第五章 简单应用程序设计

本章主要内容：

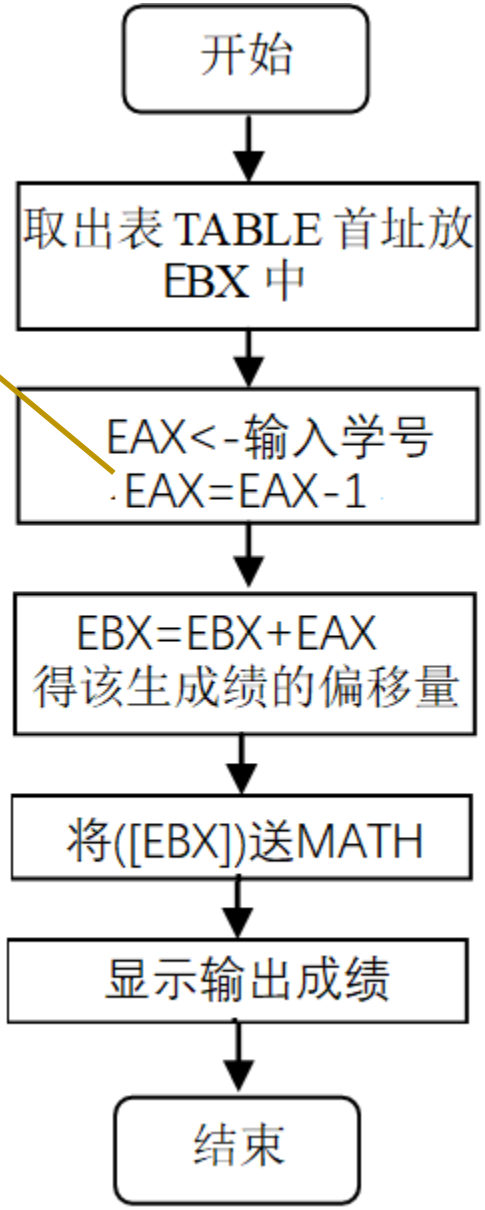
- 顺序程序设计示例
- 分支程序设计示例
- 循环程序设计示例
- 子程序设计示例

5.1 顺序程序设计示例

学号从1开始，存放地址从0开始

例1 利用学号查学生的数学成绩表。

算法分析：首先在数据段中建立一个成绩表TABLE，在表中各学生的成绩按照学号从小到大的顺序存放。要查的学号从键盘输入，查表的结果放在变量MATH中并输出显示。



```
      TITLE   TABLE       LOOKUP
.386
.model flat,stdcall
.stack  4096
include Irvine32.inc
includelib Irvine32.lib
ExitProcess PROTO,dwExitCode:DWORD

.data
table    DB   81, 78, 90, 64, 85, 76, 93, 82, 57, 80
         DB   73, 62, 87, 77, 74, 86, 95, 91, 82, 71
math     DB   ?
```

```
.code
main PROC                          ; 定义主过程
        mov  ebx, offset table   ;EBX指向表首址
        xor  eax, eax            ;将eax清零
        call ReadDec                ;调用函数从键盘输入学号
        dec   eax                    ;实际学号是从1开始的
        add   ebx, eax             ;ebx加上学号指向要查的成绩
        xor eax, eax               ;清除eax,保证输出时只显示AL
        mov  al, [ebx]             ;查到成绩送AL
        mov  math, al              ;存结果
        call WriteDec              ;调用函数输出成绩
        invoke  ExitProcess, 0  ;退出程序，返回操作系统
 main   ENDP
        END   main
```

