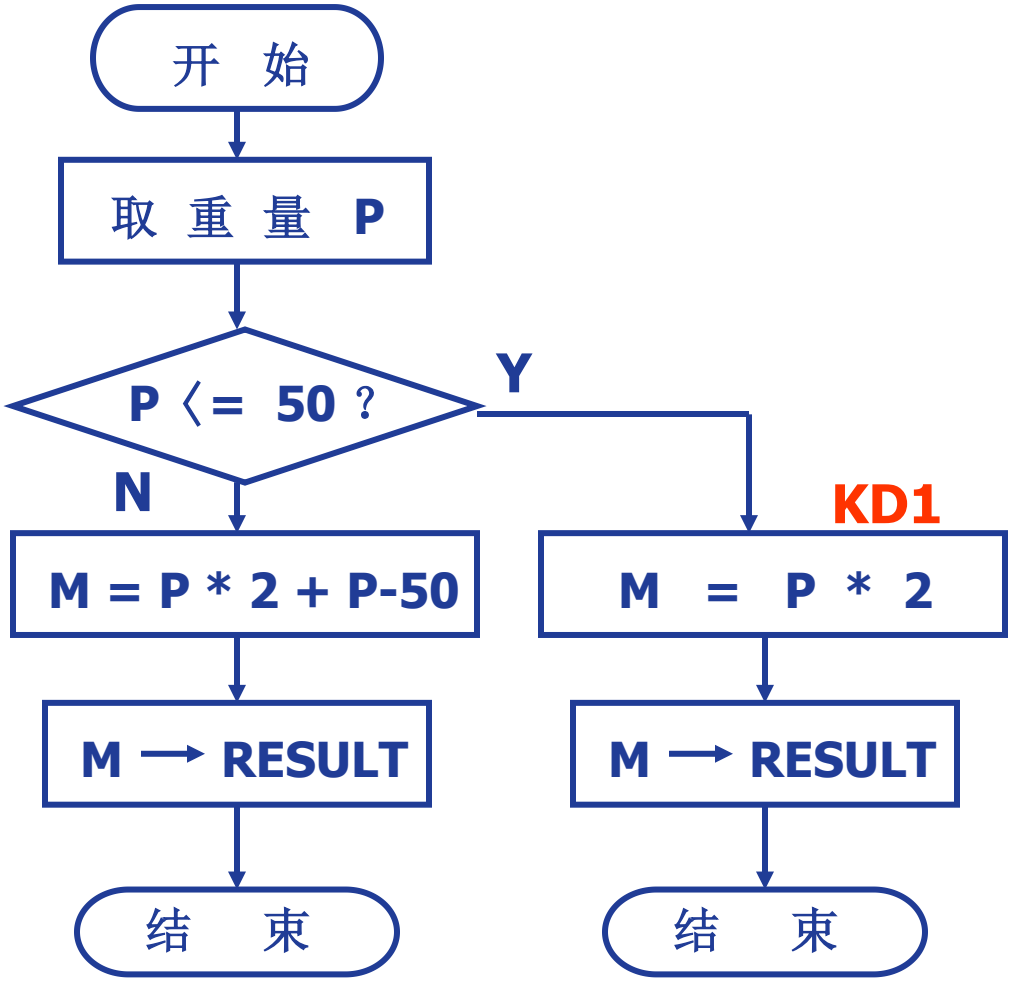
当 P 大于 50 时，

M = 2 * 50 + (P – 50) * 3

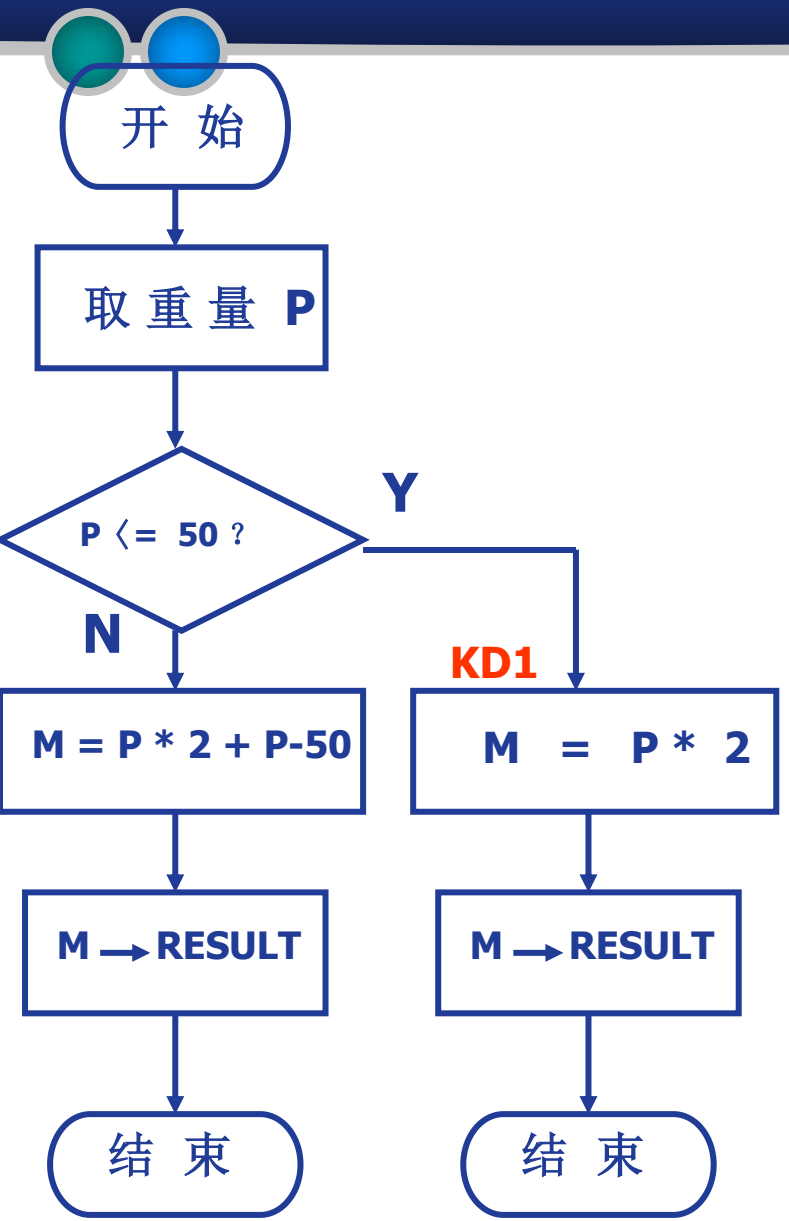
= P* 2 + P - 50 ；

否则，

M = 2 * P

