第 4 章

MCS-51 单片机汇编语言程序设计

本章知识点：

- 单片机汇编语言的格式
- 单片机汇编语言中的伪指令语句
- 单片机汇编语言编程步骤
- 常用的汇编语言程序设计结构

基本要求：

- 理解单片机汇编语言程序设计的过程
- 掌握单片机的伪指令
- 掌握单片机汇编语言常用的设计结构

能力培养目标：

通过本章的学习，掌握单片机汇编语言程序设计与编程能力，学习单片机常用的汇编语言程序结构，理解单片机程序的运行原理，提高单片机汇编语言的编程能力，并通过实例培养动手能力和工程实践能力。

4.1 汇编语言程序设计概述

4.1.1 汇编语言语句和格式

1. 汇编语言

单片机只能识别用二进制表示的指令，即机器指令或机器码。机器码非常不好记，容易出错，给编程带来了很大难度。为了解决这一问题，人们创造了一种用英文字母表示的助记符，用来反映指令的功能和主要特征，以此来代替机器指令，这就是汇编语言。汇编语言比机器语言好记，不易出错，并且和机器语言有一一对应的关系。这种对应关系，在前两节的指令系统中已经表述得非常清楚了。用汇编语言编写的程序称为汇编语言程序。汇编语言程序必须翻译成机器码才能执行。将汇编语言程序翻译成机器码的过程称为汇编。不同计算机的汇编语言不尽相同，这里介绍 MCS-51 单片机的汇编语言。

2．汇编语言的语句格式

汇编语言由汇编语句组成，这些语句在书写上有一定的规定和格式要求。我们必须严格按照汇编语言的语句格式编写程序。

一条汇编语言的语句包括 4 部分内容：标号、操作码、操作数和注释。其格式为：

标号：操作码　操作数；注释

1）标号

标号位于语句的开始，由 1 ～ 8 个字母和数字组成，它代表该语句的地址。标号必须由字母打头，以冒号结尾，标号不能使用指令助记符、伪指令或寄存器名。标号不是语句的必要组成部分，只在需要时才使用。

2）操作码

操作码是指令的助记符，表示语句的性质，不可省略，它是语句的核心部分。

3）操作数

操作数与操作码之间用空格分开。操作数一般有目的操作数和源操作数，操作数之间用逗号分开。操作数可以是立即数，也可以是地址，但必须满足寻址方式的规定。对于 MCS-51 单片机的 111 条汇编语言指令，大多数指令有两个操作数，但有的指令只有一个操作数，如 CLR A。空操作指令 NOP 无操作数。操作数中的常数可以是二进制数、八进制数、十进制数、十六进制数和字符串常数。二进制数以 B 结尾；八进制数以 Q 结尾；十进制数以 D 结尾，也可省略；十六进制数以 H 结尾；字符串用单引号‘’引用。

4）注释

注释是用户为方便阅读程序而加的说明，可有可无。注释与操作数之间用分号；隔开。

例如：

```
LOOP: MOV      A , #20H   ;   (A)←20H
```

4.1.2　汇编语言程序的设计步骤

程序是指令的有序集合。一个好的程序不仅应该完成规定的任务，而且应该层次清晰，易于阅读，尽可能少占内存，并且执行时间应尽可能短，以满足解决实际问题的需要。但也不要一味地追求少占内存，缩短执行时间。这样做可能会使程序的易读性变差。随着大规模和超大规模集成电路的发展，芯片的内存容量在不断增加，计算机执行指令的平均时间在大大缩短。因此，程序的长短和执行时间不再显得那么重要，而程序的易读性和程序的开发周期显得越来越重要。

程序设计一般分为以下几个步骤。

1．分析任务，确定算法或解题思路

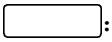
这一步是能否编制出高质量程序的关键，因此，不要一拿到任务就急于编写程序，而应该仔细地分析任务，找出合理的算法和解决思路。

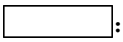
2．根据算法和解决思路画出程序流程图

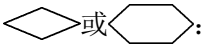
流程图给出了程序结构，直观地表示程序的执行过程。它将复杂的程序分成若干较简单的

部分，为编程带来方便。流程图还充分表达了程序设计思路，将问题与程序联系起来，有助于阅读程序。对于初学者来说，画流程图特别有助于程序设计，并可减少出错。画流程图时，并不一定要一气呵成，对于较复杂的问题，可以从粗到细，把算法和思路逐步细化。

流程图由以下一些框图和流程线组合而成。

（1）开始框和结束框  ：表示程序的开始或结束。

（2）处理框  ：表示某种处理过程、完成一定功能。

（3）判断框  ：根据不同的判断结果，执行不同的分支程序。

（4）流向线↓、→：表示程序执行的前进方向。

（5）连接圈 ©：用于连接出现在不同位置的框图。欲将不同位置的框图连接在一起，位于不同框图的圈内应标注相同的字母。

3．根据流程图编写程序

根据流程图中细化的各部分功能编写出具体的程序，然后，由流程图各部分之间的关系整理出全部程序。

4．上机调试程序

任何程序都必须上机调试，才能验证其是否正确。在调试过程中，应该善于利用计算机提供的调试工具（如 DEBUG）。调试工具会提供很大的帮助。

4.1.3　伪指令

伪指令是仅在汇编时起作用、机器执行时不起作用的指令。伪指令只提供汇编控制信息，如规定程序存放的首地址，为源程序预留存储区，以及规定汇编语言程序何时结束等。伪指令没有相对应的机器码，不是 CPU 能执行的指令，在机器执行时不产生机器动作。下面介绍常用的伪指令。

