模块四、单片机的程序设计及其应用

本章主要介绍单片机的程序设计及相关的项目应用，汇编语言的程序设计部分主要介绍了伪指令的概念、程序设计

【教学聚焦】

知识目标：
1、熟悉基础的程序设计方法
2、掌握程序编制的方法和技巧
3、掌握应用程序设计方法
技能目标：
1、能够熟练的编写调试实验程序
2、能够利用单片机仿真器开发调试单片机应用程序的过程
3、能够掌握单片机编程器的使用方法
4、能独立进行一般性的程序设计

【课时建议】6 课时

教学重点：掌握基础的程序设计方法
教学难点：能自己编写调试实验程序

【课堂随笔】

结合实训介绍电动机控制技术

项目 4．1 汇编语言源程序的编辑和汇编

4.1.1 伪指令

汇编程序中提供了一套伪指令，以支持汇编的运行。这些伪指令仅在汇编过程中起控制作用，不产生可执行目标代码，与机器指令代码无一一对应关系，只能被汇编程序识别。汇编后，目标程序中不再出现伪指令，故又称为软指令。

这里介绍一些常用的伪指令。

1、起点命令 ORG

格式：

ORG　××××H

给程序起始地址或数据块的起始地址赋值命令。总是出现在每段源程序或数据块的开始，它指明此语句后面的程序或数据块的起始地址为××××H。在一个源程序中可多次使用 ORG 命令，以规定不同程序段或数据块的起始位置，所规定的地址从小到大，不允许重叠。

例如：

```
        ORG     8000H
START:  MOV     A,#74H
          .
          .
          .
```

表示源程序的人口地址为8000H，即程序从8000H开始执行。

2、结束命令END

格式：

END

汇编程序结束标志，该命令附在源程序的结尾。在END之后所写的指令，汇编时不予处理，因此一个源程序只能有一个END命令。

3、定义字节命令DB

格式：

标号：　DB　字节常数或字符

从指定单元开始，定义了若干个8位存储单元，以存放指令给出的数据或字符，字符若用引号括起来，则表示ASCII码。

例如：

```
        ORG     8000H
TAB:    DB      45H,73,'5','A'
TAB1:   DB      101B
```

这里数据块的首址由ORG命令定义，即TAB＝8000H，则有：

(8000H)＝45H
(8001H)＝49H
(8002H)＝35H
(8003H)＝41H
(8004H)＝05H

由DB命令定义的标号可以任选，DB所确定的单元地址有两种方法。

（1）若DB命令是在其他源程序之后，则源程序的最后一条指令地址之后，就是DB定义的数据或数据表格。

（2）由ORG定义数据块首址。

4、定义字命令DW

格式：

标号：　DW　字或字表

从指定单元开始，定义若干个字（双字节数）。

例如：

```
        ORG     8000H
HETAB:  DW      7234H，8AH，10
```

汇编后则：

(8000H)=72H
(8001H)=34H
(8002H)=00H
(8003H)=8AH
(8004H)=00H

(8005H)=0AH

5、定义空间命令 DS

格式：

标号：　　DS　数据或字符表达式

从指定单元开始，由数据或表达式确定保留若干个字节内存空间备用

例如：

```
ORG         8000H
DS          08H
DB          30H,8AH
```

即 8000H～8007H 单元保留备用

(8008H)＝30H

(8009H)＝8AH

以上 DB、DW、DS 伪指令只对程序存储器起作用。

6、等值命令 EQU

格式：

字符名称　　EQU　数据或汇编符号

此命令把一个数据或特定的汇编符号赋予标号段规定的字符名称。为“取代”之意，即以数据或汇编符号取代字符名称。用 EQU 定义的字符必须先定义后使用，这些被定义的字符名称可用作数据地址，位地址或立即数。

例如：

```
        ORG   8000H
AA      EQU   R6          ;AA 与 R6 等值
        MOV   A,AA    ;A(R6)
        …
```

7、数据地址赋值命令 DATA

格式：

字符名称　　DATA　数据或表达式

此命令把数据地址或代码地址赋予标号段规定的字符名称。

例如：

```
INDEXJ  DATA  8389H
```

定义了 INDEXJ 这个字符名称的地址为 8389H，主要用于程序的模块式调试。

例如：

```
                ORG         8000H
INDEXG          DATA        8096H
                LGMP        INDEXG
                END
```

等价于

```
ORG         8000H
LJMP         8096H
END
```

被定义的字符名称也可先使用后定义。

DATA 和 EQU 的区别在于用 DATA 定义的字符名称作为标号登记在符号表中，故可先使用后定义；而用 EQU 定义的字符名称必须先定义后使用，其原因是 EQU 不定义在符号表中。

8、位地址符号命令 BIT

格式：

字符名称　　BIT　位地址

该命令把位地址赋予标号段的字符名称。

例如：

A1　BIT　P1．0

A2　BIT　P1．1

这里位地址 P1．0、P1．1 分别赋给标号段的字符 A1 当作位地址用。

4.1.2 程序设计

1、概述

所谓程序设计，就是人们把要解决的问题用计算机能接受的语言，按一定的步骤描述出来。程序设计时要考虑两个方面：一是针对某种语言进行程序设计；二是解决问题的方法和步骤。对同一个问题，可以选择高级语言(如 PASCAL、C 等)来进行设计，也可以选择汇编语言来进行设计，并且往往有多种不同的解决方法。通常把解决问题而采用的方法和步骤称为“算法”。

2、采用汇编语言的优点

汇编语言与高级语言相比具有以下优点：

（1）占用的内存单元和 CPU 资源少；

（2）程序简短，执行速度快；

（3）可直接调动计算机的全部资源，并可有效地利用计算机的专有特性；

（4）能准确地掌握指令的执行时间，适用于实时控制系统。

3、汇编语言程序设计步骤

用汇编语言编写程序，一般可按如下步骤进行：

（1）建立数学模型

根据要解决的实际问题，反复研究分析并抽象出数学模型。

（2）确定算法

解决一个实际问题，往往有多种方法，要从诸多算法中确定一种较为简洁的方法是至关重要的。

（3）制订程序流程图

算法是程序设计的依据，把解决问题的思路和算法的步骤画成程序流程图。

（4）确定数据结构

合理地选择和分配内存工作单元以及工作寄存器。

（5）写出源程序

根据程序流程图，精心选择合适的指令和寻址方式来编制源程序。

（6）上机调试程序

将编制好的源程序进行汇编，成为可执行目标代码后，便可执行目标程序，检查修改程序中的错误，对程序运行结果进行分析，直至正确为止。

4、评价程序质量的标准

解决某一问题、实现某一功能的程序不是唯一的。程序有简有繁，占用的内存单元有多有少，执行时间有长有短，因而编制的程序也不同，怎样来评价程序的质量呢，通常有以下几个标准：