5.2 分支程序设计示例

分支程序的结构有两种常见结构：

- 用比较/测试指令+条件转移指令实现分支
- 用跳转表形成多路分支

1、用比较/测试指令+条件转移指令实现分支

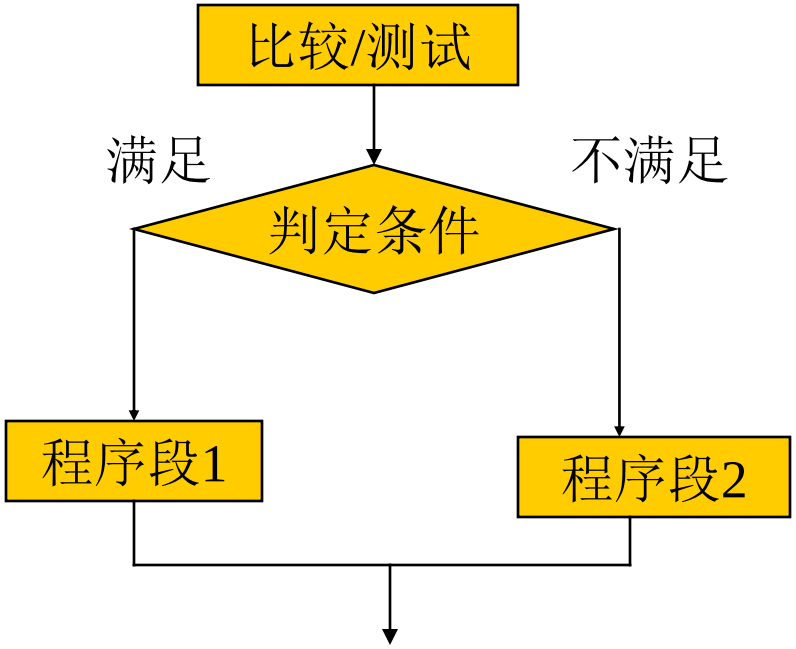
比较指令：CMP DEST，SRC

该指令的功能与减法指令SUB相似，区别是(DEST)-(SRC)的差值不送入DEST。而其结果影响标志位。

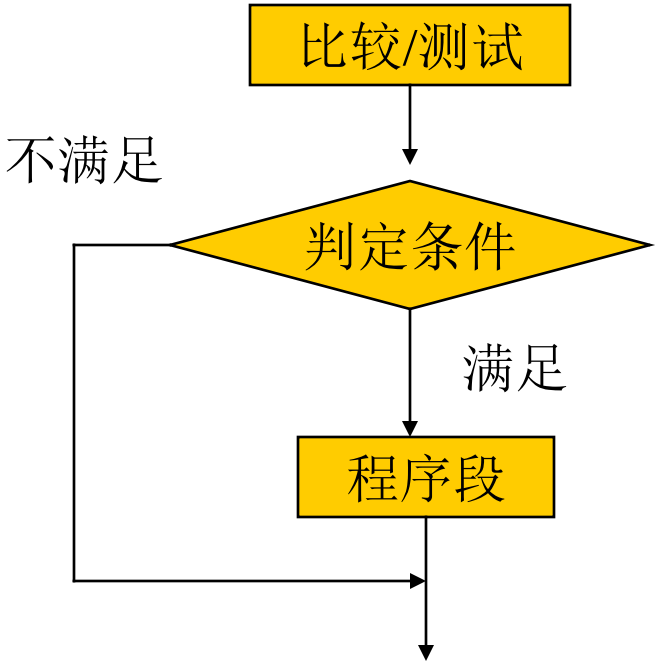
测试指令：TEST DEST，SRC

该指令的功能与逻辑指令AND相似，区别是逻辑“与”的结果不送入DEST。只是结果影响标志位。

这种类型的分支程序有两种结构



IF-THEN-ELSE



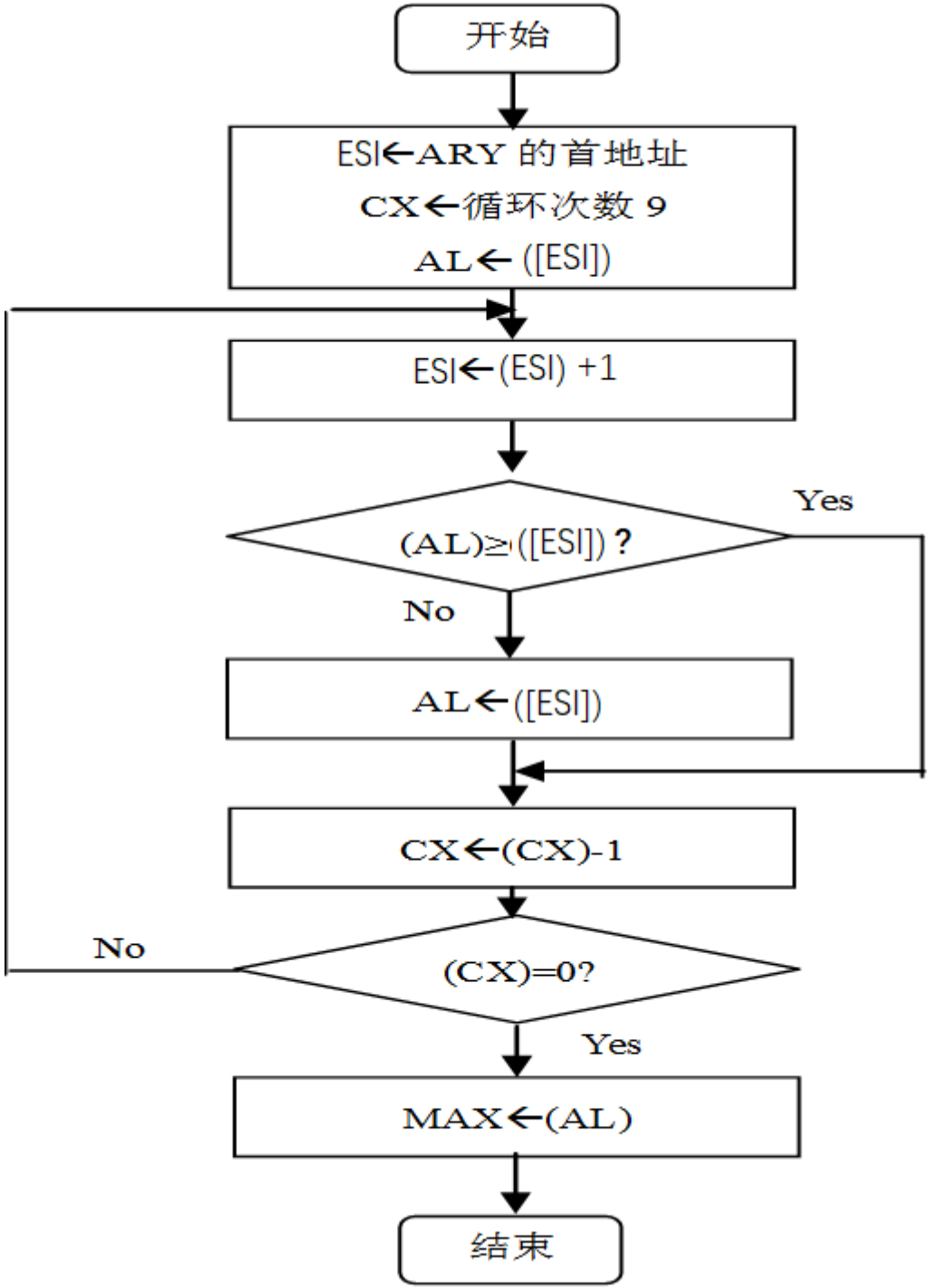
IF-ELSE

一条条件转移指令只能实现两条分支的程序。要实现更多条分支的程序，需使用多条条件转移指令。

例2 数据段的ARY数组中存放有10个无符号数，试找出其中最大者送入MAX单元。

算法分析：

- 依次比较相邻两数的大小，将较大的送入AL中。
- 每次比较后，较大数存放在AL中，相当于较大的数往下传。
- 比较一共要做9次。
- 比较结束后，AL中存放的就是最大数。


开始
ESI←ARY 的首地址
CX←循环次数 9
AL← ([ESI])
ESI←(ESI) +1
(AL)≥([ESI])？
Yes
No
AL←([ESI])
CX←(CX)-1
(CX)=0?
No
Yes
MAX←(AL)
结束

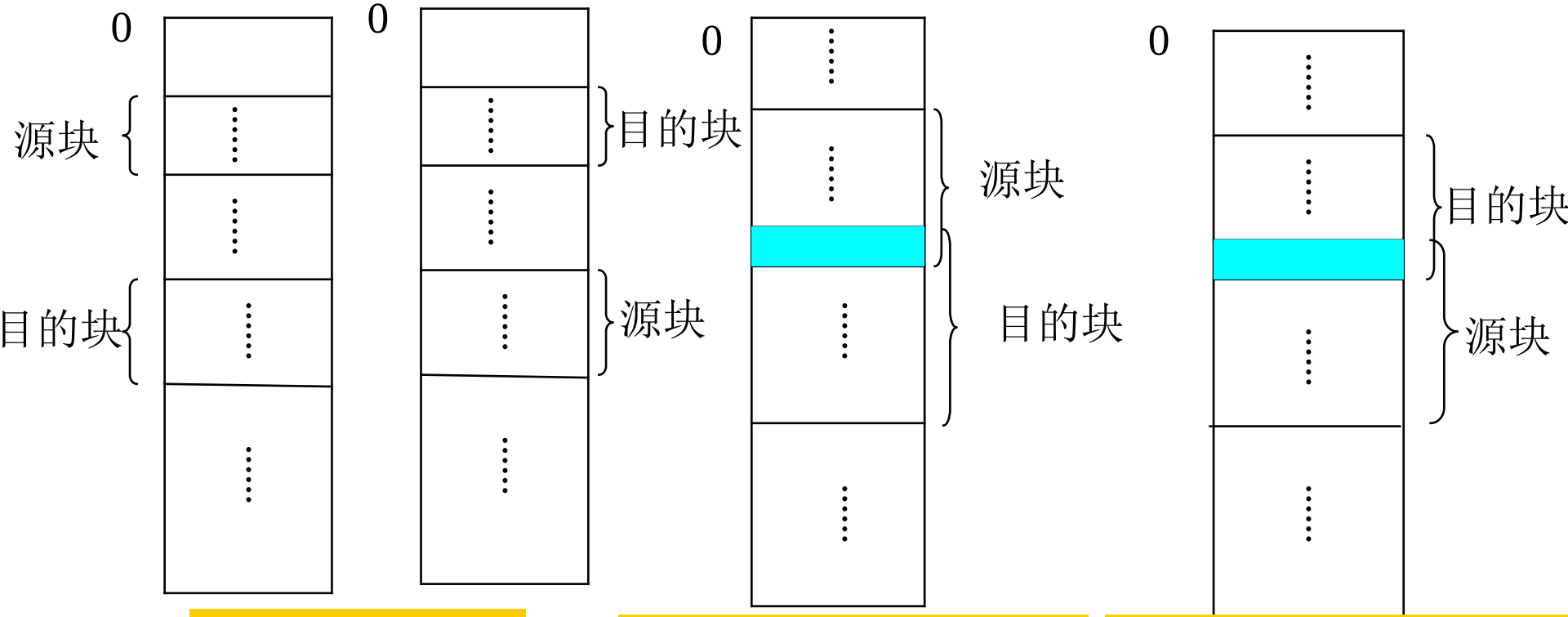
```
.386
.MODEL flat,stdcall
.STACK  4096
ExitProcess PROTO,dwExitCode:DWORD

.DATA
   ARY    DB    17，5，40，0，67，12，34，78，32，10
   MAX    DB    ?

.CODE
MAIN PROC
          MOV    ESI, OFFSET ARY ; ESI指向ARY的第一个元素
          MOV    CX, 9          ;CX作次数计数器
          MOV    AL, [ESI]      ;取第一个元素到AL
LOP：     INC    ESI            ;ESI指向后一个元素
          CMP    AL, [ESI]      ;比较两个数
          JAE    BIGER          ;前元素≥后元素转移
          MOV    AL, [ESI]      ;取较大数到AL
BIGER：   DEC    CX             ;减1计数
          JNZ    LOP            ;未比较完转回去，否则顺序执行
          MOV    MAX, AL    ;存最大数
          INVOKE  ExitProcess, 0
MAIN  ENDP
        END   MAIN
```