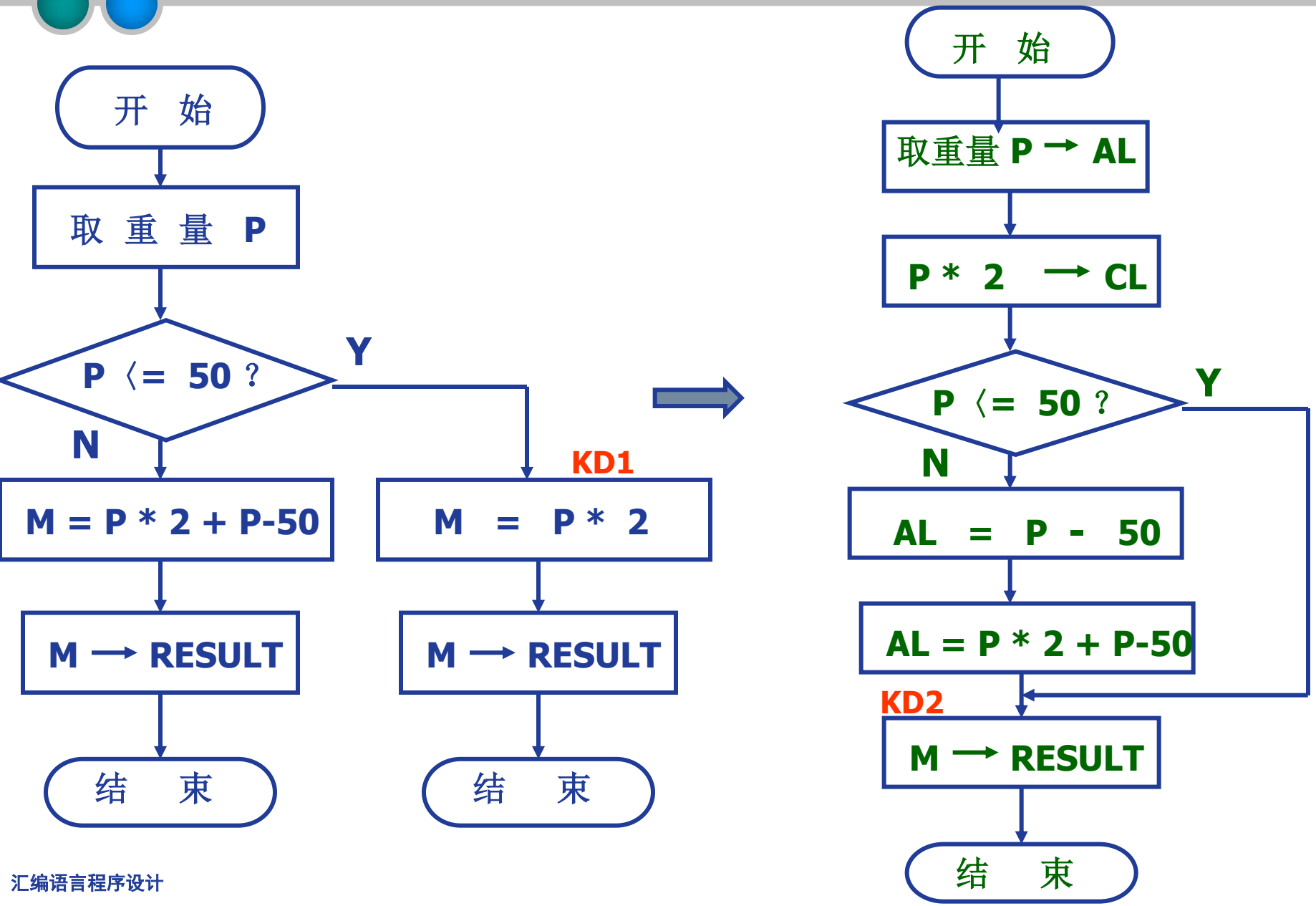


程序代码如下（1）：

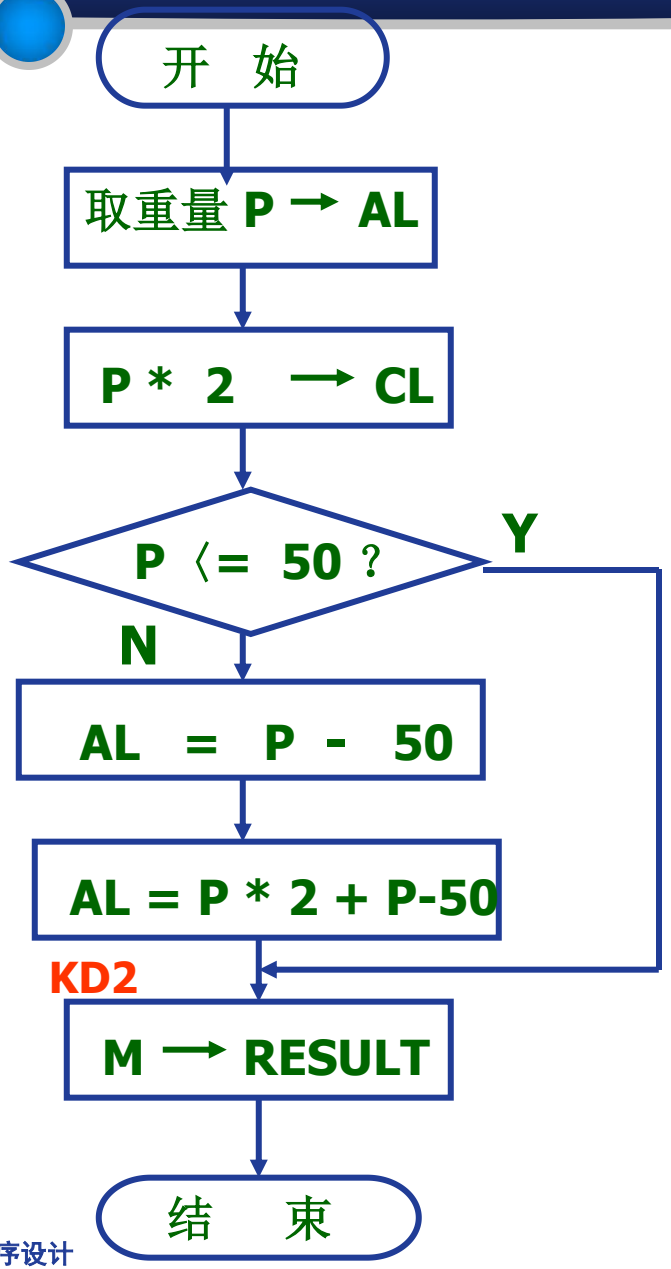



```
MAIN   PROC  FAR
       MOV   AL，DATA;  取重量送AL
       CMP   AL，50
       JBE   KD1 ;      小于等于50 ,转
KD1
       MOV   BL，AL;
       ADD   AL，BL;    P+P 送 AL
       MOV   CL，AL;    2P 送 CL
       MOV   AL，BL;    P 送 AL
       SUB   AL，50;    P-50
       ADD   AL，CL ;   2P + P-50 送 AL
       MOV   RESULT，AL ; 存结果
       RET
KD1：  ADD   AL，AL ；p + p 存入 AL
       MOV   RESULT，AL ; 存结果
       RET
MAIN   ENDP
```