（1）程序的执行时间；
（2）程序所占用的内存字节数目；
（3）程序的逻辑性、可读性；
（4）程序的兼容性、可扩展性；
（5）程序的可靠性。

一般来说，一个程序执行时间越短，占用的内存单元越少，其质量越高。这就是程序设计中的“时间”和“空间”的概念。程序设计的逻辑性强、层次清楚、数据结构合理、便于阅读也是衡量程序优劣的重要标准；同时还要保护程序在任何实际工作条件下，都能正常运行。在较复杂的程序设计中，必须充分考虑程序的可读性和可靠性。另外程序的可扩展性、兼容性以及容错性等都是衡量与评价程序优劣的重要标准。

4.1.3 程序结构

1、简单程序

程序的简单与复杂很难有一个绝对标准，这里所说的简单程序是一种顺序执行的程序，它既五分支又无循环。这种程序虽然简单，但能完成一定的功能，是构成复杂程序的基础。

例 4.1.1 假设两个双字节无符号数，分别存放在 R1RO 和 R3R2 中，高字节在前，低字节在后。编程使两数相加，和数存放回 R2R1R0 中。

此为简单程序，求和的方法与笔算类同，先加低位，后加高位，无须画流程图。直接编程如下。

```
ORG       1000H
CLR       C
MOV       A,R0     ；取被加数低字节至 A
ADD       A,R2     ；与加数低字节相加
MOV       R0，A    ；存和数低字节
MOV       A,R1     ；取被加数高字节至 A
ADDC      A,R3     ；与加数高字节相加
MOV    R1，A   ；存和数高字节
MOV    A,#0
ADDC   A,#0     ；加进位位
MOV    R2,A     ；存和数进位位
*SJMP  $        ；原地踏步
END
```

由于 MCS—5l 指令系统无暂停指令，故用“SJMP $”指令（$表示“rel=OFEH”）实现原地踏步以代替暂停指令，后面将不再重复解释。

2、分支程序

在一个实际的应用程序中，程序不可能始终是直线执行的。要用计算机解决一些实际问题，要求计算机能够作出某种判断并根据判断作出不同的处理。通常会根据实际问题中给定的条件，判断条件满足与否，产生一个或多个分支，以决定程序的流向。因此条件转移指令形成的分支结构程序能够充分地体现计算机的智能。

（1）简单分支程序

例 4.1.2 设内部 RAM 30H，31H 单元中存放两个无符号数，试比较它们的大小。将较小的数存放在 30H 单元，较大的数存放在 31H 单元中。

这是一个简单分支程序，可以使两数相减，若 CY＝1，则被减数小于减数。即用 JC 指令进行判断。程序流程图如图 4.1 所示。

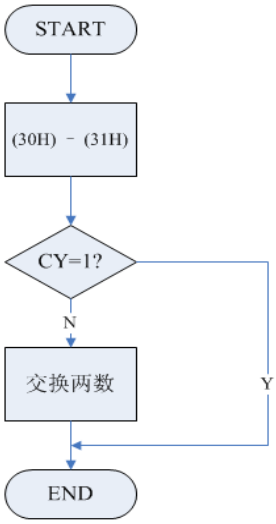


图 4.1 例 4.1.2 流程图

```
        ORG     1000H
START:  CLR     C           ；0—CH
        MOV     A,30H
        SUBB    A,31H       ；做减法比较两数
        JC      NEXT        ；若（30H）小，则转移
        MOV     A,30H
        XCH     A,31H
        MOV     30H,A       ；交换两数
NEXT:   NOP
        SJMP    $
        END
```

（2）多重分支程序

仅凭判断一个条件产生的分支无法解决的问题，需要判断两个或两个以上的条件，通常也称为复合条件，进行多方面测试产生的分支程序称为多重分支程序。

例 4.1.3 设 30H 单元存放的是一元二次方程 $ax^2+bx+c=0$ 根的判别式 $\Delta=b^2-4ac$ 的值。在实数范围内，若 $\Delta>0$，则方程有两个不同的实根；若 $\Delta=0$，则方程有两个相同的实根；若 $\Delta<0$，则方程无实根。试根据 30H 中的值，编写程序判断方程根的三种情况，在 31H 中存放“0”代表无实根；存放“1”代表有相同的实根；存放“2”代表两个不同的实根。

Δ 值为有符号数，有三种情况，即大于零、等于零、小于零。可以用两个条件转移指令来判断，首先判断其符号位，用指令 JNB　ACC.7，rel 判断，若 ACC.7＝1，则一定为负数；若 ACC.7＝0，则 $\Delta\geq 0$。此时，再用指令 JNZ　rel 判断，若 $\Delta\neq 0$，则一定是 $\Delta>0$；否则 $\Delta=0$。

程序流程图如图 4.2 所示。

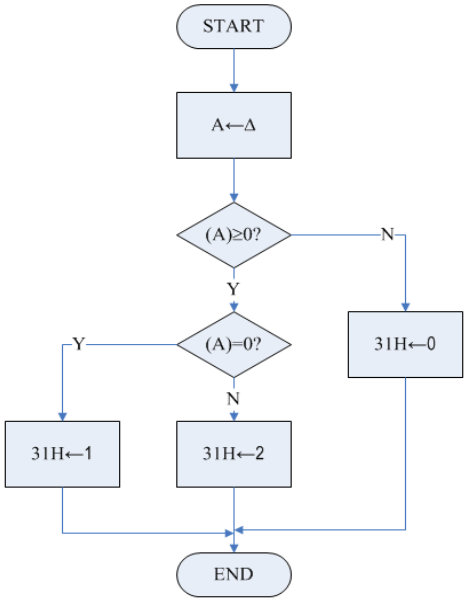


图 4.2 例 4.1.3 流程图

```
        ORG     1000H
START   MOV     A,30H           ；Δ值送 A
        JNB     ACC.7，YES      ；Δ≥0 转 YES
        MOV     31H,#0          ；Δ<0 无实根
        SJMP    FILISH
YES     JNZ     TOW             ；Δ>0 转 TOW
        MOV     31H,#2              ；Δ=0 有相同实根
        SJMP    FILISH
TOW     MOV     31H,#2          ；有两个不同实根
FILISH  SJMP    $
        END
```

（3）N 路分支程序

N 路分支程序是根据前面程序运行的结果，可以有 N 种选择，并能转向其中任一处处理程序。

例 4.1.4 N 路分支程序，设 N≤8。根据程序运行中产生的 R3 值，来决定如何进行分支。

分析：若逐次按图 4.3 流程图进行处理亦可使程序进入 8 个处理程序之一的入口地址。但这种方法判断次数多，当 N 较大时，运行速度慢。然而对 MCS-5l 来说，由于有间接转移（也称为散转）指令 JMP@A+DPTR 可通过一次转移，即可方便地进入相应的分支处理程序，效率大大提高。实现 N 路分支程序的方法如下:

①在程序存储器中，设置各分支程序人口地址表。

②利用 MOVC A，@A+DPTR 指令，根据条件查地址表，找到分支人口地址。方法是使 DPTR 指向地址表首址，再按运行中累加器 A 的偏移量找到相应分支程序人口地址，并将该地址存于 A 中。

③利用散转指令 JMP @A+DPTR 转向分支处理程序。

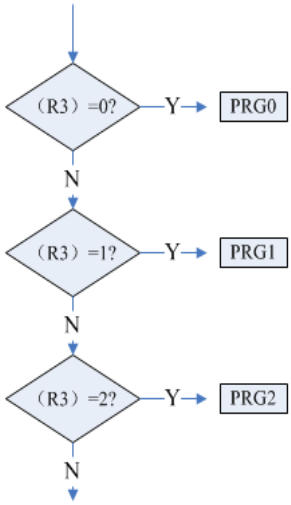


图 4.3 例 4.1.4 流程图

解　按以上分支，用几条指令便可实现多分支程序的转移。编程如下：

```
                MOV     A,R3
                MOV     DPTR,#PRGTBL   ；分置入口地址表首址送 DPTR
                MOVC        A,@A+DPTR      ；查表
                JMP     @A+DPTR            ；转移
PRGTBL          DB      PRGO—PRGTBL
                DB          PRG1—PRGTBL
                …
```

第三条指令是查表，查表结果如下。

$(A) = PRGi - PRGTBL$；　　即第 i 段分支程序的人口地址与散转表首址之差

执行第 4 条指令时，

$PC \leftarrow A + DPTR = PRGi - PRGTBL + PRGTBL = PRGi$

程序转入 PC 直接指向的第 i 个分支入口地址 PRGi。

设：N＝4，即有 4 个分支。

功能：根据人口条件转向 4 个程序段，每个程序段分别从内部 RAM 256B、外部 RAM 256B、外部 RAM 64KB、外部 RAM 4KB 数据缓冲区读取数据

入口条件：　(R3)＝(0，1，2，3)；

(R0)＝RAM 的低 8 位地址；

(R1)＝RAM 的高 8 位地址。

出口条件：累加器 A 中的内容为执行不同程序段后读取的数据

参考程序如下。

```
                MOV     A,R3
                MOV     DPTR,#PRGTBL
                MOVC    A,@A+DPTR
                JMP     @A+DPTR
PRGTBL          DB          PRG0-PRGTBL
                DB      PRG1-PRGTBL
                DB      PRG2-PRGTBL
```

```
        DB      PRG3-PRGTBL
PRG0    MOV     A,@R0           ；从内部 RAM 读数
        SJMP    PRGE
PRG1    MOV     P2，R1
        MOVX    A,@R0           ；从外部 RAM256B 读数
        SJMP    PRGE
PRG2    MOV     DPL,R0
        MOV     DPH,R1
        MOVX    A,@DPTR             ；从外部 RAM64KB 读数
        SJMP    PRGE
PRG3    MOV     A,R1
        ANL     A,#OFH              ；屏蔽高 4 位
        ANL     P2，#11110000B  ；P2 口高 4 位可作他用
        ORL     P2，A       ；只送 12 位地址
        MOVX    A,@R0       ；从外部 RAM4KB 读数
PRGE    SJMP    $
```

最后一个分支程序是从外部 RAM 的 4KB 存储区域读数，只需送出 12 位地址即可，不必占用 16 位地址线，P2 口的高 4 位可作它用。

使用这种方法，地址表长度加上分支处理程序的长度，必须小于 256 个字节，如果希望更多分支，则应采用其他方法。

3、 循环程序

（1）循环程序的导出

前面介绍的是简单程序和分支程序，程序中的指令一般执行一次。而在一些实际应用系统中，往往同一组操作要重复执行多次，这种有规可循又反复处理的问题，可采用循环结构的程序来解决。这样可使程序简短，占用内存少，重复次数越多，运行效率越高。

循环程序的基本结构如下：

①初始化部分

程序在进入循环部分之前，应对各循环变量，其他变量和常量赋初值。为循环作必要的准备工作。

②循环体部分

这一部分是由重复执行部分和循环控制部分组成。这是循环程序的主体，又称为循环体。值得注意的是每执行一次循环体后，必须为下一次循环创造条件。如对数据地址指针、循环计数器等循环变量的修改工作，还要检查判断循环条件，符合循环条件，则继续重复循环不符合时就退出循环，以实现对循环的判断与控制。

③结束部分

用来存放和分析循环程序的处理结果。

循环程序的关键是对各循环变量的修改和控制，尤其是循环次数的控制。一般在一些实际系统中有循环次数为已知的循环，可以用计数器控制循环；还有循环次数为未知的循环，可以按问题给定的条件控制循环。