例3　编写一程序，实现将存储器中的源数据块传送到目的数据块。

在存储器中两个数据块的存放有下列情况：两个数据块分离和有部分重叠。



两个数据块分离

可以从首址或末址开始传送

源块首址 < 目的块首址

必须从数据块末址开始传送

源块首址 > 目的块首址

必须从数据块首址开始传送

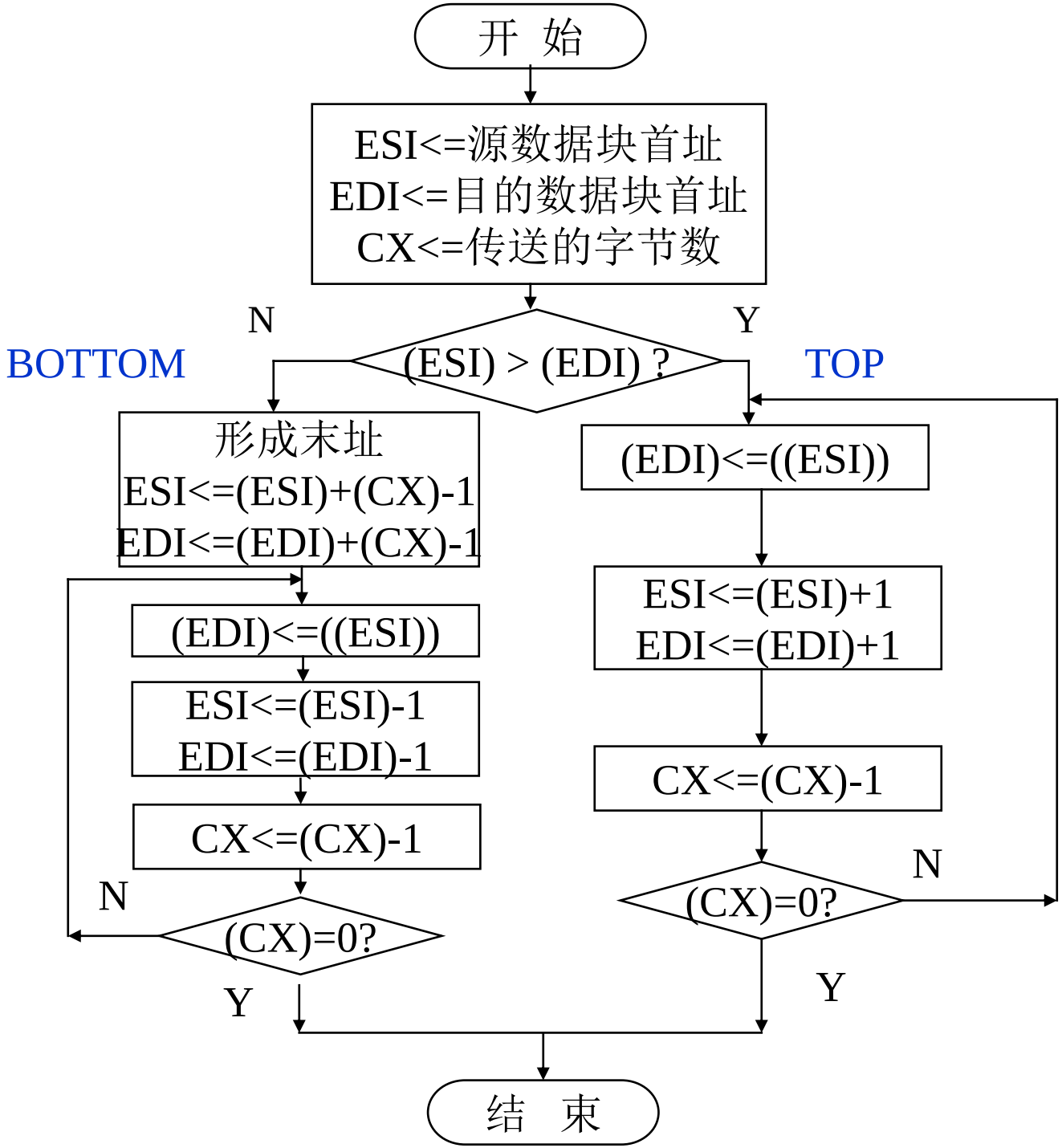
三种相对位置情况的传送方法归纳：

对于源块和目的块分离的情况，不论是从数据块的首址还是末址开始传送都可以。

对于源块与目的块有重叠且源块首址<目的块首址的情况，必须从数据块末址开始传送。

对于源块与目的块有重叠且源块首址>目的块首址的情况，必须从数据块首址开始传送。

因此，我们设定：当源块首地址<目的块首地址时，从数据块末地址开始传送。反之，则从首地址开始传送。


开 始
ESI<=源数据块首址
EDI<=目的数据块首址
CX<=传送的字节数
N
Y
BOTTOM
(ESI) > (EDI) ?
TOP
形成末址
ESI<=(ESI)+(CX)-1
EDI<=(EDI)+(CX)-1
(EDI)<=((ESI))
ESI<=(ESI)-1
EDI<=(EDI)-1
CX<=(CX)-1
N
(CX)=0?
Y
(EDI)<=((ESI))
ESI<=(ESI)+1
EDI<=(EDI)+1
CX<=(CX)-1
N
(CX)=0?
Y
结 束

```
 TITLE  DATA BLOCK MOVE
.386
.MODEL flat,stdcall
.STACK  4096
ExitProcess PROTO,dwExitCode:DWORD

.DATA
            ORG  $+20H
STRG      BYTE   'ABCDEFGHIJ' ; 数据块
LENG    EQU   $-STRG        ;数据块字节长度
BLOCK1 DWORD   STRG           ;源块首址
BLOCK2 DWORD  STRG-5         ;目的块首址
.CODE
MAIN PROC
            MOV CX,LENG       ;设置计数器初值
            MOV ESI,BLOCK1    ;ESI指向源块首址
            MOV EDI,BLOCK2   ;EDI指向目的块首址
```

```
        CMP  ESI,EDI            ;源块首址>目的块首址吗？
        JA  TOP                 ;大于则转到TOP处，否则顺序执行
        ADD ESI,LENG-1          ;ESI指向源块末址
        ADD EDI,LENG-1          ;EDI指向目的块末址
BOTTOM: MOV AL，[ESI]           ;从末址开始传送
        MOV [EDI],  AL
        DEC ESI
        DEC EDI
        DEC CX
        JNE   BOTTOM
        JMP   END1
  TOP: MOV AL,[ESI]             ;从首址开始传送
        MOV [EDI],AL
        INC ESI
        INC EDI
        DEC CX
        JNE TOP
 END1:  INVOKE  ExitProcess, 0
MAIN    ENDP
        END   MAIN
```