上述程序可以优化：

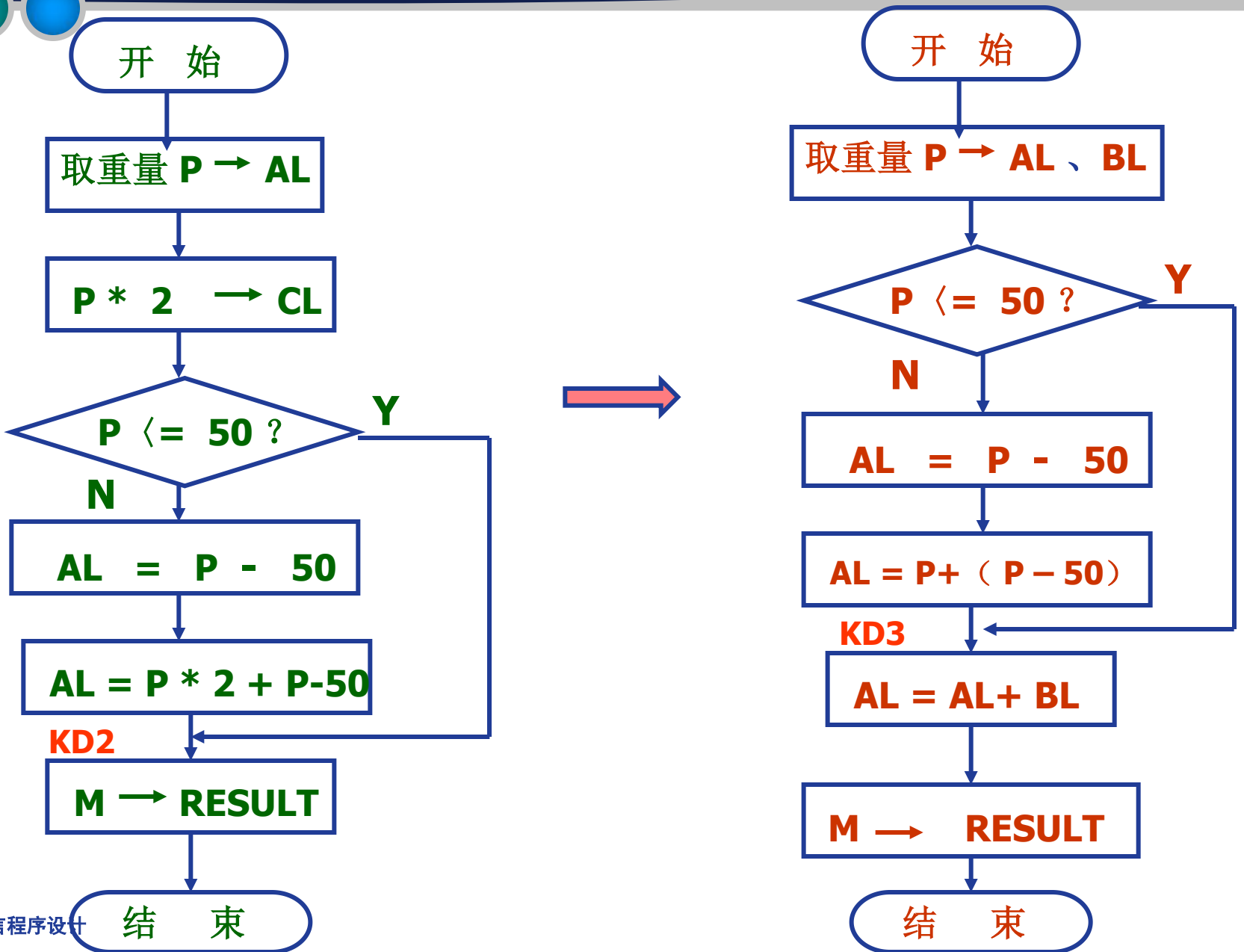

开 始
取 重 量 P
P 〈= 50 ?
Y
N
M = P * 2 + P-50
M → RESULT
结 束
KD1
M = P * 2
M → RESULT
结 束
开 始
取重量 P → AL
P * 2 → CL
P 〈= 50 ?
Y
N
AL = P - 50
AL = P * 2 + P-50
KD2
M → RESULT
结 束