例 4.1.5 从外部 RAM BLOCK 单元开始有一无符号数据块，数据块长度存人 LEN 单元，求出其中最大数存人 MAX 单元。

这是一基本搜索问题。采用两两比较法，取两者较大的数再与下一个数进行比较，若数据块长度 LEN＝n 则应比较 n-1 次，最后较大的数就是数据块中的最大数。

为了方便地进行比较，使用 CY 标志来判断两数的大小，使用 B 寄存器作比较与交换

的暂存器，使用 DPTR 作外部 RAM 地址指针。流程图如图 4.4 所示。

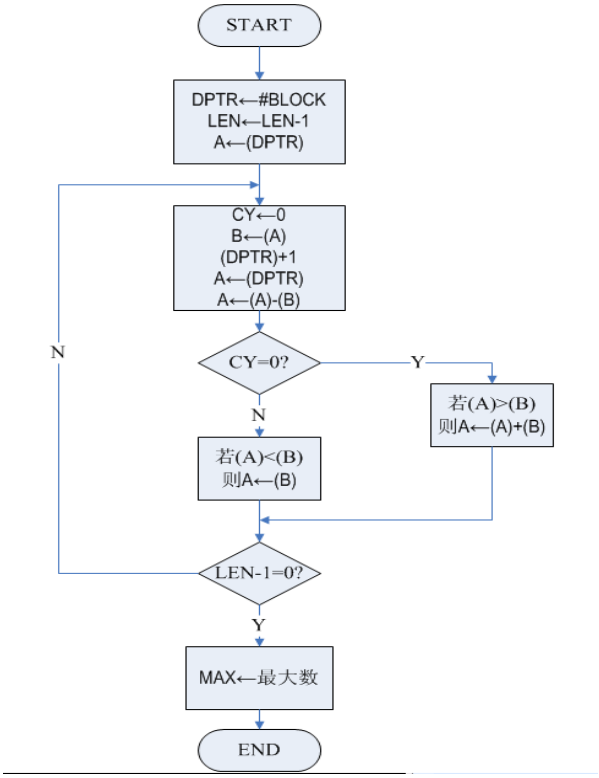


图 4.4 例 4.1.5 流程图

```
        ORG     0400H
BLOCK   DATA    0100H           ；定义数据块首址
MAX     DATA    31H             ；定义最大数暂存单元
LEN     DATA    30H             ；定义长度计数单元
FMAX：  MOV     DPTR，#BLOCK        ；数据块首址送 DPTR
        DEC     LEN             ；长度减 1
        MOVX    A，@DPTR            ；取数至 A
LOOP    CLR     C               ；0’CY
        MOV     B,A             ；暂存于 B
        INC     DPTR            ；修改指针
        MOVX    A,@DPTR             ；取数
        SUBB    A,B
        JNC     NEXT                ；大者送 A
        MOV     A,B             ；(A)>(B)，则恢复 A
        SJMP    NEXT1
NEXT：  ADD     A,B             ；(A)>(B)，则恢复 A
NEXT1： DJNZ    LEN,LOOP            ；未完继续比较
        MOV     MAX,A           ；存最大数
        SJMP    $               ；x 若用 RET 指令结尾则
        END                     ；该程序可作子程序调用
```

（2）多重循环

程序只有一个循环，这种程序被称为单循环程序。而遇到复杂问题时，采用单循环往往不够，还必须采用多重循环才能解决。所谓多重循环，就是在循环程序中还套有其他循环程

序，这就是多重循环结构的程序。利用机器指令周期进行延时是最典型的多重循环程序。

例 4.1.6 延时 20ms 子程序，设晶振主频为 12MHz。

在系统晶振主频确定之后，延时时间主要与两个因素有关。其一是循环体(内循环)中指令的执行时间的计算；其二是外循环变量(时间常数)的设置。

已知主频为 12MHz，一个机器周期为 1 μs ，执行一条 DJNZ Rn，rel 指令的时间为 2 μs 。延时 20ms 子程序如下:

```
DELY：    MOV     R7，#100
DLY0：    MOV     R6，#100
DLY1：    DJNZ    R6，DLY1    ；100×2=200μs
          DJNZ    R7，DLY0    ；100×200μs=20ms
          RET
```

以上延时时间不太精确，没有把执行外循环中其他指令计算进去。若把循环体以外的指令计算在内，则它的延时时间为：

$$(200\mu s+3\mu s)\times 100+3=20003\mu s=20.003ms$$

如果要求比较精确的延时，程序修改如下。

```
DELY：    MOV     R7，#100
DLY0：    MOV     R6，#98
DLY1：    DJNZ    R6，DLY1    ；98×2=196μs
          DJNZ    R7，DLY0
          RET
```

它的实际延时为：

$$(196+2+2)\times 100+3=20003\mu s=20.003ms$$

也有一定误差。如果需要延时更长时间，则可以采用更多的循环。

（3）编写循环程序应注意的问题

从上面介绍的几个例子，不难看出，循环程序的结构大体上是相同的。要特别注意以下几个问题：

①在进入循环之前，应合理设置循环初始变量。

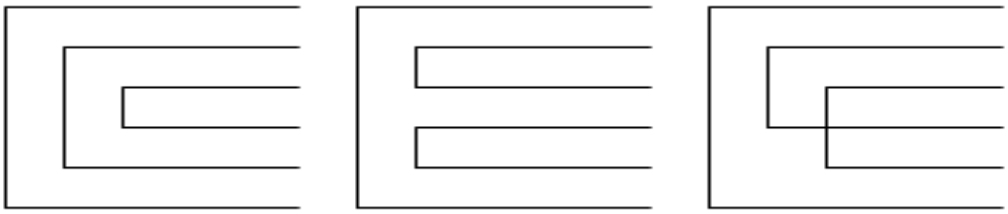
②循环体只能执行有限次，如果无限执行的话，称之为“死循环”，这是应当避免的。

③不能破坏或修改循环体，要特别注意是避免从循环体外直接跳转到循环体内。

④多重循环的嵌套，应当是以下两种形式：图 4.5（a）和图 4.5（b）均正确，应避免图 4.5（c）的情况。由此可见，多重循环是从外层向内层一层层进入，从内层向外层一层层退出。不要在外层循环中用跳转指令直接转到内层循环体内。

⑤循环体内可以直接转到循环体外或外层循环中，实现一个循环由多个条件控制结束的结构。

⑥对循环体的编程要仔细推敲，合理安排，对其进行优化时，应主要放在缩短执行时间上，其次是程序的长度。



(a)嵌套一　　（b）嵌套二　　（c）嵌套三

图 4.5 嵌套示意图

4、查表程序

查表是程序是程序设计中经常遇到的，对于一些复杂参数的计算，不仅程序长，难以计算，而且要耗费大量时间。尤其是一些非线性参数，用一般算术运算解决是十分困难的。它涉及对数、指数。三角函数，以及微分和积分运算。对于这样一些运算，用汇编语言编程都比较复杂，有些甚至无法建立数学模型，如果查表法解决就容易多了。

所谓查表，就是把事先计算或测得的数据按一定顺序编制成表格，存放在程序存储器中。查表程序的任务就是根据被测数据，查出最终所需要的结果。因此查表比直接计算简单得多，尤其是对非数值计算的处理上。利用查表法可完成数据运算、数据转换和数据补偿等工作。并具有编程简单，执行速度快，适合于实时控制等优点。

编程时可以方便利用伪指令 DB 或 DW 把表格的护具存入程序存储器 ROM。MCS-5 指令系统中有两条指令具有极强的查表功能。

1. MOVC A，@A+DPTR

该指令以数据地址指针 DPTR 内容作基址，它指向数据表格的首址，以变址器 A 的内容为所查表格的项数（即在表格中的位置是第几项）。执行指令时，基址加变址，读取表格中的数据，（A+DPTR）内容发送 A。

该指令以数据地址指针 DPTR 内容,可在 64K 程序存储器范围内查表，故称为长查表指令。

2. MOVC A，@A+PC

该指令以程序计数器 PC 内容作基础，以变址器 A 内容为项数加变址调整值。执行指令时，基址加变址，读取表格中护具，（A+PC）内容送 A。

变址调整值即 MOVC A，@A+PC 指令执行后的地址到表格首址之间的距离，即两地址之间其他指令所占的字节数。

用 PC 内容作基础查表只能查距离本指令 256 个字节以内的表格数据，被称为页内查表指令或短查表指令。执行该指令时，PC 当前值是由 MOVC A，@A+PC 指令在程序中的位置加 2 以后决定的，还要计算变址调整值，使用起来比较麻烦。但是不影响 DPTR 内容，使程序具有一定的灵活性，扔是一种常用的查表方法。