1．定位伪指令

格式：

```
ORG   n
```

ORG 规定紧接其后的程序或数据块的起始地址。ORG n 规定其后的程序或数据块从地址 n 开始存放。n 可以是十进制常数，也可以是十六进制常数（一般使用十六进制常数）。

例如，下面的程序从 1000H 开始存放。

```
        ORG   1000H
        MOV   R2 , #08H
        MOV   R0 , 30H
        MOV   R1 , 40H
LOOP: MOV   A , @R0
        MOV   @R1 , A
        INC    R0
        INC    R1
        DJNZ   R2 , LOOP
```

2．定义字节伪指令

格式：

标号：DB　X1, X2, …, Xn

此伪指令的功能是把 Xi 存入从标号开始的连续单元中。此伪指令常用来建立常数表，其中 Xi 为 8 位数据或 ASCII 码，表示 ASCII 码时应使用单引号‘’。当 Xi 为数值常数时，取值范围为 00～FFH；为字符串常数时，其长度不应超过 80 个字符。

例如：ORG　1000H

DB　20H , 21H , 22H

此时（1000H）=20H , （1001H）=21H , （1002H）=22H

例如：ORG　2000H

DB　‘03’

此时 （2000H）=30H

（2001H）=33H

DB‘03’伪指令中的‘03’为字符串常数，其中‘0’表示 ASCII 码的‘0’，‘3’表示 ASCII 码的‘3’。

3．定义双字节伪指令

格式：

标号：DW　X1, X2, …, Xn

此伪指令的功能是把 Xi 存入从标号开始的连续单元中。其中 Xi 为 16 位数值常数，16 位数据占两个存储单元，先存高 8 位，再存低 8 位。

例如：ORG　2100H

DW　1226H , 0562H

此时 （2100H）=12H , （2101H）=26H, （2102H）=05H, （2103H）=62H

4．预留存储区伪指令

格式：

DS　n

此伪指令的功能是从标号指定单元开始，预留 n 个单元的存储区。

例如：ORG　2200H

STOR: DS　06H

DB　21H , 22H

上述伪指令从 2200H 单元开始，连续预留 6 个单元，然后从 2206H 单元开始按 DB 命令给内存单元赋值，即（2206H）=21H，（2207H）=22H。

5．赋值伪指令

格式：

字符名称 x　EQU　n

此伪指令的功能是将数据或地址 n 赋给字符名称，使 x 与 n 等值。其中 n 可以是单字节数据，也可以是双字节数据，还可以是工作寄存器及直接地址。

例如： LG　EQU　10H　；LG 与 10H 等值

```
DE   EQU   R0     ；DE 与 R0 的内容等值
MOV   A , LG      ；A←10H
MOV   R1 , DE     ；R1←（R0）
```

使用赋值伪指令给程序的编制、调试、修改带来方便。如果在程序中要多次使用某一数据，可以使用 EQU 指令将该数据赋给一个字符名。一旦此数据发生变化，只要改变 EQU 指令中的数据即可。若不使用 EQU 指令，则要对所有涉及这一数据的指令进行修改。使用 EQU 指令，必须先赋值后使用。

6．结束汇编伪指令

格式：

```
END
```

此伪指令的功能是指示源程序到此结束，常将其放在汇编语言源程序的末尾。

4.1.4　常用的程序设计结构

1．顺序结构程序设计

顺序结构程序是一种最简单、最基本的程序。它按照程序编写的顺序依次执行。任何复杂的程序都含有较大成分的顺序结构程序。

【例 4-1】 将两位压缩 BCD 码转换成二进制数。

编程思路：（a1a0）BCD=a1×10+a0

编程说明：待转换的两位压缩 BCD 码存放于 R2 中，转换结果存回 R2。

程序流程图如图 4-1 所示。

编程如下。

```
        ORG 0000H
        AJMP START
        ORG 0100H
START:  MOV    A , R2
        ANL    A , #0F0H     ；取高位 BCD 码
        SWAP   A
        MOV    B , #0AH
        MVL    AB
        MOV    R3 , A
        MOV    A , R2
        ANL    A , #0FH      ；取低位 BCD 码
        ADD    A , R3
        MOV    R2 , A
        END
```

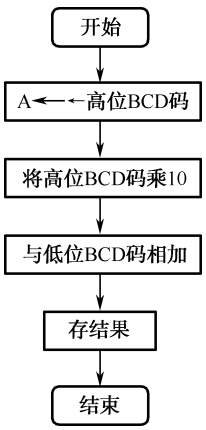


图 4-1　两位压缩 BCD 码转换成二进制数流程图

【例 4-2】请编写能把 20H 单元内两个 BCD 数变成相应的 ASCII 码并放置 21H（高位 BCD 数的 ASCII 码）和 22H（低位 BCD 数的 ASCII 码）单元的程序。