2、用跳转表形成多路分支

当程序的分支数量较多时，采用跳转表的方法可以使程序长度变短， 跳转表有两种构成方法：

- 跳转表用入口地址构成
- 跳转表用无条件转移指令构成

（1）跳转表用入口地址构成

在程序中将各分支的入口地址组织成一个表放在数据段中，在程序中通过查表的方法获得各分支的入口地址。

例4 设某程序有10路分支，试根据变量N的值（1~10），将程序转移到其中的一路分支去。

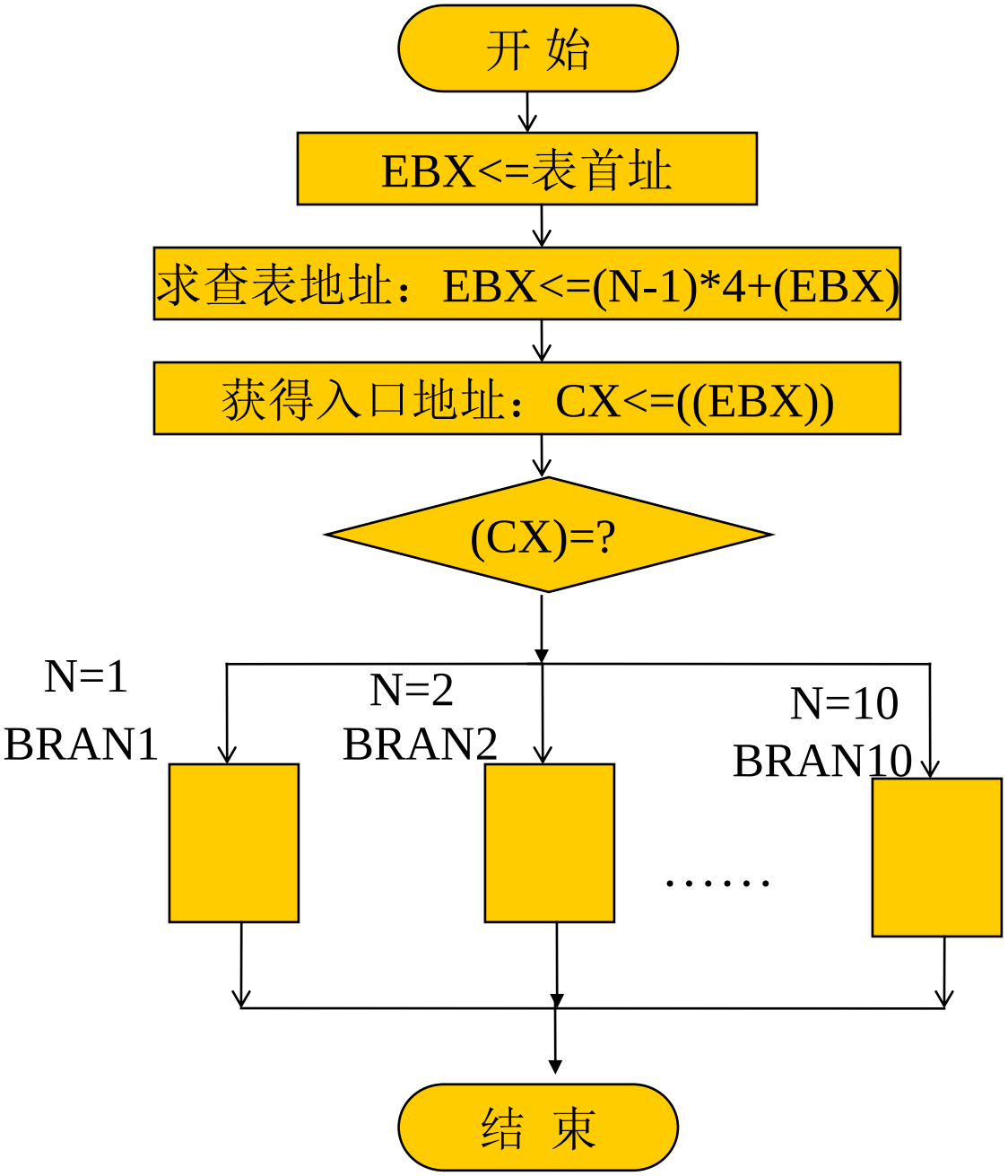
设10路分支程序段的入口地址分别为：BRAN1、BRAN2.....BRAN10。

当变量N为1时，转移到BRAN1；N为2时，转移到BRAN2，依次类推。

在跳转表中每四个字节存放一个入口地址的偏移量，如右图所示。

程序中，先根据N的值形成查表地址：（N-1）×4+表首址。

跳转表


开 始
EBX<=表首址
求查表地址：EBX<=(N-1)*4+(EBX)
获得入口地址：CX<=((EBX))
(CX)=?
N=1
BRAN1
N=2
BRAN2
N=10
BRAN10
……
结 束

多路分支结构流程图

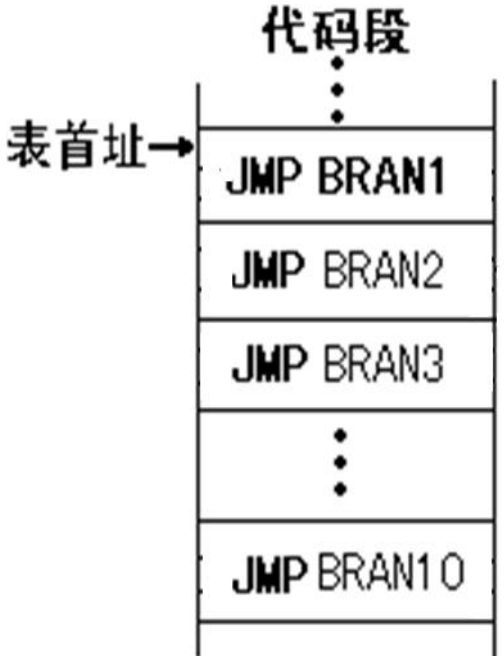
```
.DATA
ATABLE    DWORD     BRAN1,BRAN2,BRAN3,...,BRAN10
N             BYTE      3
.CODE
MAIN PROC
          XOR    EAX，EAX
          MOV   AL，N
          DEC    AL
          SHL    AL，2
          MOV    EBX，OFFSET ATABLE  ;EBX指向表首址
          ADD    EBX，EAX            ;EBX指向查表地址
          MOV    ECX，[EBX]          ;将N对应的分支入口地址送到ECX中
          JMP    ECX                 ;转移到N对应的分支入口地址
BRAN1  LABEL  NEAR          ; 定义一个入口地址放入ATALE表中
          …
          JMP    END1
BRAN2  LABEL NEAR           ; 定义一个入口地址放入ATALE表中
          …
          JMP    END1
BRAN10 LABEL NEAR           ; 定义一个入口地址放入ATALE表中
           …
END1：INVOKE  ExitProcess, 0
MAIN    ENDP
          END   MAIN
```

（2）跳转表用无条件转移指令构成

- 跳转表的每一个项目就是一条无条件转移指令。这时跳转表是代码段中的一段程序。
- 考虑转移距离短，用直接近转移，每条JMP指令占5字节。

```
.DATA
        N  DWORD  3
.CODE
MAIN    PROC
        MOV  EBX, N
        DEC   EBX
        MOV  EAX, EBX
        SHL    EBX,2
        ADD   EBX+EAX  ;4条指令实现(N-1)*5
        ADD  EBX, OFFSET  ITABLE ;EBX指向查表地址
       JMP  EBX              ;转移到N对应的JMP指令

ITABLE:JMP  BRAN1        ;JMP指令构成的跳转表
        JMP  BRAN2        ;每一条指令都是2字节的编码
        JMP  BRAN3
            …                    :
        JMP  BRAN10
```