注意：如果数据表格存放在外部程序存储器中，执行这两条查表指令时，均会在控制引脚 *PSEN* 上产生一个程序存储器读信号。

5、子程序的设计及其调用

（1）子程序的概念

在一个程序中，往往许多地方需要执行同样的运算和操作。例如，求三角函数和各种加减乘除运算，代码转换以及延时程序等。这些程序是在程序设计中经常可以用到的。如果编程过程中每遇到这样的操作都编写一段程序，会使编程工作十分繁琐，也会占用大量程序存储器。通常把这些能完成某种基本操作并具有相同操作的程序段单独编制成子程序，以供

不同程序或同一程序反复调用。在程序中需要执行这种操作的地方执行一条调用指令，转到子程序中完成规定操作，并返回到原来的程序中继续执行下去。这就是所谓的子程序结构。在程序设计中恰当地使用子程序有如下优点。

①不必重复书写同样的程序，提高编程效率。

②程序的逻辑结构简单，便于阅读。

③缩短了源程序和目标程序的长度，节省了程序存储器空间

④使程序模块化、通用化，便于交流，共享资源。

⑤便于按某种功能调试。

通常人们将一些常用的标准子程序驻留在 ROM 或外部存储器中，构成子程序库丰富的子程序库对用户十分方便，对某子程序的调用，就像使用一条指令一样方便。

（2）调用子程序的要点

①子程序结构

用汇编语言编制程序时，要注意以下两个问题。

A、子程序开头的标号区段必须有一个使用户了解其功能的标志(或称为名字)，该标志即子程序的人口地址。以便在主程序中使用绝对调用指令 ACALL 或长调用指令 LCALL 转入子程序，例如调用延时子程序。

这两条调用指令属于程控类(转子)指令，不仅具有寻址子程序人口地址的功能，而且在转入子程序之前能自动使主程序断点人栈，具有保护主程序断点的功能。

B、子程序结尾必须使用一条子程序返回指令 RET。它具有恢复主程序断点的功能，以便断点出栈送 PC，继续执行主程序。

一般来说，子程序调用指令和子程序返回指令要成对使用。

②参数传递

子程序调用时，要特别注意主程序与子程序的信息交换问题。在调用一个子程序时，主程序应先把有关参数(子程序人口条件)放到某些约定的位置，子程序在运行时，可以从约定的位置得到有关参数。同样子程序结束前，也应把处理结果(出口条件)送到约定位置。返回后，主程序便可从这些位置中得到需要的结果，这就是参数传递。参数传递可采用多种方法。 ’

A、子程序无须传递参数

这类子程序中所需参数是子程序赋予，不需要主程序给出。

例 4.1.7 调用延时 20ms 子程序 DELY。

主程序：

```
        ……
        LCALL   DELY
        ……
```

子程序：

```
DELY：   MOV    R7，#100
DLY0：   MOV    R6，#98
         NOP
DLY1：   DJNZ   R6，#DLY1
         DJNZ   R7，#DYL0
         RET
```

子程序根本不需要主程序提供入口参数，从进入子程序开始，到子程序返回，这个过程即花费 CPU 时间约 20ms。

B、用累加器和工作寄存器传递参数

这种方法要求所需的人口参数，在转子之前将它们存人累加器 A 和工作寄存器 R0～R7 中。在子程序中就用累加器 A 和工作寄存器中的数据进行操作，返回时，出口参数即操作结果就在累加器和工作寄存器中。采用这种方法，参数传递最直接最简单，运算速度最高。但是工作寄存器数量有限，不能传递更多的数据。

例 4.1.8 双字节求补子程序 CPLD。

人口参数：(R7R6)＝16 位数。

出口参数：(R7R6)＝求补后的 16 位数。

```
CPLD:   MOV     A,R6
        CPL     A
        ADD     A,#1
        MOV     R6，A
        MOV     A,R7
        CPL     A
        ADDC    A,#0
        MOV     R7，A
        RET
```

这里采用“变反+1”的方法，值得注意的是十六位数变反加 1 要考虑进位问题，不仅低字节要加 1，高字节也要加低字节的进位，故采用 ADD A，#1 指令，而不能用 INC 指令，因为 INC 指令不影响 CY 位。

C、通过操作数地址传递参数

子程序中所需操作数存放在数据存储器 RAM 中。调用子程序之前的人口参数为 R0、R1 或 DPTR 间接指出的地址；出口参数(即操作结果)仍为 R0、R1 或 DPTR 间接指出的地址。一般内部 RAM 由 R0、R1 作地址指针，外部 RAM 由 DPTR 作地址指针。这种方法可以节省传递数据的工作量，可实现变字长运算。

例 4.1.9 n 字节求补子程序。

入口参数：(R0)＝求补数低字节指针，(R7)=n-1

出口参数：(R0)＝求补后的高字节指针。

```
CPLN:   MOV     A，@R0
        CPL     A
        ADD     A,#1
        MOV     @R0，A
NEXT:   INC     R0
        MOV     A,@R0
        CPL     A
        ADDC    A,#0
        MOV     @R0,A
        DJNZ    R7.NEXT
        RET
```

D、通过堆栈传递参数

堆栈可用于参数传递，在调用子程序前，先把参与运算的操作数压人堆栈。转入子程序之后，可按堆栈指针 SP 间接访问堆栈中的操作数，同时又可以把运算结果推入堆栈中。返回主程序后，可用 POP 指令获得运算结果。这里值得注意的是：转子时，主程序

的断点地址也要压人堆栈，占用堆栈两个字节，弹出参数时要用两条 DEC SP 指令修改 SP 指针，以便使 SP 指向操作数。另外在子程序返回指令 RET 之前要加两条 INC SP 指令，以便使 SP 指向断点地址，保证能正确返回主程序。

例 4.1.10 在 HEX 单元存放两个十六进制数，将它们分别转换成 ASCH 码并存入 ASC 和 ASC+1 单元。

由于要进行两次转换，故可调用查表子程序完成。

主程序：

```
MAIN：        ……
              PUSH      HEX         ;取被转换数
              LCALL     HASC        ;转子
*  PC→  POP       ASC          ; ASCL  ASC
              MOV       A,HEX       ;取被转换数
              Swap      A           ;处理高 4 位
              PUSH      ACC
              LCALL     HASC        ;转子
              POP       ASC+1       ;ASCH  ASC+1
```

在主程序中设置了人口参数 HEX 人栈，即 HEX 被推入 SP+1 指向的单元，当执行 LCALL HASC 指令之后，主程序的断点地址 x PC 也被压人堆栈，即 x PCL 被推人 SP+2 单元 x PCH 被推人 SP+3 单元。堆栈中的数据变化如图 4.6 所示。