编程如下。

```
        ORG 0000H
```

```
        AJMP START
        ORG 0100H
START:MOV R0，  #22H        ；R0←22H
      MOV @R0，#00H         ；22H 单元清零
      MOV A, 20H            ；20H 中 BCD 数送 A
      XCHD  A,@R0           ；低位 BCD 数至 22H
      ORL   22H,#30H        ；完成低位 BCD 数转换
      SWAP  A               ；高位 BCD 数送低 4 位
      ORL   A,#30H          ；完成高位 BCD 数转换
      MOV   21H,A           ；存入 21H 单元
      SJMP  $               ；结束
      END
```

2．分支程序设计

在解决实际问题时，常常需要根据不同的条件去执行不同的处理程序，这样程序便产生了分支，这种结构的程序称为分支程序。分支程序的结构如图 4-2 所示。

把图 4-2 的分支结构编写成分支程序，可用条件转移指令、比较转移指令和位转移指令，这些指令均可实现程序的分支。下面通过两个例子来说明分支程序的编写。

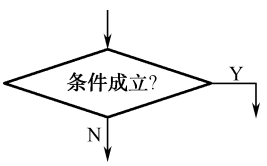


图 4-2　分支程序结构图

【例 4-3】　求符号函数的值。

$$Y=\begin{cases}1 & \text{当} X>0\\ 0 & \text{当} X=0\\ -1 & \text{当} X<0\end{cases}$$

编程说明：设变量 X 存放在 40H 单元中，函数 Y 存放在 41H 单元中。此程序为 3 个分支程序。

程序流程如图 4-3 所示。

编程如下。

```
        ORG 0000H
        AJMP START
        ORG 0100H
START: MOV   A，40H
       JZ    COMP
       JNB   ACC.7，  POST
       MOV   A，#81H              ；表示－1
       SJMP  COMP
       POSI: MOV   A，#01H        ；表示+1
COMP: MOV   41H，A
       END
```

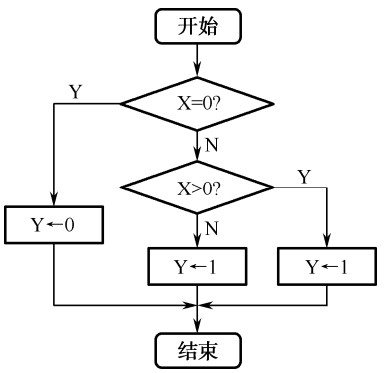


图 4-3　求符号函数值的流程图

【例 4-4】　求单字节有符号二进制数的补码。

编程思路：正数的补码是正数自身，负数的补码是负数的反码加 1。因此，编程时首先应判断待转换数的符号。数的最高位为符号位。最高位为 0 时，该数为正数；最高位为 1 时，该数为负数。

编程说明：设单字节二进制数存放在内存 30H 中，转换后的补码仍存回 30H。程序流程如图 4-4 所示。

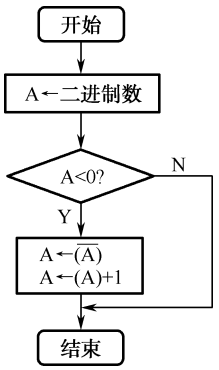


图 4-4　求单字节有符号二进制数的补码流程图

编程如下。

```
        ORG 0000H
        AJMP CMPT
        ORG 0100H
CMPT:   MOV   A , 30H
        JNB   ACC.7 , NCH      ;（A）≥0，不需要转换
        MOV   C , ACC.7        ; 保存符号
        MOV   10H , C
        CPL   A
        ADD   A , #1
        MOV   C , 10H
        MOV   ACC.7 , C        ; 恢复符号
NCH:    END
```

3．散转程序设计

散转程序属于分支程序的范畴，是一种并行多分支程序。它根据某种输入或运算结果，分别转向各个处理程序。散转程序常使用散转指令 JMP @A+DPTR 实现程序的跳转。其中，DPTR 常存放散转地址表的首地址，累加器 A 存放转移地址序号，这与分支程序不同。分支程序一般采用条件转移指令或比较转移指令实现程序的跳转。散转程序的基本结构如图 4-5 所示。

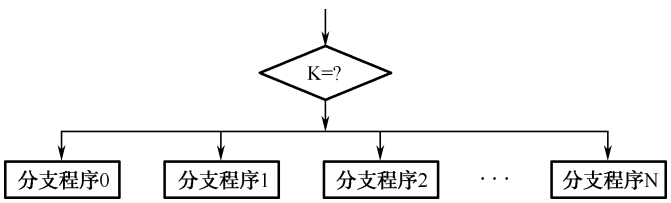


图 4-5　散转程序基本结构图

【例 4-5】 用单片机做四则运算。

编程说明：在单片机系统中，设置+、−、×、÷ 4 个运算键，这 4 个运算键的键值分别为 0、1、2、3（键值存放在寄存器 R2 中），当其中一个按键按下时，进行相应的运算。P1 口输入被加数、被减数、被乘数或被除数，以及运算结果的低 8 位或商。P3 口输入加数、减数、乘数或除数，以及结果的高 8 位或余数。键号存放在累加器 A 中。