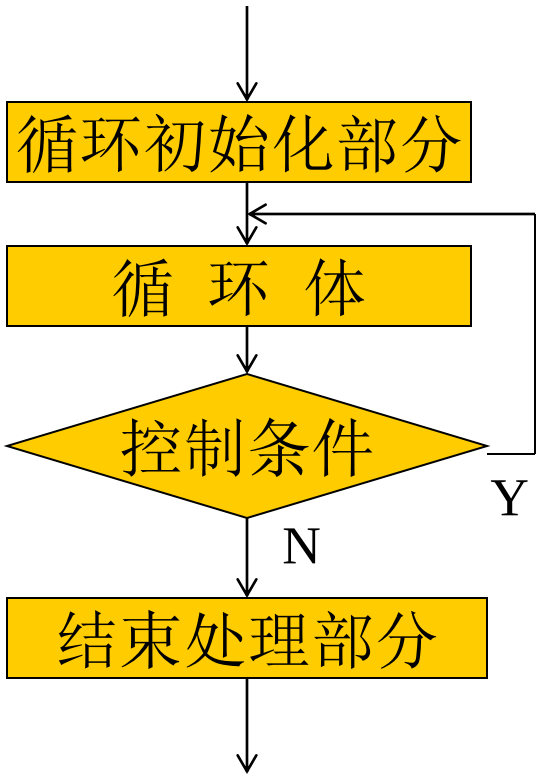


```
BRAN1: ...
          …
          JMP  END1
BRAN2: …
          …
          JMP  END1
          …
BRAN10:…
          :
          …
END1:
          INVOKE  ExitProcess, 0
MAIN    ENDP
          END   MAIN
```

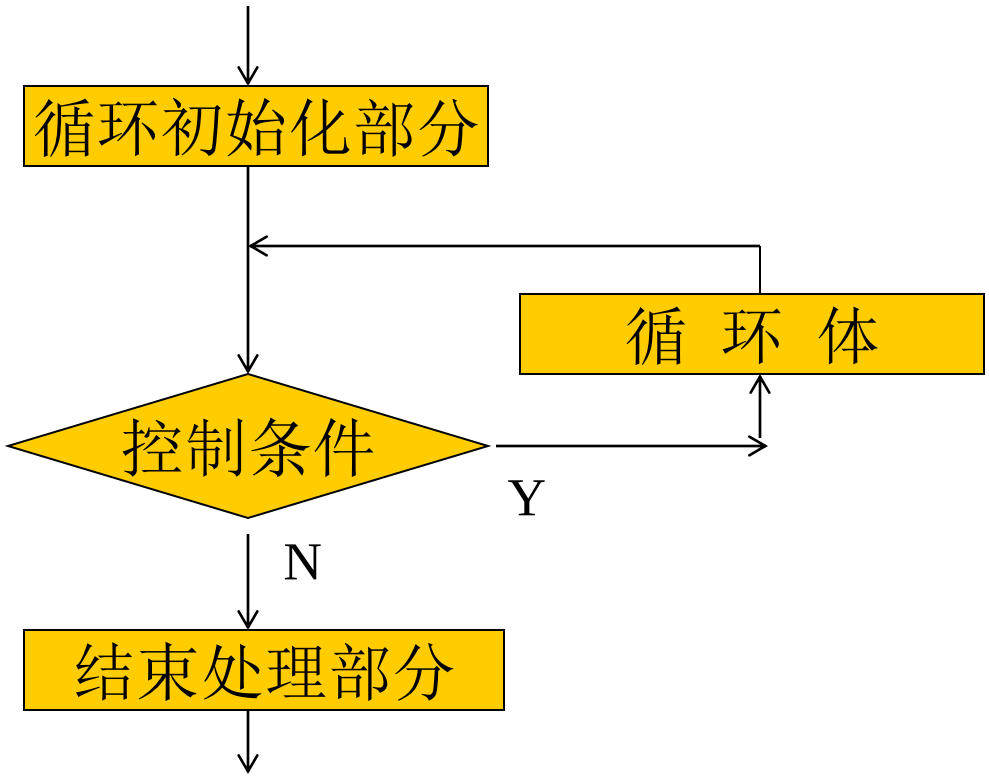
5.3 循环程序设计示例

循环程序有两种结构形式

1、先执行后判断结构



2、先判断后执行结构



3、循环控制部分

判断循环条件是否成立，可以有以下两种判断方法：

- 用计数控制循环——循环次数已知
- 用条件控制循环——循环次数未知

根据控制循环的条件分成两种情况：

- 计数控制循环——循环次数已知
- 条件控制循环——循环次数未知

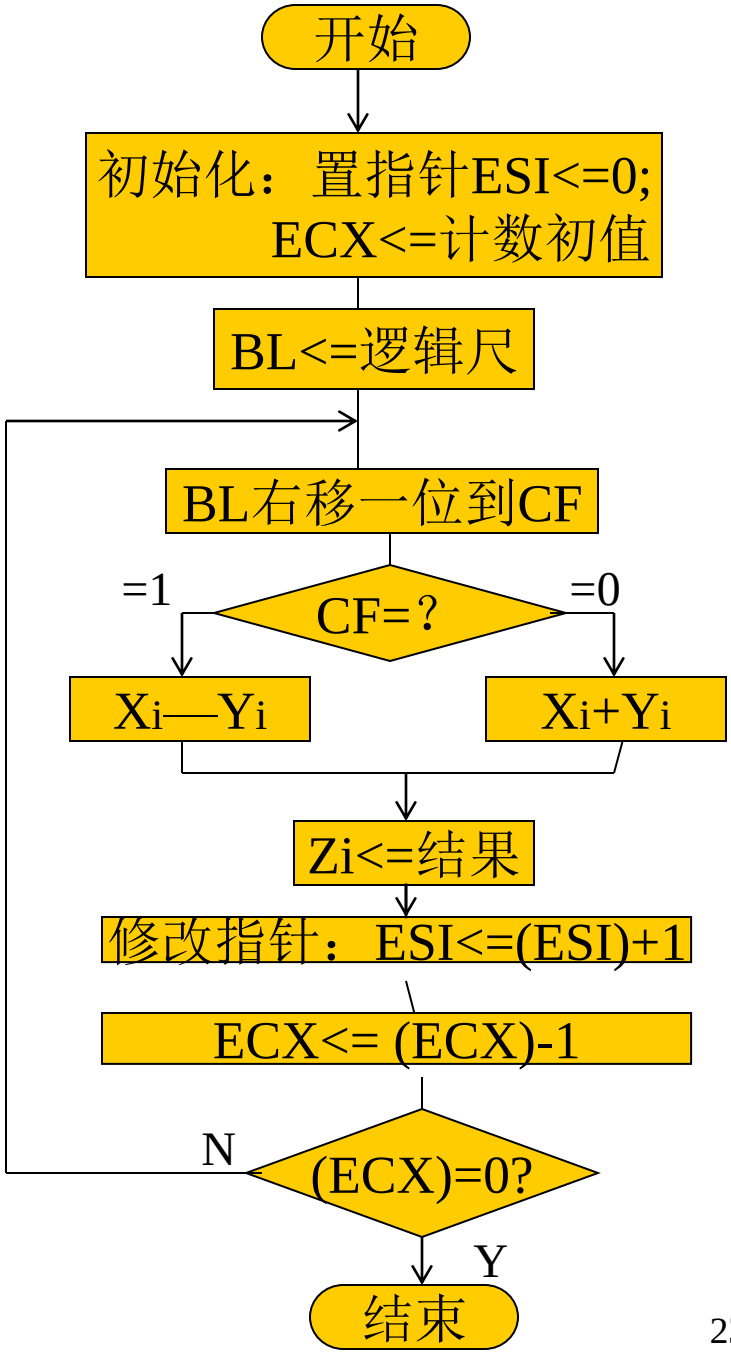
例5 设有两个数组X和Y，它们都有8个元素，其元素按下标从小到大的顺序存放在数据段中。试编写程序完成下列计算：