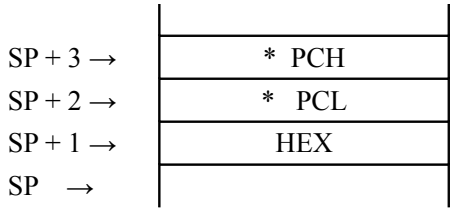


图 4.6

子程序

```
HASC      DEC     SP
          DEC       SP          ;修改 SP 指向 HEX
          POP       ACC         ;弹出 HEX
          ANL       A,#0FH      ;屏蔽高四位
          ADD       A,#5         ;变址调整
          MOVC      A,@A+PC     ;查表
          PUSH      ACC         ;结果入栈              （2 字节）
          INC       SP          ;              (1 字节)
          INC       SP          ;修改 SP 指向断点位置（1 字节）
          RET
ASCTAB    DB        ’0 1 2 … 7’
          DB        ’8 9 A … F’
```

使用堆栈来传递参数，方法简单，能传递大量参数，不必为特定参数分配存储单元。

（3）现场保护

在转入子程序时，特别是进人中断服务子程序时，要特别注意现场保护问题。即主程序使用的内部 RAM 内容，各工作寄存器内容，累加器 A 内容和 DPTR 以及 PSW 等寄存器内容，都不应因转子程序而改变。如果子程序所使用的寄存器与主程序使用的寄存器

有冲突，则在转人子程序后首先要采取保护现场的措施。方法是将要保护的单元推人堆栈，而空出这些单元供子程序使用。返主程序之前要弹出到原工作单元，恢复主程序原来的状态，即恢复现场。

例如，十翻二子程序的现场保护。

```
BCDCB    PUSH    ACC
         PUSH    PSW
         PUSH    DPL      ;保护现场
         PUSH    DPH
         ……               ;十翻二
         POP     DPH
         POP     DPL
         POP     PSW      ;恢复现场
         POP     ACC
         RET
```

推入与弹出的顺序应按“先进后出”，或“后进先出”的顺序，才能保证现场的恢复。对于一个具体的子程序是否要进行现场保护，以及哪些单元应该保护，要具体情况待，不能一概而论。

（4）设置堆栈

恰当的设置堆栈指针 SP 的初始值是十分必要的。调用子程序时，主程序的断点将自动入栈，转子后，现场的保护都要占用堆栈工作单元，尤其多重转子或子程序嵌套，需要使栈区有一定的深度。由于 MSC-51 的堆栈是由 SP 指针组织的内部 RAM 区，仅有 128 个单元堆栈并非越深越好，深度要恰当。

3、子程序的调用及嵌套

（1）子程序调用

一个子程序可以供同一程序或不同程序多次调用或反复调用而不会被破坏，不仅给程序设计带来极大灵活性，方便了用户，而且简化了程序设计的逻辑结构，节省了程序存储器空间。

例 4.1.11 将内部 RAM 4lH～43H 中内容左移 4 位，移出部分送 40H 单元。

由于多字节移位是程序设计中经常用到的，有一定普遍性。为了给程序设计带来灵活性，试编制一个“n 字节左移一位’’子程序，反复调用 4 次即为”n 字节左移 4 位‘

功能：n 字节左移一位。

入口：(R0)指向内部 RAM 的操作数地址，高字节在先

(R4)＝字节长度。

出口：(R0)指向内部 RAM 的结果地址，低字节在先。

子程序：

```
RLC1:    CLR    C
LOOP0:   MOV    A,@R0
         RLC    A
         MOV    @R0,A
         DEC    R0
         DJNZ   R4,LOOP0
         MOV    A,@R0
         RLC    A
         MOV    @R0,A
```

```
                RET
```

为了完成例 4.1.11 的要求，可编制左移 4 位子程序。

```
RLC4:       MOV       R7,#4              ;R7 为左移位数计数器
NEXT:   MOV     R0,#43      ;为进入 RLC1 子程序
            MOV       R4#3          ;设置入口条件
            ACALL     RLC1          ;转子
*PC→        DJNZ    R7,NEXT     ;未完，继续
            MOV     A,@R0
            ANL     A,#0FH        ;屏蔽结果高 4 位
            MOV     @R0,A         ;存结果高 4 位
            RET
```

注意：*PC 是子程序的返回地址，即当前主程序的断点。

在这个简单的子程序中，由于子程序 RLC1 和主程序 RLC4(相对于子程序 RLC1 而言)所用的寄存器没有冲突，即调用子程序 RLC1 时，主程序 RLC4 的现场没有被破坏，因此无须在子程序 RLC1 中保护现场。否则将在 RLC1 的人口用 PUSH 指令保护现场，在 RET 指令之前，用 POP 指令恢复现场。

在这个例子中，不难看出参数传递的方式，采用了地址传递参数方式和工作寄存器参数传递方式。人口参数是由 R0 给出的地址指针，指向内部 RAM 中操作数的低字节，由 R4 给出字节长度。出口参数也是由 R0 给出的地址，它指向结果存放 RAM 的高字节。

子程序调用指令 ACALL(LCALL)不仅具有寻址子程序人口地址的功能，而且能在转入子程序之前，利用堆栈技术自动将断点‘PCL 一(SP+1)，XPCH 一(SP+2)推人堆栈有效保护了断点。当子程序返回，执行 R 耳 T 指令时，能使断点出栈送人 PC，即返回到主程序继续执行。

（4）子程序嵌套

主程序与子程序的概念是相对的，一个子程序除了末尾有一条返回 RET 指令外，其本身的执行与主程序并无差异，因而在子程序中完全又可以引用其他子程序，这种情况称为子程序嵌套或多重转子。

例如在一个数据处理的程序中，经常要调用“左移 4 位”子程序 RLC4。数据处理程序如下。

主程序：

```
MAIN:   MOV     SP,#5FH             ;数据
            ……                  ;处理
            ACALL   RLC4          ;程序
            ……                          ;
```

这个程序就采用了子程序的嵌套。为什么要在主程序的第一条指令就要定义堆栈指针呢?因为子程序的嵌套必须借助堆栈来完成。

多次调用子程序伴随着多次子程序返回操作，每次调用指令都有一个断点人栈操作，每次的返回指令都有一个断点出栈操作。而最后一次被调用的子程序返回地址，必须最先被弹出才能保证程序的正确性。换句话说，这时保护人栈的断点地址及从栈中弹出的返回地址必须按照“先进后出”(或后进先出)的操作次序，这种操作恰好是堆栈操作的原则。

堆栈与子程序调用的关系。每一次调用子程序，都要将断点压入堆栈，并自动修改(加 2)SP 指针；每一次返回都要将断点弹出，SP 自动减 2。调用和返回总是成对进行的，保证