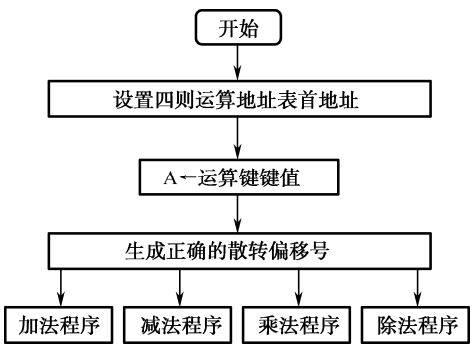


图 4-6　四则运算流程图

程序流程如图 4-6 所示。

参考程序如下。

```
        ORG 0000H
        AJMP START
        ORG 0100H
START:  MOV   P1 , #0FFH
        MOV   P3 , #0FFH
        MOV   DPTR, #TABLE
        CLR   C
```

```
        MOV   A, R2
        SUBB   A,#04H
        JNC   ERROR
        ADD   A , #0-4H
        CLR   C
        RL    A                    ；正确的散转偏移号，键号×2
        JMP   @A+DPTR
TABLE: AJMP   PRG0
        AJMP   PRG1
        AJMP   PRG2
        AJMP   PRG3
        END

ERROR:  （错误处理）
        RET
PRG0:  MOV   A，P1
        ADD   A，P3
        MOV   P1，A
        CLR   A
        ADDC   A，#00H              ；进位位放入 A 中
        MOV   P3，A
        RET
 PRG1: MOV   A，P1
        CLR   C
        SUBB   A，P3
        MOV   P1，A
        CLR   A
        RLC   A                     ；借位放入 A 中
        MOV   P3，A
        RET
PRG2:  MOV   A，P1
       MOV   B，P3
       MUL   AB
       MOV   P1，A
       MOV   P3，B
       RET
PRG3:  MOV   A，P1
       MOV   B，P3
       DIV   AB
       MOV   P1，A
       MOV   P3，B
       RET
```

几点说明：

（1）AJMP add11 为双字节指令，因此键号先乘以 2，生成正确的散转偏移号，存入累加器 A 中，再与 DDTR 中的内容相加，生成正确的目标地址。

（2）A 中的内容乘以 2 后不能大于 255，所以，本例中最多可扩至 128 个分支程序。

（3）因为散转程序采用 AJMP add11 指令，所以分支程序必须与此指令同在 2KB 地址空间内。

4．循环程序设计

在实际编程时，常常需要多次使用一段完全相同的程序。此时，可以采用循环程序，以缩短程序，减少程序占用的内存空间。循环程序一般包括下面几个部分。

1）循环初值

循环初值是指执行循环时，各工作单元的初始值以及循环的次数。

2）循环体

需要多次重复执行的程序段称为循环体。它是循环程序的主体。

3）循环修改

执行循环程序时，每执行一次，都要对参与工作的各工作单元的地址进行修改，以指向下一个待处理的单元，这叫循环修改。

4）循环控制

循环控制部分，首先对循环次数进行修改，然后对循环结束条件进行判断。如果不满足循环结束条件，继续执行循环；如果满足循环结束条件，则退出循环，继续执行后续程序。MCS-51 单片机对循环次数的修改与循环控制部分使用指令 DJNZ　Rn，rel 或 DJNZ direct，rel。

循环程序的一般结构如图 4-7 所示。

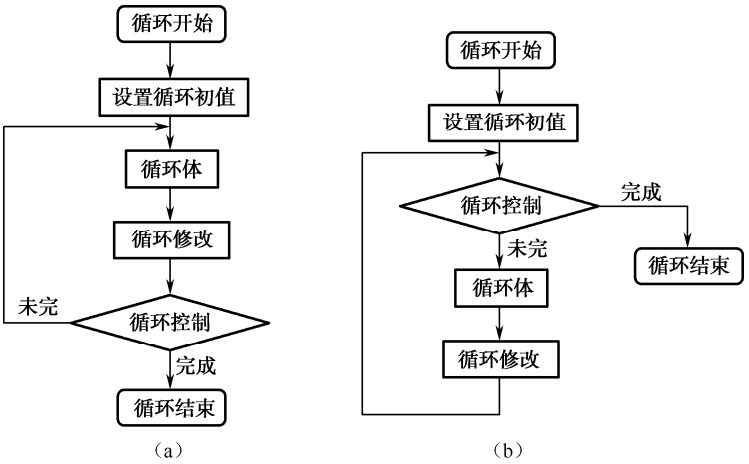


图 4-7　循环程序的一般结构图

循环程序存在两种结构：一种是图 4-7（a）所示的结构，先执行循环体和循环修改，再进行循环控制判断；另一种是图 4-7（b）所示的结构，先进行循环控制判断，再执行循环体和循环修改。

