第3章 汇编语言程序设计的基本技术

在汇编语言程序中，最常见的形式有顺序程序、分支程序、循环程序和子程序。这几种程序的设计方法是汇编语言程序设计的基础。本章将结合实例详细地介绍这些程序的设计技术以及第2章尚未讲述的指令。

3.1 顺序程序设计

顺序程序是最简单的程序，它的执行顺序和程序中指令的排列顺序完全一致。下面先介绍乘除法指令及十进制运算的调整指令。

3.1.1 乘除法指令

乘除法指令应该有无符号数乘除法指令和符号数乘除法指令之分。这是因为乘除法不同于加减法，无符号数的乘法和除法指令对符号数进行乘除运算不能得到正确的结果。如FFH乘以FFH结果为FE01H。把它们看作无符号数为255×255＝65 025(FE01H＝65 025)，其结果是正确的，若把它们看作符号数为(－1)×(－1)＝－511(FE01H＝－511)，显然是错误的。MCS-51的乘除法指令只有无符号数的乘除法指令。

1. 乘法指令 MUL

指令格式：

```
MUL AB
```

乘法指令(Multiply)所执行的操作是A乘以B，乘积放回到B和A(高位字节放在B中，低位字节放在A中)，如图3-1所示。

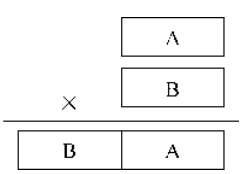

图3-1 乘法指令的操作

乘法运算一定不产生进位，因此进位标志位CY总是被清零；溢出标志位与乘积有关，若乘积小于0100H(即B的内容为0)则溢出标志位OV被清零(OV＝0)，否则OV置1(OV＝1)。

2. 除法指令 DIV

指令格式：

```
DIV AB
```

除法指令(Divide)所执行的操作是用B除A，商放入A，余数放入B，如图3-2所示。

除法运算和乘法运算一样，一定不产生进位，因此进位标志位CY总是被清零；溢出标志位OV则反应除数情况，当除数为零(B＝0)时，OV置1，其他情况OV均被清零。

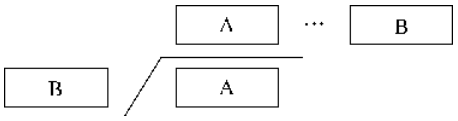

图 3-2　除法指令的操作

3.1.2　BCD 数加法调整指令 DAA

第 2 章第 2.2 节介绍的加减指令和本节介绍的乘除指令都是对二进制数进行操作，其结果也是二进制数。二进制算术指令对 BCD 数进行运算，会得到一个非 BCD 数或不正确的 BCD 数。如：

0000 0011B＋0000 1001B＝0000 1100B

0000 1001B＋0000 0111B＝0001 0000B

第一个结果是非 BCD 数；第二个结果是不正确的 BCD 数。其原因是 BCD 数向高位的进位是逢 10 进 1，而 4 位二进制数向其高位的进位是逢 16 进 1，中间相差 6。若再加上 6，就可以得到正确的 BCD 数：

00001100B＋00000110B＝ 00010010B

00010000B＋00000110B＝ 00010110B

计算机对 BCD 数的运算是首先使用二进制数算术运算指令进行运算，然后执行一条能把算术运算结果转换成正确的 BCD 数的专用调整指令来处理 BCD 数的运算。

指令格式：

```
DAA
```

DAA 指令的意义是将 A 中的数当作两个 BCD 数相加之和来进行调整，得到两位 BCD 数。具体操作是，若(A & 0FH) ＞9 或 AC＝1，则 A 加上 6；若(A & 0F0H) ＞90H 或 CY＝1，则 A 加上 60H。

由调整指令所执行的具体操作可以看到，对结果进行调整时要用到进位标志和辅助进位标志，所以调整指令应紧跟在 BCD 数作为加数的加法指令之后。所谓“紧跟”是指在调整指令与加法指令之间不得有改变 CY 和 AC 标志位的指令。

MCS-51 单片机仅有一条十进制数加法调整指令，只能对加操作进行调整，不能对乘操作和除操作进行调整。要对减操作进行调整，须采用补数相加的办法：先用(99＋1)H 即 9AH 减去减数得其补数，再与被减数进行加操作，实现调整。如：