了堆栈里的数据(断点地址)有秩序地进出。

中断响应与中断返回和子程序调用与子程序返回具有相同的过程。所不同的是调用指令CALL 是编程者在程序中安排的，断点为已知固定的；而中断响应是随机的，因而中断的断点地址也是随机的。有了堆栈技术，不管断点是固定的还是随机的，都可以得到有效的保护和恢复。这里要强调的是伴随着断点的进出栈，SP 指针也将不断地得到修正，它总是指向栈顶。

4.1.4 汇编语言源程序设计举例

这一节将进一步讨论常用的程序设计方法，包括算术运算、代码转换等。结合这些程序，可进一步分析熟悉 MCS-51 指令系统，掌握汇编语言程序设计方法和技巧。

1、算术运算程序

算术运算程序包括各种有符号数或无符号数的加减乘除，这类程序很多。

例 4.1.12 多字节无符号数加法子程序 NADD。

功能：n 字节无符号数加法。

入口：(R0)＝被加数低字节地址指针，(R1)=加数低字节地址指针

(R7)＝字节数 n。

出口：(R0)＝和数高字节地址指针。

若两数相加，和数可能为 n+1 字节，若无进位，则第(n+1)字节为 0；若有进位，则第（n+1）字节为 1。

程序流程图如图 4.7 所示。

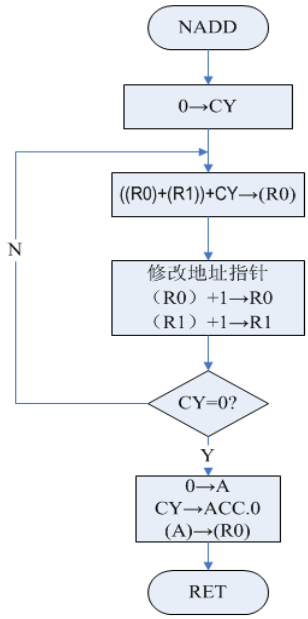


图 4.7 NADD 子程序流程图

```
NADD:   CLR     C
LOOP:   MOV     A,@R0
        ADDC    A,@R1
        MOV     @R0,A
        INC     R0
```

```
INV     R1
DJNZ    R7，LOOP
CLR     A
MOV     ACC.0，C
RET
```

子程序执行后，被加数被冲掉。

2、代码转换程序

在计算机内部，数据的运算一般都采用二进制，二进制具有运算方便、占用的内存单元少等特点。而人们习惯用十进制数，因此计算机的输入输出通常采用 BCD 码和 ASCH 码。对于这样一些代码需要进行代码转换。

例 4.1.13 4 位十进制数转换为二进制数子程序 BCDCB。

BCD 码存放格式如下。

@R0 →	0 千
@R0+1 →	0 百
@R0+2 →	0 十
@R0+3 →	0 个

功能：4 位非压缩 BCD 数转换为二进制数。

入口：(R0)＝内部 RAM 中 BCD 码高字节地址指针

(R7)＝BCD 码位数减 1，即 n＝3。

出口：(R3R4)＝转换结果。

4 位十进制数可表示为多项式

$$A = a_3 \times 10^3 + a_2 \times 10^2 + a_1 \times 10^1 + a_0$$

$$= ((a_3 \times 10 + a_2) \times 10 + a_1) \times 10 + a_0$$

BCD 码存放在内部 RAM 4 个相邻的单元中。按上述多项式计算，应先取 BCD 码的千位数乘以 10，加上百位数再乘以 10，再加上十位数乘以 10，最后加上个位数，即为转换结果。

程序流程图如图 4.8 所示。

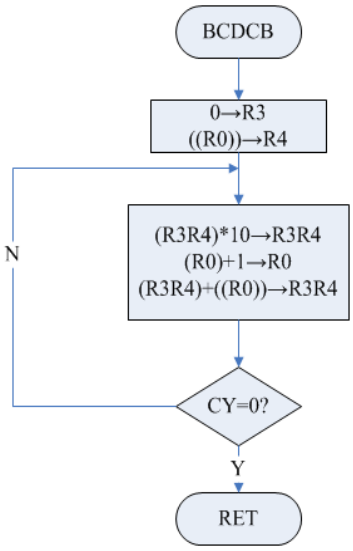


图 4.8 BCDCB 子程序框图

```
BCDCB：     MOV     R3，#0      ；结果高字节清零
            MOV     A／QR0      ；取 BCD 码
            MOV     R4，A       ；暂存
BCDCL：     MOV     A,R4
            MOV     B,#10
            MUL     AB
            MOV     R4,A        ；暂存 R4*低 8 位
            MOV     A,B
            XCH     A,R3        ；暂存 R4 *10 高 8 位
            MOV     B,#10
            MUL     AB          ；R3）*为一字节数
            ADD     A,R3        ；（R3）*10+（R4）*10 高 8 位
            MOV     R3，A       ；暂存
            INC     R0
            MOV     A，R4
            ADD     A,@R0
            MOV     R4,A
            MOV     A,R3
            ADDC    A,#3        ；（R3R4）+((R0))+CY---R3R4
            MOV     R3,A
            DJNZ    R7,BCDCL    ；未完，继续
            RET
```

例 4.1.14 多位十六进制数转换为 ASCⅡ 码子程序 HEXASC。
功能：n 位十六进制数转换为 ASCII 码。
入口：(R0)十六进制数低字节地址指针，(R7)＝字节数。
出口：(R1)＝ASCII 码存放地址指针。
计算法，采用

$$(XH+90H)\rightarrow \text{十进制调整}\rightarrow (XD'+40H+cy)\rightarrow \text{十进制调整}\rightarrow XD$$

其中 XH 为十六进制数，XD'、XD 为十进制数。

当 $XH\le 9$ 时，第一次十进制调整的结果 $XD'\le 99$ $cy=0$。

当 $XH>9$ 时，$cy=1$，在第二次调整前把 cy 加进去。这样，累加器 A 的内容就是 0～F 的 ASCII 码。

```
HEXASC：    MOV     A,@R0              ；取数
            ANL     A,#00001111B       ；处理低 4 位
            ADD     A,#90H
            DA      A
            ADD     A,#40H
            DA      A                  ；转换成 ASCII 码
            MOV     @R1,A              ；存放 ASCII 码
            INC     R1                 ；修正 R1 指针
            MOV     A，@R0             ；修正 R1 指针
            SWAP    A                  ；处理高 4 位
            ANL     A，#15
            ADD     A，#90H
            DA      A
            ADDC    A，#40H
            DA      A                  ；转换成 ASDII 码
            MOV     @Rl，A             ；存放 ASCII 码
            INC     R0                 ；修正 R0 指针
            INC     Rl                 ；修正 R1 指针
            DJNZ    R7，HEXASC
            DEC     R1                 ；修正 R1 指针
            RET
```