下面举例说明循环程序的应用。

【例 4-6】 已知内存单元有 16 个二进制无符号数，分别存放在 30H～3FH 单元中，试求它们的累加和，并将其存放在 R4、R5 中。

编程说明：存放 16 个二进制无符号数的首地址为 30H，此循环程序的循环次数为 16 次，和数放在 R4、R5 中。程序流程如图 4-8 所示。

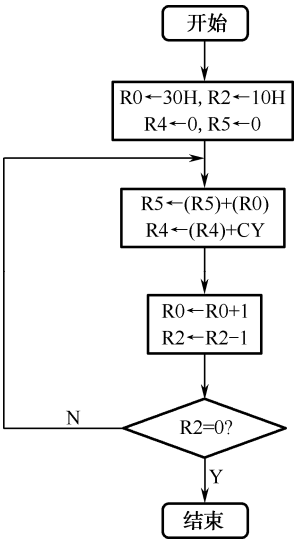


图 4-8　16 个无符号二进制数累加程序流程图

参考程序如下。

```
        ORG 0000H
        AJMP START
        ORG 0100H
START:  MOV   R0 ,#30H
        MOV   R2 , #10H
        MOV   R4 , #00H
        MOV   R5 , #00H
LOOP:   MOV    A , R5
        ADD   A , @R0
        MOV   R5 , A
        MOV   A , #00H
        ADDC   A , R4
        MOV   R4 , A
        INC   R0
        DJNZ   R2 , LOOP
        END
```

【例 4-7】 将内部数据存储器 30H～4FH 单元中的内容传送至外部数据存储器 2000H 开始的单元中。

编程说明：

内部数据区首址：　R0←30H

外部数据区首址：　DPTR←2000H

循环次数：　　R2←32H

程序流程如图 4-9 所示。

参考程序如下。

```
        ORG 0000H
        AJMP START
        ORG 0100H
START:    MOV   R0 , #30H
          MOV   DPTR , #2000H
          MOV   R2 , #32H
LOOP:   MOV   A , @R0
          MOVX   @DPTR , A
          INC   R0
          INC   DPTR
          DJNZ   R2 , LOOP
          END
```

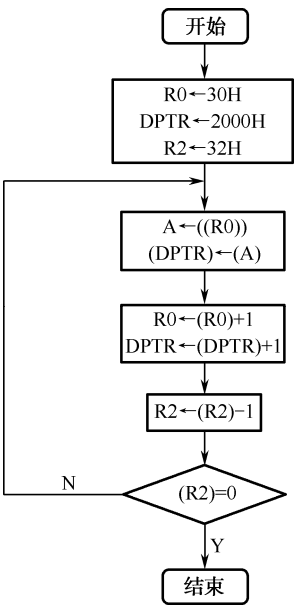


图 4-9　数据块传送程序流程图

图 4-7 所示的循环程序结构是单重循环结构。当循环程序中还包含循环程序时，称为循环嵌套。具有循环嵌套的程序称为多重循环程序。下面以排序程序为例，介绍双重循环。

【例 4-8】 将内部数据存储器中连续存放的 N 个数据由小到大进行排列。

设数据区首地址存于寄存器 R0 中，数据个数 N 存于寄存器 R6 中。程序流程如图 4-10 所

示。

参考程序如下。

```
        ORG 0000H
        AJMP START
        ORG 0100H
START: MOV   30H , R0
L1:   DEC   R6
      MOV     A , R6
      MOV   R3 , A
      MOV   R2 , A
      MOV   A , R0
      MOV   R1 , A
      INC   R1
L2:   MOV     A , @R0
      CLR   C
      SUBB     A , @R1
      JC   L3
      MOV     A , @R0
      XCH     A , @R1
      MOV   @R0 , A
L3:   INC   R1
      DJNZ   R2 , L2
      INC   R0
      DJNZ   R3 , L1
      MOV   R0 , 30H
      END
```