99－56＝43

先用 9AH 减去 56H，得补数 44H，再用被减数 99H 加上补数 44H 得 DDH。加调整指令将 DDH 加上 66H 得 43H，得到正确的结果。

实现累加器 A 中的 BCD 数减 1 的指令为

```
ADD A,＃99H
DA A
```

如 A 中有一 BCD 数 40H，执行第一条指令后 A＝D9H；因为 A 的高 4 位大于 9 故执行第 2 条指令时应加 60H，调整后 A＝39H。

【例 3-1】 将内部 RAM 40H 单元中一压缩 BCD 数转换为二进制数。

把 40H 单元中的十位 BCD 数乘以 10 再加上个位 BCD 数，即可实现转换，结果为一个小于 64H 的 1 字节无符号二进制数。如 40H 中有一压缩 BCD 数 98H，9 乘以 10，再加上 8 的二进制运算如下：

```
  00001001      01011010
×00001010    +00001000
  0001001      01100010
 +01001
 01011010
```

即 09×0AH+08H=62H

实现转换的程序段如下：

```
MOV A,40H       ;将 BCD 数的十位放 A 的低 4 位
SWAP A
ANL A,#0FH
MOV B,#10
MUL AB          ;十位乘以 10
ANL 40H,#0FH    ;将 BCD 数的个位保留在 40H 单元中
ADD A,40H       ;十位×10+个位
MOV 40H,A
```

【例 3-2】 将内部 RAM 30H 单元中的二进制数(小于 64H)转换为 BCD 数。

可以用除 10 取余法，将二进制数转换为 BCD 数。如 A 中有一个二进制数 62H，62H 除以 10 的运算如下：

```
          1001
      ------------
1010 ) 01100010
       1010
      ------
        10010
         1010
      ------
         1000
```

即 01100010B÷1010B 商为 1001B，余数为 1000B。商 1001B 再除以 10 即可得到第 2 个余数，把第 2 个余数即第一次除法的商(高位)和第一个余数(低位)拼成一个字节便可得到 BCD 数 10011000B。

实现转换的程序段如下：

```
MOV A,30H
MOV B,#10
DIV AB
SWAP A                          ;将商放 A 的高 4 位
ADD A,B                         ;加上余数
MOV 30H,A
```

【例 3-3】 将内部 RAM 20H、21H 和 22H、23H 中的 2 个 4 位压缩 BCD 数相加，结果送 24H、25H、26H 中。

4 位压缩 BCD 数相加，分成高 2 位 BCD 数和低 2 位 BCD 数相加，先加低 2 位 BCD 数。程序段如下：

```
MOV  A,21H
ADD  A,23H                          ;BCD 数十位和个位相加
DA   A
MOV  26H,A                          ;存结果的十位和个位
MOV  A,20H
ADDC  A,22H                         ;BCD 数千位和百位相加
DA   A
MOV  25H,A                          ;存结果的千位和百位
CLR  A
RLC  A                              ;将 4 位 BCD 数相加的进位(即万位)移入 A 中
MOV  24H,A                          ;存结果的万位
```

【例 3-4】 求内部 RAM 20H、21H 中的 4 位压缩 BCD 数与 22H、23H 中的 4 位压缩 BCD 数的差，结果送 24H、25H 中。

先用(9999＋1)H 减去减数得其补数，再加被减数即得。程序段如下：

```
CLR  C
MOV  A,＃9AH
SUBB  A,23H                         ;一定无借位,CY = 0
MOV  25H,A                          ;暂存补数低位(即十位和个位)
MOV  A,＃99H
SUBB  A,22H
MOV  24H,A                          ;暂存补数高位(即千位和百位)
MOV  A,25H
ADD  A,21H
DA   A
MOV  25H,A                          ;存结果的十位、个位
MOV  A,24H
ADDC  A,20H
DA   A
MOV  24H,A                          ;存结果的千位、百位
```

3.1.3 顺序程序设计举例

【例 3-5】 设一个非压缩 BCD 数(0～9)放在内部 RAM 60H 单元中，求其平方值并放入内部 RAM 61H 单元中。

求一个数的平方值可以用乘法运算实现，也可以用查表法实现。查表法运算速度比较快，可避免复杂的计算和编程，是常用的计算方法，应熟练掌握。将 BCD 数 0～9 的平方值按顺序建立一平方表，放入程序存储器的一片连续区域内。0～9 的平方值的最大值为 81，用 BCD 数来建造，每个平方值均为 1 字节。所以平方表的首址与任意 BCD 数之和正是该 BCD 数的平方值的存放地址。程序段如下。

```
      MOV  DPTR,＃SQTAB
      MOV  A,60H
      MOVC  A,@A＋DPTR
      MOV  61H,A
      ⋮
SQTAB: DB 0,1,4,9,16H,25H,36H,49H,64H,81H
```