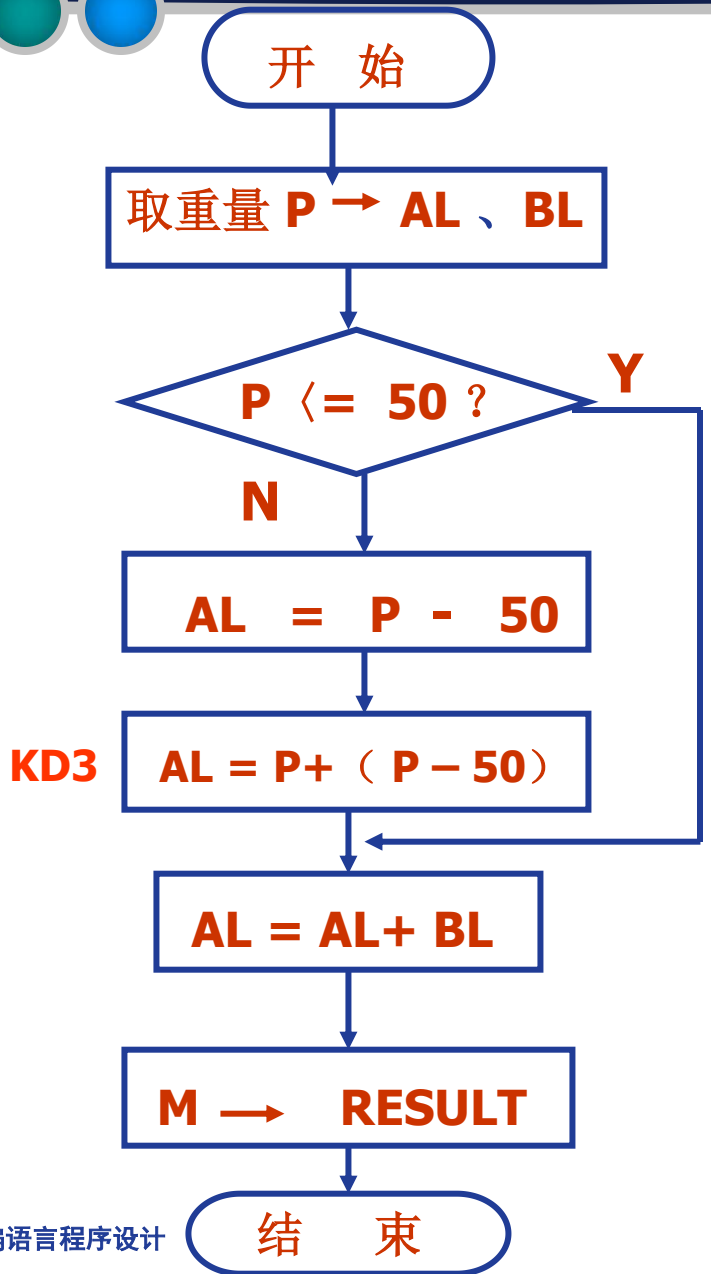
程序代码如下（2）：




```
MAIN   PROC  FAR
       MOV   BL，DATA
       MOV   AL，BL
       ADD   AL，BL
       MOV   CL，AL；  CL = 2P
       MOV   AL，BL；  AL = P
       CMP   AL，50
       JBE   KD2 ; 小于等于50 ,转KD2
       SUB   AL，50 ; AL=P-50
       ADD   AL，CL;
       MOV   CL，AL
KD2:   MOV   AL，CL ;
       MOV   RESULT，AL
       RET
MAIN   ENDP
```

上述程序进一步优化（3）：


开 始
取重量 P → AL
P * 2 → CL
P〈= 50？
Y
N
AL = P - 50
AL = P * 2 + P-50
KD2
M → RESULT
结 束
开 始
取重量 P → AL 、BL
P〈= 50？
Y
N
AL = P - 50
AL = P+（P－50）
KD3
AL = AL+ BL
M → RESULT
结 束




```
MAIN  PROC FAR
      --------
        MOV   AL，DATA；  AL=P
        MOV   BL，AL
        CMP   AL，50
        JBE   KD3
        SUB   AL，50; AL =P-50
        ADD   AL，BL; AL=AL+P
KD3：   ADD   AL，BL; AL=AL+P
        MOV   RESULT，AL ;存结果
        RET
MAIN  ENDP
```

例：　奖学金评定时，规定：

总评成绩在 90到100之间获一等奖，

总评成绩在 80到89之间获二等奖，

总评成绩在 70到79之间获三等奖。

在缓冲区GRADE中放有10个学生的总评成绩，分别统计获一等奖、二等奖、三等奖的人数。

编程思路：

此程序完成10个数据的判断、统计，程序选用CX作计数器，以减计数的方式控制循环。

选用 SI 作地址指针，指针初始应指向GRADE 。

在循环体中修改地址指针 SI = SI +2;