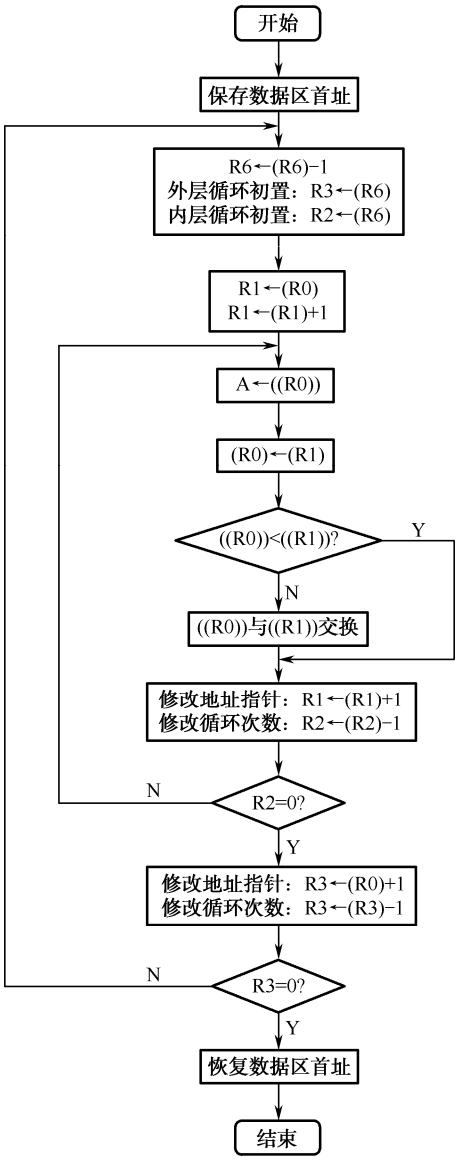


图 4-10　排序程序流程图

5．查表程序设计

在 MCS-51 指令系统中，有两条查表指令：

```
MOVC   A , @A+DPTR
MOVC   A , @A+PC
```

它们为查表程序设计奠定了基础。复杂的运算或转换过程，可用查表的方法设计程序，这往往可以简化程序，缩短程序的长度和程序的执行时间。查表程序广泛应用于数码显示、打印字符的转换，以及数据的补偿等复杂的运算程序中。

用查表的方法设计程序，首先应该在程序存储器中建立相应的表。例如，为 Y=X*X（设 X=0～9）函数建立一个表：先计算出 X=0,1,2,…,9 时所对应的 Y 值；将 Y 值按顺序存放在始地址为 TABLE 的程序存储器中。根据 TABLE+X 的值，就可以找到与 X 相对应的 Y 值。若将表的基地址 TABLE→DPTR，将 X 的值→A，则使用远查表指令 MOVC　A, @A+DPTR 后，累加器 A 中的值就是待求的 Y 值；也可以用近查表指令 MOVC　A , @A+PC 来实现。

【例 4-9】 利用查表的方法编写 $Y= X^2$（X=0,1,2,…,9）的程序。

编程说明：设变量 X 的值存放在内存 30H 单元中，变量 Y 的值存入内存 31H 单元中。先用远查表指令 MOVC A, @A+DPTR 编写程序（见参考程序 1）；再用近查表指令 MOVC　A , @A+PC 编写程序（见参考程序 2）。

参考程序 1：

```
        ORG 0000H
        AJMP START
        ORG   1000H
START:  MOV   A , 30H
        MOV   DPTR , #TABLE
        MOVC  A , @A+DPTR
        MOV   31H , A
TABLE:  DB    0 , 1 , 4 , 9 , 16
        DB    25 , 36 , 49 , 64 , 81
        END
```

参考程序 2：

```
        ORG 0000H
        AJMP START
        ORG   1000H
START:  MOV   A , 30H
        ADD   A , #02H
        MOVC  A , @A+PC
        MOV   31H , A
        DB    0 , 1 , 4 , 9 , 16
        DB    25 , 36 , 49 , 64 , 81
        END
```

参考程序 2 中，执行完第一条指令 MOV　A, 30H 后，A 中的内容为 X 的值。第二条指令 ADD　A , #02H 的作用是为了正确定位表的位置，以使第三条指令能正确地取出与 X 对应的 Y 值。表的首址与 MOVC　A, @A+PC 指令之间的指令 MOV 31H , A 占了两字节存储空间，因此，第二条指令对累加器 A 加 2。若查表指令与表之间的指令占用了 n 个存储单元，则在查表指令前，累加器 A 的内容要先加 n。

【例 4-10】 将一位十六进制数转换成相应 ASCII 码。用计算求解和查表求解进行比较。

（1）计算求解。

编程说明：设待转换的一位十六进制数存放在 40H 单元中，转换后的 ASCII 码仍存放在 40H 中。

编程思路：十六进制数 0～9 的 ASCII 码为 41H～46H，当待转换的数小于等于 9 时，加 30H，即其对应的 ASCII 码；当待转换的数大于 9 时，加 37H。程序流程如图 4-11 所示。

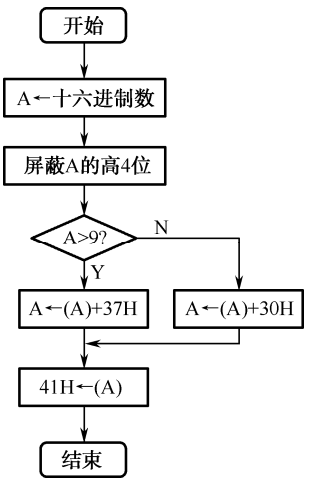


图 4-11　一位十六进制数转换成 ASCII 码程序流程图

参考程序如下。

```
ORG 0000H
AJMP START
  ORG    0100 H
  MOV    A , 40 H
  ANL    A    , # 0F H
  CLR    C
  SUBB   A, # 0AH
  JC     NEXT
```

```
        ADD   A , #0AH
        ADD   A , #37H
        SJMP   SAVE
NEXT:   ADD   A , #0AH
        ADD   A , #30H
SAVE:   MOV   40H , A
        END
```