Z1=X1+Y1　Z2=X2-Y2　Z3=X3+Y3

Z4=X4-Y4　Z5=X5-Y5　Z6=X6+Y6

Z7=X7+Y7　Z8=X8-Y8

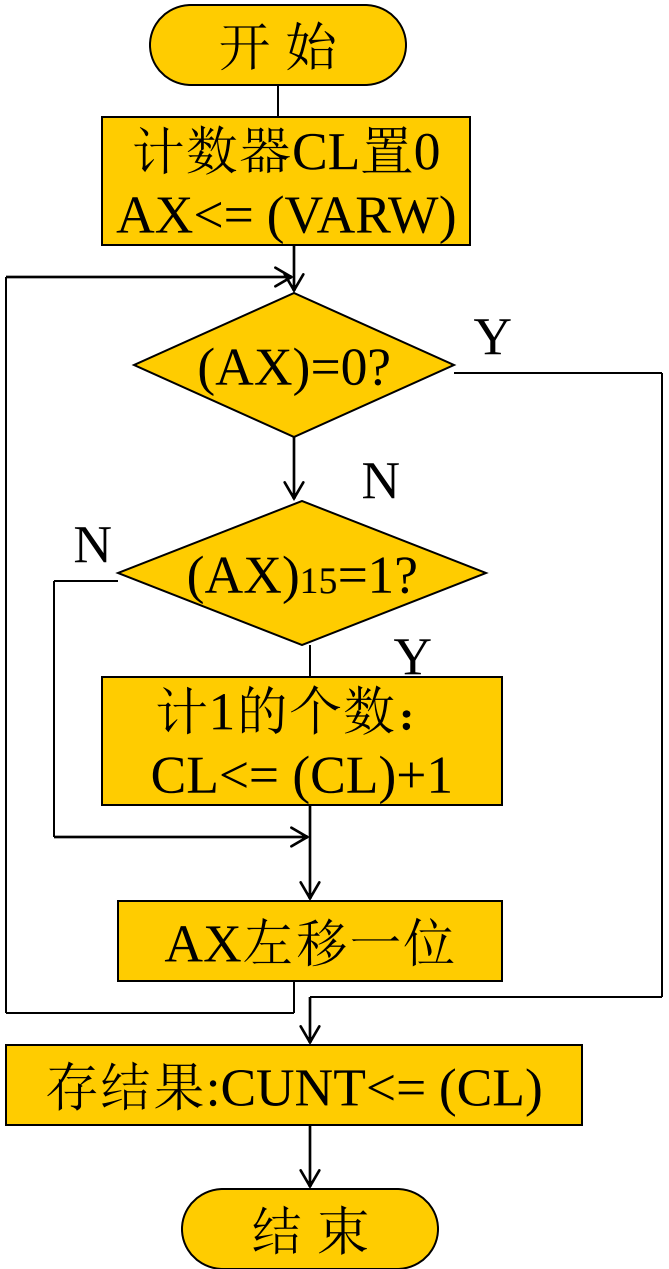
由于循环体中有“+”和“-”两种可能的运算，通过设置标志0(+)和1(-)来判断，低位表示低下标的运算。八个运算表达式由8位**逻辑尺**：10011010B来识别。



```
.DATA
X  DB 0A2H,7CH,34H,9FH,0F4H,10H,39H,5BH
Y  DB 14H,05BH,28H,7AH,0EH,13H,46H,2CH
LEN       EQU $ -Y
Z     DB  LEN DUP(?)
LOGR     DB  10011010B;设置标志0(+)和1(-)
.CODE
MAIN  PROC
        MOV ECX,LEN  ;初始化计数器
        MOV ESI,0    ;初始化指针
        MOV BL,LOGR ;初始化逻辑尺
LOP:   MOV  AL,X[ESI]
       SHR   BL,1        ;标志位送CF
       JC      SUB1      ;为1，转做减法
       ADD   AL,Y[ESI]  ;为0，做加法
       JMP    RES
SUB1:  SUB    AL,Y[ESI]
RES:   MOV   Z[ESI],AL ;  存结果
       INC    ESI          ;  修改指针
       LOOP   LOP
       INVOKE  ExitProcess, 0
MAIN  ENDP
       END   MAIN
```

例6 编写一程序，将字单元VARW中含1的个数(含1的个数是指用二进制表示时,有多少个1)统计出来，存入CONT单元中。

- 通过将字单元各位逐位移入最高位来判断。
- 为了减少循环次数，循环中加上了判断各位是否全为0，这样可使低位为全0时的循环次数减少。



```
.386
.MODEL flat,stdcall
.STACK  4096
ExitProcess PROTO,dwExitCode:DWORD
.DATA
VARW    DW   1101010010001000B
CONT     DB   ?
.CODE
MAIN PROC
          MOV  CL,0          ;初始值为0,统计1的个数
          MOV  AX,VARW
LOP:      TEST AX,0FFFFH  ;测试（AX）是否为0
          JZ   END0            ;为0，循环结束
          JNS  SHIFT           ;判最高位，为0则转SHIFT
          INC  CL              ;最高位为1，计数
 SHIFT: SHL AX,1
          JMP LOP
END0:   MOV   CONT,CL ;存结果
          INVOKE  ExitProcess, 0
MAIN    ENDP
          END   MAIN
```