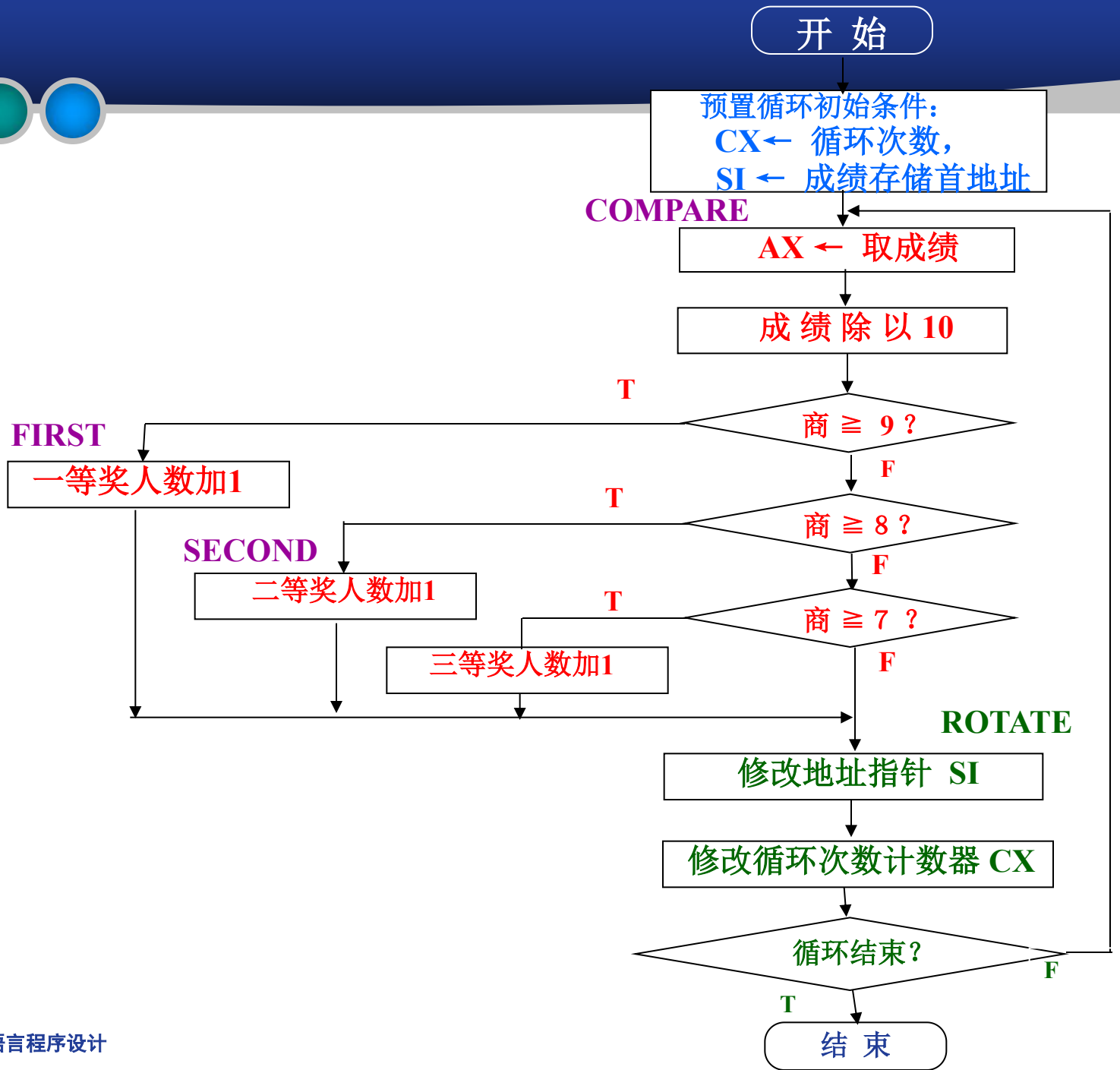
修改计数器 CX = CX - 1。

当 CX =0 时，退出循环。

设 三个计数器：　PRIZE_1　　保留获一等奖人数

PRIZE_2　　保留获二等奖人数

PRIZE_3　　保留获三等奖人数


开 始
预置循环初始条件：
CX← 循环次数，
SI ← 成绩存储首地址
COMPARE
AX ← 取成绩
成 绩 除 以 10
T
商 ≧ 9 ？
FIRST
一等奖人数加1
F
T
商 ≧ 8 ？
SECOND
二等奖人数加1
F
T
商 ≧ 7 ？
三等奖人数加1
F
ROTATE
修改地址指针 SI
修改循环次数计数器 CX
循环结束？
F
T
结 束