【例 3-6】 编程实现图 3-3 所示的逻辑功能，其中 20H、2FH 和 37H 是位地址中的信息。

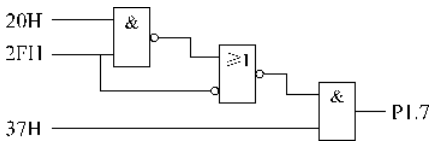

图 3-3 例 3-6 的逻辑运算电路

图中的逻辑功能是：

$$P1.7=\overline{\overline{20H \cdot 2FH}+\overline{2FH}} \cdot 37H$$

将该式化简得：

$$P1.7=20H \cdot 2FH \cdot 37H$$

故实现图 3-3 的逻辑功能的程序段如下：

```
MOV  C ,20H
ANL  C ,2FH
ANL  C ,37H
MOV  P1.7,C
```

【例 3-7】 编写双字节无符号数乘法程序。两个乘数分别放在 R0R1 和 R2R3 中，乘积放入 R4R5R6R7 中。

因为 R0R1×R2R3=(2^8R0+R1)×(2^8R2+R3)=2^{16}R0×R2+2^8(R0×R3+R1×R2)+R1×R3，所以 R0×R2 应放入 R4R5 中，R1×R3 应放入 R6R7 中，R0×R3+R1×R2 应放入 R5R6 中。程序段如下：

```
MOV A,R1                          ;R1 × R3
MOV B,R3
MUL AB
MOV R6,B                          ;部分积 R1 × R3 放 R6R7
MOV R7,A
MOV A,R0                          ;R0 × R3
MOV B,R3
MUL AB
ADD A,R6                          ;部分积 R0 × R3 放 R5R6
MOV R6,A
MOV A,B
ADDC A,＃0
MOV R5,A
MOV A,R1                          ;R1 × R2
MOV B,R2
MUL AB
ADD A,R6                          ;部分积 R1 × R2 放 R5R6
MOV R6 A
MOV A,B
ADDC A,R5
MOV R5,A
CLR A
ADDC A,＃0                        ;部分积 R1 × R2 + R0 × R3 产生的进位放入 R4
MOV R4,A
MOV A,R0                          ;R0 × R2
MOV B,R2
MUL AB
ADD A,R5                          ;部分积 R0 × R2 放 R4R5
MOV R5,A
```

```
MOV A,B
ADDC A,R4
MOV R4,A
```


3.2 分支程序设计

顺序程序的特点是从程序的第一条指令开始，按顺序执行，直到最后一条指令。然而，许多实际问题并不能设计成顺序程序，需要根据不同的条件作出不同的处理。把不同的处理方法编制成各自的处理程序段，运行时由计算机根据不同的条件自动作出选择判别，绕过某些指令，仅执行相应的处理程序段。按这种方式编制的程序，执行的顺序与指令存储的顺序失去了完全的一致性，称之为分支程序。分支程序是单片机利用改变累加器的状态和位状态的指令和转移指令或者直接用比较不等转移指令来实现的。

转移指令有无条件转移、条件转移和比较不等转移 3 类。无条件转移指令将控制转向其后的目的标号指定的地址。条件转移指令伴随着能改变累加器 A 的状态和位状态的指令之后，根据设置条件决定是否将控制转向其后的目的地址。比较不等转移指令把指令中的目的操作数和源操作数进行比较，它们不相等将控制转向其后的目的地址。

3.2.1 条件转移指令

指令格式：

```
Jcond [操作数,] short - lable
```

该指令的功能是，若条件满足则转向短标号 short-lable 运行，否则顺序运行程序。

条件转移指令是相对转移指令，相对转移指令实现转移的范围为一个字节的偏移量，该偏移量即是附录的指令系统表中的机器指令代码 rel。偏移量 rel 的大小从当前地址（执行该指令时的 PC 值，也就是该相对转移指令的下一条指令的地址）到短标号 short-lable 地址为－128～127，从相对转移指令的地址到短标号 short-lable 地址则为－126～129。偏移量 rel 是一补码数，若其为负数则是向低地址转移，若其为正数则是向高地址转移。偏移量 rel 的大小为：