（2）查表求解。

```
        ORG 0000H
        AJMP START
        ORG 0100H
          ORG   0100H
START:  MOV   A , 40H
        ANL   A , #0FH
        ADD   A , 02H
        MOVC   A , @A+PC
        MOV   40H , A
        DB   ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’
        DB   ‘6’, ‘7’, ‘8’, ‘9’, ‘A’, ‘B’
        DB   ‘C’, ‘D’, ‘E’, ‘F’
        END
```

通过两种求解方法不难看出，此例使用查表方法可将分支结构程序变成顺序结构程序，更为简单方便。

6．子程序设计

经常会遇到这样的情况，在程序的不同位置需要使用一段完全相同的程序。为了避免多次出现相同的程序段，可将相同的程序写成独立的程序段。在任何需要的地方调用该程序，运行完毕后，再从该程序返回到原来的程序继续运行。这样独立的程序段称为子程序。调用子程序的程序称为主程序。

使用子程序会大大简化主程序的结构，增加程序的可读性，避免重复性的工作，缩短整个程序，节省程序存储器空间。同时，子程序还增加了程序的可移植性。一些常用的程序，如代码转换程序、运算程序、外部设备的输入/输出驱动程序等，写成公用子程序可以被随时引用，为用户提供了方便。

1）子程序的调用与返回

MCS-51 单片机有 ACALL addr11 和 LCALL addr16 两条子程序调用指令；一条 RET 子程序返回指令。

（1）子程序调用。主程序在需要调用子程序时，使用 ACALL addr11 或 LCALL addr16 来完成。调用指令中的地址为子程序的入口地址，在汇编语言中通常用标号来表示。执行子程序调用指令时，单片机首先将当前的 PC 值（即调用指令的下条指令的首地址）压入堆栈保存，将子程序的入口地址送入 PC 中，然后转去执行子程序，完成主程序对子程序的调用。

（2）子程序返回。被调用的子程序执行完后，需要返回主程序。这时，子程序中的一条指令 RET（一般在子程序最后）将调用子程序时压栈的 PC 值返弹给 PC，使程序返回调用的地方，

继续向下执行主程序。

2）保存与恢复寄存器内容

主程序和子程序常常要使用系统相同的资源，如寄存器。如不采取相应措施，可能会发生冲突，使系统产生错误。例如，子程序改变了主程序正在使用的寄存器的内容。为此，可在进入子程序的开始部分，把寄存器的内容保存在堆栈中，这叫保护现场；在退出子程序前，把寄存器内容恢复原状，这叫恢复现场。

例如：
```
SUB1: PUSH   PSW
      PUSH   A
      PUSH   R6
             ⋮          （此处省略了子程序的内容）
      POP    R6
      POP    A
      PUSH   PSW
```

堆栈操作遵循后进先出的原则。

保护现场和恢复现场的工作也可在主程序中完成。

3）子程序的参数传递

主程序在调用子程序时，经常需要传送一些参数，子程序运行完后也经常将一些参数回送给主程序，这叫参数传递。

（1）入口参数。在调用子程序时，主程序应先把有关参数放在某些约定的寄存器或存储单元中。子程序运行时，可以从这些约定位置得到这些参数，这叫入口参数。

（2）出口参数。子程序结束前，也应把运算结果送到约定的寄存器或数据存储单元中，返回主程序后，主程序从约定位置获得这些参数，这叫出口参数。

在编写子程序时需首先确定入口参数和出口参数的存放位置。对存放入口参数和出口参数的寄存器，不能进行现场保护，否则就破坏了应该向主程序传送的参数。参数传递可以采用累加器 A、工作寄存器和堆栈等来完成。

【例 4-11】 编程计算 $c= a^2+b^2$。

编程说明：计算某数的平方可以用子程序来实现，两次调用该子程序并求和便得到所需结果。设 a、b 分别存于内部 RAM 的 30H、31H 单元，结果 c 存于内部 RAM 的 40H 单元。

参数传递：主程序中，将某数存放到累加器 A 中，作为子程序的入口参数；子程序中，将所求数的平方值存放在累加器 A 中，作为出口参数（即主程序的返回值）。

子程序的入口参数：A 中存放某数的值。

子程序的出口参数：A 中存放所求数的平方。

子程序如下。

```
SQR:    INC   A
        MOVC  A , @A+PC    ；查平方表
        RET
TABLE:   DB     0 , 1 , 4 , 9 , 16
         DB     25 , 36 , 49 , 64 , 81
```

主程序流程如图 4-12 所示。

主程序如下。

```
        ORG 0000H
        AJMP START
        ORG 0100H
START:  MOV   A , 30H
        ACALL   SQR          ；调查表子程序
        MOV   R1 , A         ；a² 暂存 R1 中
        MOV   A , 31H
        ACALL   SQR          ；调查表子程序
        ADD   A , R1
        MOV   40H , A
        END
```