5.4 子程序（子过程）设计示例

例 将两个给定的二进制数(16位和32位)转换为ASCII码字符串。

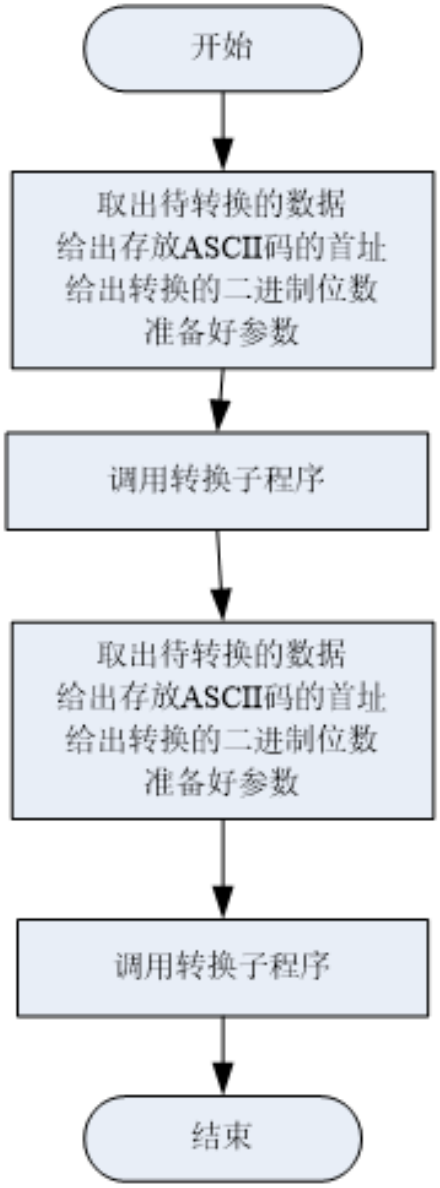
主程序提供被转换的数据和转换后的ASCII码字符串的存储区的首地址

子程序完成二进制数与ASCII码字符串的转换。子程序的入口参量有：被转换的数据、存储ASCII码字符串的首址和转换的位数。无出口参量。

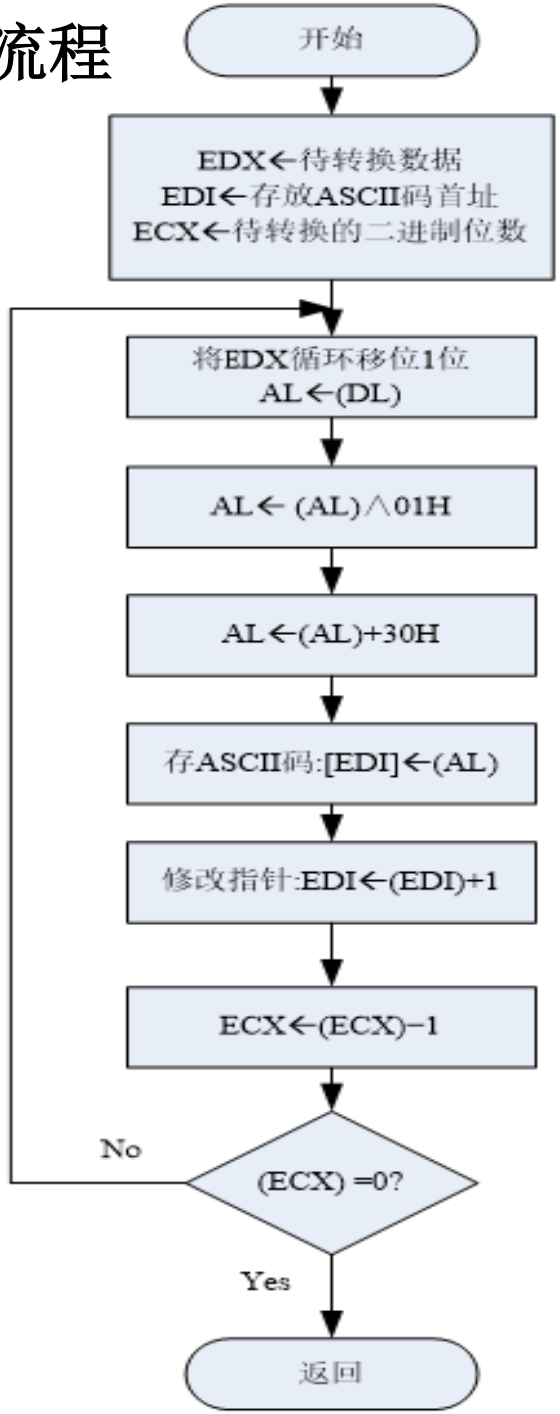
‘0’的ASCII码为30H，’1’的ASCII码为31H。

1. 用寄存器传递参量

主程序流程



子程序流程



```
.386
.MODEL flat,stdcall
.STACK  4096

ExitProcess PROTO,dwExitCode:DWORD

.DATA
        BIN1 DW 0F0F0H
        BIN2 DD 0F0F0F0F0H
        ASCBUF DB 30H DUP(0)
```

```
.CODE
BEGIN PROC
        XOR EDX,EDX
        LEA EDI, ASCBUF+15 ;存放ASCII码的单元末址送EDI
        MOV DX, BIN1 ;待转换的第1个16位数据送DX
        MOV EAX, 16 ;待转换的二进制数的位数送EAX
        CALL BINASC ;调用转换子程序

        MOV EDX, BIN2 ;待转换的第二个数据送EDX
        MOV EAX, 32
        LEA EDI, ASCBUF+2FH ;存放ASCII码的单元末址送EDI
        CALL BINASC

        INVOKE  ExitProcess, 0
BEGIN ENDP
```

```
BINASC PROC
        MOV ECX,EAX
LOP:  MOV AL, DL;最低位移入最低位
        AND AL, 1;保留最低位，屏蔽其它位
        ADD AL, 30H;AL中即为该数字符（0或1）的ASCII码
        MOV [EDI], AL;存结果
        ROR EDX, 1;转换的下一位移入最低位
        DEC EDI;修改地址指针
        LOOP LOP
        RET
BINASC ENDP
END BEGIN
```

2、用地址表传递参数

```
.386
.MODEL flat,stdcall
.STACK  4096

        ExitProcess PROTO,dwExitCode:DWORD

.DATA
        BIN1  DW  0F0F0H
        BIN2  DD  0F0F0F0F0H
        CUNT  DB  16, 32
        ASCBUF  DB  30H DUP(?)
        ADR_TAB DD  3  DUP(0)  ;存放参数地址表
```

```
.CODE
BEGIN PROC
        MOV ADR_TAB, OFFSET BIN1;存参数地址
        MOV ADR_TAB+4, OFFSET CUNT
        MOV ADR_TAB+8, OFFSET ASCBUF
        MOV EBX, OFFSET ADR_TAB;传表首址
        CALL  BINASC

        MOV ADR_TAB, OFFSET BIN2
        MOV ADR_TAB+4, OFFSET CUNT+1
        MOV ADR_TAB+8, OFFSET ASCBUF+16
        MOV EBX, OFFSET ADR_TAB;传表首址
        CALL BINASC

        INVOKE  ExitProcess, 0

BEGINE NDP
```

```
BINASC PROC
        MOV EDI, [EBX];取待转换数据
        MOV EDX, [EDI]
        MOV EDI, [EBX+4];取待转换数据位数
        XOR ECX, ECX
        MOV CL, [EDI]
        MOV EDI, [EBX+8];取存ASCII码首址
        ADD EDI, ECX;形成存SCII码的末地址后的一位
        DEC  EDI
LOP:    MOVAL, DL;待转换的1位送到AL中转换
        AND AL, 1
        ADD AL, 30H;构成相应的ASCII码
        MOV [EDI], AL;存结果
        ROR EDX, 1
        DEC EDI
        LOOP LOP
        RET
BINASCE NDP
END BEGIN
```