偏移量 rel＝short-lable 地址－当前地址

或

偏移量 rel＝short-lable 地址－（相对转移指令地址＋相对转移指令字节数）

相对转移指令执行的操作是：条件成立，PC 的当前地址加上偏移量 rel，亦即短标号 short-lable 地址送 PC；条件不成立，不进行上述加操作，即执行相对转移指令的下一条指令。

相对转移指令的机器指令代码有 2 字节和 3 字节（较 2 字节多 1 字节的位地址，反映在符号指令就是有操作数）两种，都是最后一个字节为偏移量 rel。偏移量 rel 由汇编程序汇编时自动计算填入。

1. 累加器 A 状态转移指令

这是根据累加器 A 的内容（A 自身带零标志）为零或不为零实现转移的相对转移指令。该指令有如下 2 条：

```
JZ short - lable
JNZ short - lable
```

2. 位状态转移指令

这是根据进位 CY 或位地址位等于 1 或 0 实现转移的相对转移指令，指令格式及功能如表 3-1 所示。

表 3-1　位状态转移指令

指　令	功　能
JC short-lable	CY＝1 转移，去 short-lable
JNC short-lable	CY＝0 转移，去 short-lable
JB bit，short-lable	bit＝1 转移，去 short-lable
JNB bit，short-lable	bit＝0 转移，去 short-lable
JBC bit，short-lable	bit＝1 转移，去 short-lable，并使 bit＝0

3.2.2　比较不等转移指令

指令格式：

```
CJNE dest,sourse,short - lable
```

CJNE 指令执行的操作是把目的操作数和源操作数进行比较，若它们不相等则转移去 short-lable；否则顺序执行。比较不等转移指令是 3 字节的相对转移指令。

CJNE 指令有累加器 A 与内部 RAM 或 SFR 比较不等转移、累加器 A 与立即数比较不等转移、通用寄存器 Rn 与立即数比较不等转移以及 Ri 间址的内部 RAM 与立即数比较不等转移。所以 CJNE 指令有如下 4 条指令：

```
CJNE A,direct,short - lable
CJNE A,#data,short - lable
CJNE Rn,#data,short - lable
CJNE @Ri,#data,short - lable
```

比较不等转移指令的操作，要影响进位标志位 CY；若 dest≥source，无借位，CY＝0；若 dest＜source，有借位，CY＝1。MCS-51 单片机没有比较指令，要比较两个数的大小，可以用如下 2 条指令实现。

```
      CJNE dest,source,NEXT
NEXT: JNC short - lable 或 JC short - lable     ;等不等都执行该条件转移指令
```

例如，将累加器 A 中的一位十六进制数转化为其 ASCII 码程序段的混合方式（即指令地址、机器指令代码和符号指令）如下。

```
0000  2430          ADD A,#30H
0002  B43A00        CJNE A,#3AH,NEXT
0005  4002    NEXT: JC NA7
0007  2407          ADD A,#7
0009          NA7:
```

该程序的第 2 条指令和第 3 条指令都是相对转移指令。第 2 条机器指令代码的第 3 字节 00H 是偏移量。该指令是 3 字节指令，其指令地址是 0002H，当前地址是 0005H，0005H＝0002H＋3。由此可见，当前地址＝相对转移指令地址＋相对转移指令字节数。该指令执行的操作是，若 A≠3AH，则 PC＝NEXT 的地址＝0005H；若 A＝3AH，则 PC＝当前地址＋偏移量＝0005H＋0000H＝0005H。由此可见，累加器 A 的内容等不等于 3AH 都执行第 3 条指令。所以，此处第 2 条指令的作用是设置进位标志位，第 3 条指令则根据进位标志位实现转移或者不转移。第 3 条机器指令代码的第 2 字节是 0002H。该指令是 2 字节指令，其指令地址是 0005H，当前地址是 0007H，short-lable 地址是 0009H，0002H(偏移量)＝0009H(short-lable 地址)－0007H(当前地址)。

为了书写方便，也可以将比较两个数大小的两条指令写为如下形式：

```
CJNE dest,source, $ + 3                    ; $ 为本指令的地址, $ + 3 为下条指令的地址
JNC short - lable 或 JC short - lable
```

套用偏移量计算公式：CJNE dest，source，$ ＋3 指令的转移地址是 $ ＋3，指令地址是 $，指令字节数是 3，该指令的偏移量为 0。执行 CJNE dest，source，$ ＋3 指令时，程序计数器 PC 的当前值是它下一条指令的地址。程序计数器 PC 的当前值与该指令的偏移量相加，其和就是它下一条指令的地址。因而，无论目的操作数与源操作数比较的结果是等还是不等都将执行它的下一条指令。它的下一条指令将根据它的比较结果实现转移或者执行再下一条指令。