程序如下：

```
DATA    SEGMENT
GRADE   DB      45,70,78,86,94,100,83,88,76,65
PRIZE_1 DB      0       ;一等奖人数
PRIZE_2 DB      0       ;二等奖人数
PRIZE_3 DB      0       ;三等奖人数
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
```

```
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        MOV     CX,10           ;  置循环计数器初值
        LEA     SI, GRADE       ;  置地址指针SI初值
        MOV     BL, 10
COMPARE: MOV    AL,[SI]         ;  将成绩送入AL
        CBW
        DIV     BL              ;  成绩除10
        CMP     AL,9            ;  商是否大于等于9
        JAE     FIRST           ;  是, 转FIRST
```

```
                CMP     AL,8            ; 商是否大于等于8
                JAE     SECOND            ; 是, 转SECOND
                CMP     AL,7            ; 商是否大于等于7
                JB      ROTATE          ; 小 于7, 进入下一轮循环
                INC     PRIZE_3         ; 大于等于7, 三等奖人数加1
                JMP     ROTATE
FIRST:          INC     PRIZE_1         ; 大于等于9, 一等奖人数加1
                JMP     ROTATE
SECOND:         INC     PRIZE_2         ; 大于等于8, 二等奖人数加1
ROTATE:         INC     SI              ; 修改地址指针
                LOOP    COMPARE
                RET

MAIN            ENDP
CODE            ENDS
                END     MAIN
```

例 计算有符号数的平均值。

;入口参数用堆栈传递，出口参数用寄存器AX传递。

;要计算16位有符号数的和，被加数一定要进行符号扩展。

```
        .model small
        .stack
        .data
array   dw 1234,-1234,1,1,-1,32767,
-32768,5678,-5678,9000
count   equ ($-array)/2 ;数据个数
wmed    dw ?
        .code
        .startup
        mov ax,count
        push ax         ; 参数1
        mov ax,offset array
        push ax         ; 参数2
        call mean
        add sp,4        ; 平衡堆栈
        mov wmed,ax
        .exit 0
```

```
mean    proc
        push bp
        mov bp,sp
        push bx
        push cx
        push dx
        push si
        push di
        mov bx,[bp+4]；取参数2：偏移地址
        mov cx,[bp+6]；取参数1：数据个数
        xor si,si
        mov di,si
```

```
mean1:  mov ax,[bx]
        cwd
        add si,ax
        adc di,dx
        inc bx
        inc bx
        loop mean1
        mov ax,si
        mov dx,di
        mov cx,[bp+6]
```

```
  dx.ax
+ di.si
  di.si
```

```
        idiv cx  ;求平均值，商在AX，余数在DX
        pop di
        pop si
        pop dx
        pop cx
        pop bx
        pop bp
        ret
mean    endp
        end
```

递归求解：
Sum(n)=1+2…+n

若n=0，Sum(n)=0；
否则= Sum(n-1)+ n

```
        …
n    DW  100
Result   DW  ?
```

```
        …
SUB     SP,2
PUSH    n

CALL    SUM
POP     Result
```

```
SUM  PROC  FAR
     PUSH   BP
     MOV  BP,SP
     PUSH   AX
     MOV AX,[BP+6]
     CMP   AX,0
     JNZ    SUM1
     MOV   AX,0
     JMP    EXIT
SUM1:  SUB SP,2
     DEC AX
     PUSH  AX
     CALL  SUM
     POP AX
     ADD  AX, [BP+6]
EXIT:
     MOV [BP+8],AX
     POP AX
     POP  BP
RET 2
```