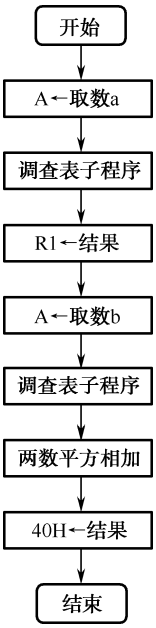


图 4-12　程序流程图

4）子程序的嵌套

在子程序中又调用其他子程序，称为子程序嵌套。MCS-51 单片机允许多重嵌套。下面列出三重嵌套子程序的结构，如图 4-13 所示。

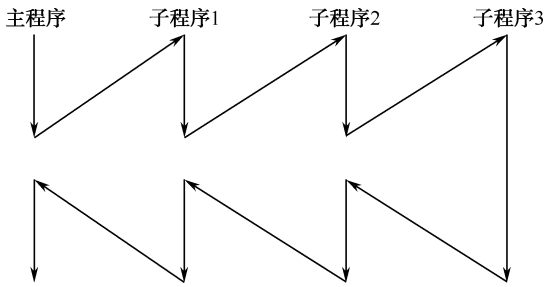


图 4-13　子程序的嵌套

4.2　汇编语言源程序的汇编

将汇编语言源程序转换为单片机能执行的机器码形式的目标程序的过程叫汇编。汇编常用的方法有两种，一是手工汇编，二是机器汇编。

手工汇编时，把程序用助记符指令写出后，通过手工方式查指令编码表，逐个把助记符指令翻译成机器码，然后把得到的机器码程序（以十六进制形式）输入到单片机开发机中，并进行调试。由于手工汇编是按绝对地址进行定位的，所以，对于偏移量的计算和程序的修改非常不便。通常只在程序较小或开发条件有限制时才使用。

机器汇编是在常用的个人计算机 PC 上，使用交叉汇编程序将汇编语言源程序转换为机器码形式的目标程序。此时汇编工作由计算机完成，生成的目标程序由 PC 传送到开发机上，经调试无误后，再固化到单片机的程序存储器 ROM 中。机器汇编与手工汇编相比具有极大的优势，所以是汇编工作的首选。

源程序经过机器汇编后，形成的若干文件中含有两个主要文件，一个是列表文件，另一个是目标码文件。因汇编软件不同，文件的格式及信息会有一些不同，但主要信息如下。

列表文件主要信息为：

地 址	目标码	汇编程序
		ORG 0040H
0040H	747F	MOV A，#7FH
0042H	7944	MOV R1，#44H
		END

目标码文件主要信息为：

首地址	末地址	目标码
0040 H	0044H	747F7944

该目标码文件由 PC 的串行口传送到开发机后，接下来的任务就是在专门的编译系统软件中对所编写的程序进行仿真调试了。

习　　题

1．解释下列名词：指令，指令系统，伪指令。

2．MCS-51 单片机有哪几种寻址方式？共有多少条指令？

3．指出下列指令中源操作数和目标操作数的寻址方式。

```
MOV     A , 40H
MOV     A , @R0
MOV     R1 , #50H
MOV     DPTR , #1000H
MOVX    @DPTR , A
MOVC    A , @A   + PC
ADD     A , R0
MOV     C , 20H
```

4．已知内部数据存储器 40H 单元中的内容为 67H，41H 单元中的内容为 68H，试分析下段程序执行后，各有关单元中的内容。

```
MOV   A , 40H
MOV   R1 , A
MOV   R0 , #41H
MOV   A , @R0
MOV   40H , A
MOV   R1 , A
MOV   R0 , #66H
```

5．试编写一段程序，将内部数据存储器 40H、41H 单元中的内容，传送到外部数据存储器 2000H、2001H 单元中。

6．已知 A=B9H，B=6AH，Cy=0，试分析执行下列指令后，标志位的内容。

```
ADD    A , B
SUBB   A , B
```

7．试编程将累加器 A 的低 4 位由 P1 口的低 4 位输出，P1 口的高 4 位保持不变。

8．试编程将 P1 口的高 3 位置位，低 5 位不变。

9．试比较指令 SJMP rel、AJMP addrll 和 LJMP addr16 的不同之处。

10．N=128 的分支程序。已知 R3 的值为 00H~7FH 中的一个，请编出根据 R3 中值转移到相应分支程序的程序。

11．已知一个补码形式的 16 位二进制数（低 8 位在 NUM 单元，高 8 位在 NUM+1 单元），试编写能求该 16 位二进制数原码的绝对值的程序。

12．已知两个带符号数分别存于 ONE 和 TWO 单元，试编程比较它们的大小，并把大数存入 MAX 单元。

13．已知内部 RAM ADDR 为起始地址的数据块内数据是无符号数，块长在 LEN 单元内。请编程求数据块中最大值并存入 MAX 单元。

14. 假设外部 RAM 中有 3 个 8 位无符号数据，它们的地址分别为 7111H、5687H 和 8766H，找出它们之中最大的数放入寄存器 A 中。

15．寄存器 A 中存放了一个成绩值（0≤A≤100），编写程序，判定该成绩所属的级别，放入寄存器 B 中。判定标准为：1 级（90≤A≤100）、2 级（80≤A≤89）、3 级（70≤A≤79）、4 级（60≤A≤69）、5 级（A≤59）。