3.2.3 无条件转移指令

无条件转移是不规定条件的程序转移。无条件转移指令有长转移、绝对转移、变址转移和短转移 4 类。条件转移指令的移动范围为－128～127，若超过这个范围，就要在这个范围的某处放一条无条件转移指令来实现转移。无条件长转移和变址转移指令没有范围限制。在分支程序中还要用无条件转移指令将各分支又重新汇集到一起。

1. 长转移指令 LJMP

指令格式：

```
LJMP target
```

指令的功能是将程序转向目的标号 target。该指令转移的范围大，可达 64KB，因此称之为“长转移”。长转移指令是 3 字节指令，依次是操作码、转移地址的高 8 位地址和低 8 位地址。

2. 绝对转移指令 AJMP

指令格式：

```
AJMP target
```

指令的功能与 LJMP 相同，差别是该指令的转移范围只有 2KB。绝对转移指令是 2 字节指令，其中 5 位是操作码，11 位是转移地址。该指令执行的操作是以指令提供的 11 位地址去替换程序计数器 PC 的低 11 位(高 5 位不变)，形成新的 PC 值，此即是转移的目的地址。

3. 变址转移指令

指令格式：

```
JMP @A + DPTR
```

变址转移指令的转移地址由 A 的内容和 DPTR 的内容之和来确定，即转移的目的地址＝A＋DPTR。该指令以 DPTR 的内容为基地址，A 的内容作变址。因此只要把 DPTR 的值固定，而给 A 赋以不同的值，即可实现程序的多分支转移。如有多个分支程序，这些分支程序都可以通过绝对转移指令 AJMP 进行转移，把这些转移指令按序填入转移表中。假定分支序号值在 R7 中，则可以用如下的程序段实现多分支转移。

```
        MOV   A,R7
        CLR   C                          ;分支序号值乘以 2
        RLC   A
        MOV   DPTR,#JPTAB                ;转移表首址送 DPTR
        JMP   @A + DPTR                  ;实现散转
JPTAB:  AJMP  ROUT0                      ;转分支程序 ROUT0
        AJMP  ROUT1                      ;转分支程序 ROUT1
        ⋮
        AJMP  ROUTN                      ;转分支程序 ROUTN
ROUT0:  ⋮
ROUT1:
        ⋮
ROUTN:
        ⋮
```

其中 CLR C 指令和 RLC A 指令把分支序号值乘以 2 的原因是 AJMP 指令为 2 字节指令。分支实现过程是根据分支序号值，通过 JMP @A＋DPTR 指令转向 JPTAB 表中的某一条 AJMP 指令把程序转移到指定的分支程序。因此这种分支实际上是通过两次转移而实现的。使用这种方法最多可实现 128 个分支的程序的转移，所以 ROUTN 中的 N 最大取值为 127。

4. 短转移指令 SJMP

指令格式：

```
SJMP short - lable
```

短转移指令是 2 字节的相对转移指令。指令的功能是将程序无条件转向短标号 short-lable。其机器指令操作码是 80H，操作数是偏移量 80H～7FH，所以该指令的转移范围为－128～127。

SJMP 指令常用来使程序原地踏步，其指令如下。

```
HERE: SJMP HERE 或 SJMP   $              ;$ 是汇编地址计数器，代表 PC 的当前值
```

该指令的机器代码是 80 rel，偏移量 rel 为：HERE－(HERE＋2)＝－2＝FEH，所以 SJMP $ 指令的机器码为 80FEH。在仿真器上调试程序时，使用 SJMP $ 作为程序段的结束。

3.2.4 应用举例

【例 3-8】 设补码数 X 放在内部 RAM 30H 单元中，函数 Y 与 X 有如下的关系式：

$$Y=\begin{cases} X, & X>0 \\ 20H, & X=0 \\ X+5, & X<0 \end{cases}$$

试编写程序，根据 X 的值求出 Y，并放回原单元。

取出 X 后先作取值范围的判断，用累加器 A 状态转移指令判断 X 是否为 0，用位状态转移指令判断 X 是大于 0 还是小于 0。程序流程图如图 3-4 所示。