3.用堆栈传递参量

（1）用 call指令调用：过程无定义参数，无局部变量

如果使用堆栈传递参量，不使用伪指令，一般应包括：

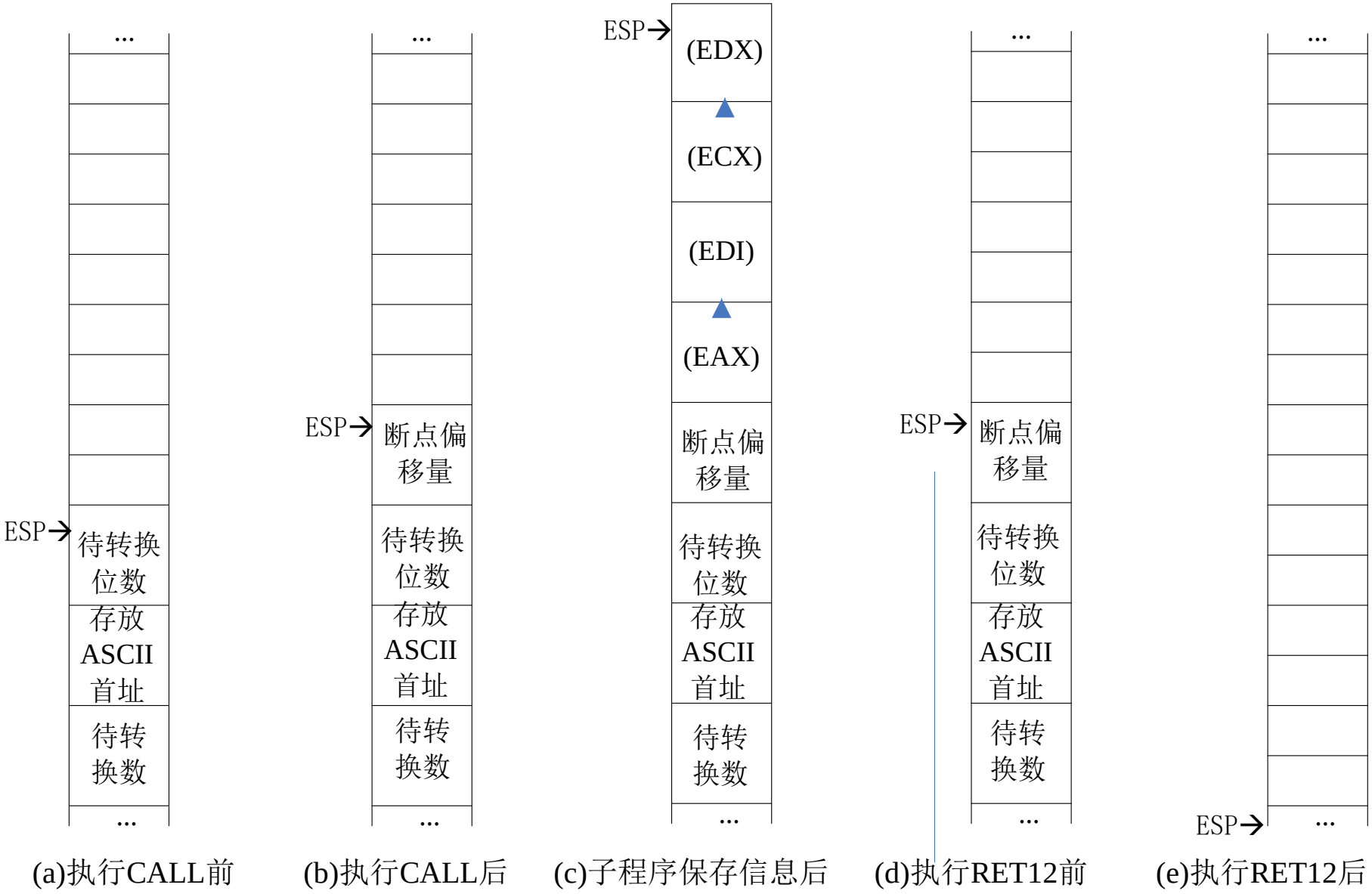
1）在主程序中，将待转换的数据、存放ASCII码的首址和转换的位数压入堆栈；

2）在子过程中保护和恢复寄存器的内容。

```
.386
.MODEL flat,stdcall
.STACK  4096
ExitProcess PROTO,dwExitCode:DWORD
.DATA
        BIN1 DW 0F0FOH
        BIN2 DD 0F0FOF0FOH
        ASCBUF  DB  30HDUP(?)
```

```
;主程序
.CODE
BEGIN PROC
        MOVZX EAX, BIN1
        PUSH EAX;待转换数据压栈
        MOV EAX, 16
        PUSH EAX;转换位数压栈
        LEA EDI,  ASCBUF+15
         PUSH EDI;存放ASCII码的末址压栈
        CALL BINASC;调用转换子程序
        MOV EAX, BIN2
        PUSH EAX
        MOV EAX, 20H
        PUSH EAX
        LEA EDI,  ASCBUF+2FH
        PUSH  EDI
        CALL  BINASC
        INVOKE  ExitProcess, 0
BEGINE  NDP
```

```
;转换子过程
BINASC  PROC
        PUSH  EAX
        PUSH  ECX
        PUSH  EDX
        PUSH  EDI
        MOV EBP, ESP
        MOV EDI, [EBP+20]
        MOV ECX, [EBP+24]
        MOV EDX, [EBP+28]
LOP:    MOV AL, DL
        AND  AL, 1
        ADD  AL, 30H
        MOV [EDI], AL
        ROR  EDX, 1
        DEC  EDI
        LOOP  LOP
        POP  EDI
        POP  EDX
        POP  ECX
        POP  EAX
        RET  12
BINASC   ENDP
END  BEGIN
```



用堆栈传递参数时堆栈的变化情况

（2）用 call指令调用：子过程用USES伪指令保护寄存器

在定义子过程时指定要保护的寄存器，就不需要用PUSH和POP指令保护和恢复寄存器内容。

```
.386
.MODEL flat,stdcall
.STACK  4096

ExitProcess PROTO,dwExitCode:DWORD

.DATA
        BIN1   DW     0FFFFH
        BIN2   DD     0FFFFFFFFH
        ASCBUF         DB30H          DUP(0)
```

```
;主程序
.CODE
BEGIN PROC
        MOVZX EAX, BIN1
        PUSH  EAX;待转换数据压栈
        MOV  EAX, 16
        PUSH   EAX;转换位数压栈
        LEA  EDI, ASCBUF+15
        PUSH  EDI;存放ASCII码的末址压栈
        CALL    BINASC;调用转换子程序
        MOV  EAX, BIN2
        PUSH  EAX
        MOV EAX, 20H
        PUSH EAX
        LEA  EDI, ASCBUF+2FH
        PUSH EDI
        CALL  BINASC
        INVOKE  ExitProcess, 0
BEGIN   ENDP
```

```
;转换子过程
BINASCPROC USES EAX ECX EDX EDI
        MOV EBP, ESP
        MOV EDI, [EBP+20];从堆栈取入口参数
        MOV ECX, [EBP+24]
        MOV EDX, [EBP+28];
LOP:   MOV AL, DL
        AND  AL, 1
        ADD  AL, 30H
        MOV [EDI], AL
        ROR  EDX, 1
        DEC  EDI
        LOOP  LOP
        RET 12;返回并从堆栈中弹出3个参数共12个字节
BINASCE  NDP
END   BEGIN
```

（3）用 invoke伪指令调用：子过程定义参数和USES保护

这种用法具有高级语言的风格。在有参数传递或者局部变量时，强烈建议使用这种方法，能够给程序带来可读性。

```
.386
.MODEL flat,stdcall
.STACK  4096

ExitProcess PROTO,dwExitCode:DWORD
BINASCPROTO,:dword,:dword,:dword
```

```
;主程序
.CODE
BEGIN PROC
LEAEDI, ASCBUF+15
MOV    EBX, 16
MOVZX EAX, BIN1
INVOKE BINASC, EDI,EBX,EAX;调用转换子程序

LEA EDI, ASCBUF+2FH
MOV EBX, 20H
MOV EAX, BIN2
INVOK  EBINASC,EDI,EBX,EAX

INVOKE  ExitProcess, 0
BEGINENDP
```

```
;转换子过程
BINASC PROC USES EAX ECX EDX EDI,pa1:dword,pa2:dword,pa3:dword
        MOV EDI, pa1;从堆栈取入口参数
        MOV ECX, pa2
        MOV EDX, pa3;
LOP:    MOV AL, DL
        AND AL, 1
        ADD AL, 30H
        MOV [EDI], AL
        ROR EDX, 1
        DE CEDI
        LOOP LOP
        RET 12;返回并从堆栈中弹出3个参数共12个字节
BINASC ENDP

END BEGIN
```

作业：5.1，5.3，5.7，5.11