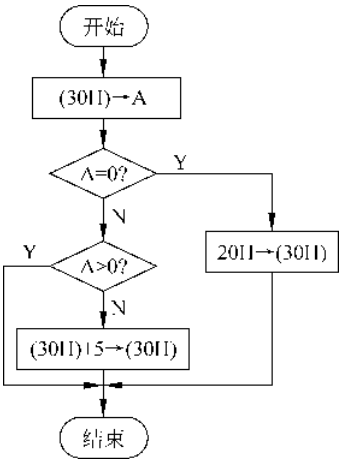

图 3-4 例 3-8 程序框图

程序如下。

```
        MOV A, 30H
        JZ ZERO
        JNB ACC. 7, PLUS
        ADD A, #5
        MOV 30H, A
PLUS:   SJMP $
ZERO:   MOV  30H, #20H
        SJMP $
        END
```

【例 3-9】 设有 3 个分支程序段，各分支程序段的功能依次是：从内部 RAM 取数、从外部 RAM 低 256 字节范围内取数和从外部 RAM 64KB 范围内取数。R0 中存内部 RAM 地址或外部 RAM 的低 8 位地址，R1 中存外部 RAM 的高 8 位地址，R3 中存放分支序号值。

在程序中建立一个差值表 BRTAB，将 3 个分支的入口地址与该表首地址的差值按序排列其中。并将差值表首地址送 DPTR，分支序号值送 A，查差值表就可以得到分支序号值确定的某一分支入口地址与差值表首地址的差值，变址转移指令将差值与差值表首址相加的入口地址送 PC，实现转移。程序如下。

```
        MOV  DPTR , #BRTAB
        MOV  A , R3
        MOVC  A , @A + DPTR                    ;查差值表
        JMP  @A + DPTR                         ;转移
BRTAB: DB BR0 - BRTAB, BR1 - BRTAB, BR2 - BRTAB  ;差值表
BR0:    MOV  A , @R0
        SJMP $
BR1:    MOVX  A , @R0
        SJMP $
BR2:    MOV  DPH , R1
        MOV  DPL , R0
        MOVX  A , @DPTR
        SJMP $
        END
```

使用差值表法，转移表的大小加上各转向程序，长度必须小于 256 字节。本例还可以使用地址表法，地址表法就没有差值表的限制，可以实现 64KB 范围内的转移。程序如下。

```
        MOV DPTR,#BRTAB
        MOV A,R3                ;分支序号值×2 放 A 中
        ADD A,R3
        JNC NADD                ;乘 2 操作的进位加入 DPH 中
        INC DPH
NADD:   MOV B,A                 ;暂存分支序号值×2
        MOVCA,@A+DPTR           ;取转移地址的高 8 位
        XCH A,B                 ;转移地址高 8 位送 B
        INC A                   ;分支序号值×2+1
        MOVC A,@A+DPTR          ;取转移地址的低 8 位
        MOV DPL,A
        MOV DP ,B
        CLR A
        JMP @A+DPTR
BRTAB: DW BR0,BR1,BR2
BR0:    ⋮
BR1:    ⋮
BR2:    ⋮
        END
```

【例 3-10】 内部 RAM 40H 和 41H 单元中各有一无符号数，比较其大小，将大数存放于内部 RAM 60H 单元，小数存放于内部 RAM 61H 单元，如两数相等，则分别送往这 2 个单元。

用比较不等转移指令 CJNE 比较这 2 个无符号数，先确定它们是否相等，若不相等时再根据借位标志确定这 2 个数的大小。程序框图如图 3-5 所示。

程序如下。

```
        MOV A,40H
        MOV 61H,41H
        CJNE A,41H,$+3
        JNC AGEQ                ;A≥(41H) 则无借位
        XCH  A ,61H             ;A<(41H)有借位,A 与(61H)交换
AGEQ:   MOV  60H ,A
        SJMP $
        END
        END
```

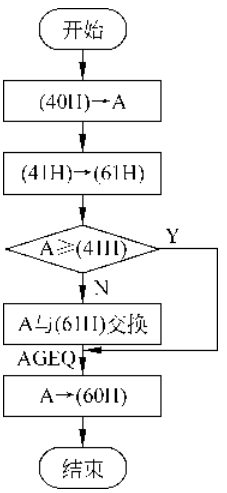

图 3-5　例 3-10 程序框图

3.3　循环程序设计

顺序程序和分支程序中的指令，最多只执行一次。在实际问题中重复地做某些事的情况是很多的，用计算机来做这些事就要重复地执行某些指令。重复地执行某些指令，最好用循环程序实现。