项目 4．2 阵列彩灯程序

4.2.1 阵列彩灯控制要求分析

阵列彩灯具有比较好的装饰效果，一般可用于广告宣传、店铺装饰、舞台灯光等场合。本阵列彩灯由单片机控制 16 单元彩灯，可实现多种动态变化效果，实现彩灯的左移，右移、由两侧想中间移动等不同的变化。

阵列彩灯控制要求如下：

花样一：一盏灯从右至左点亮，即从 LED1 开始向 LED16　方向流水移动点亮。

花样二：一盏灯从左至右点亮，即从 LED16 开始向 LED1 方向流水移动点亮。

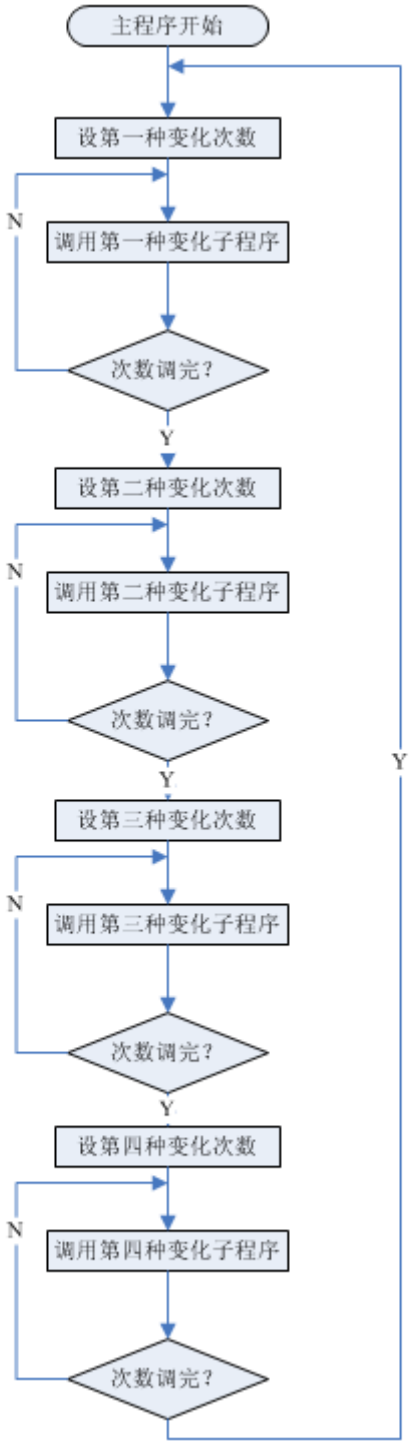
花样三：让一盏彩灯要求一盏灯从左至右逐个点亮不灭，即从 LED16 开始向 LED9 方向逐个点亮不熄灭，直到全部点亮后熄灭；然后让另一个组彩灯要求一盏灯从左至右逐个

点亮，即从 LED8 开始向 LED1 方向逐个点亮不熄灭，直到全部点亮。

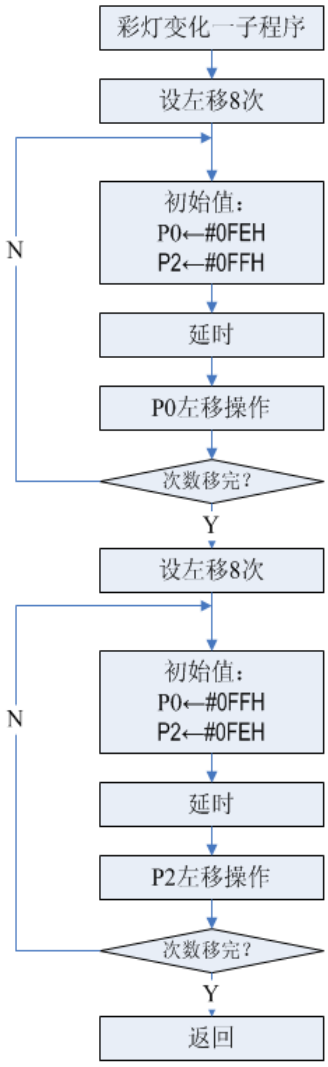
花样四：让全亮的等从右至左逐个熄灭，即从 LED1 开始向 LED16 方向，彩灯逐个熄灭。

以上四种花样每种循环四次，周而复始。

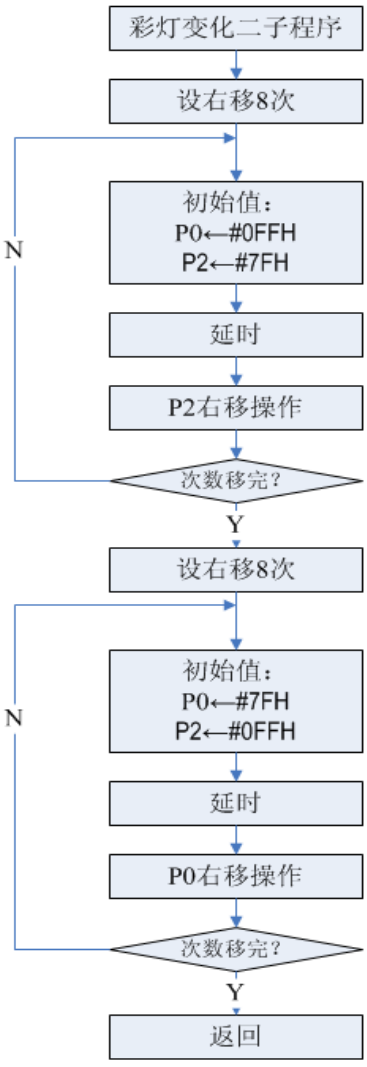
4.2.2 绘制流程图



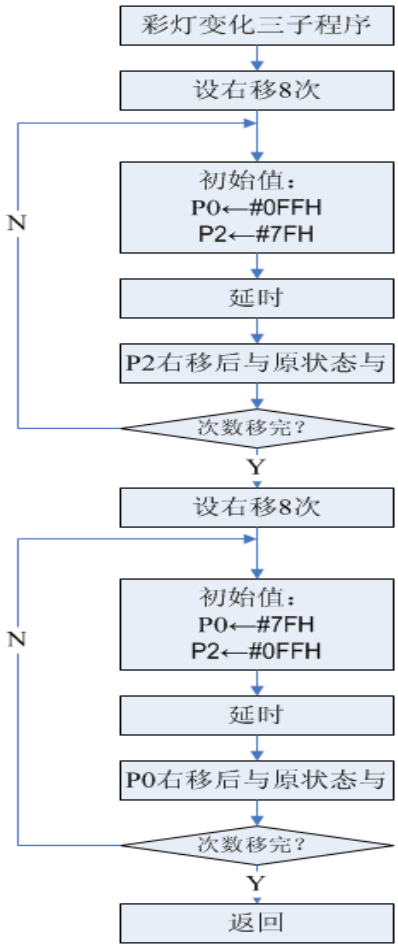
（a）主程序



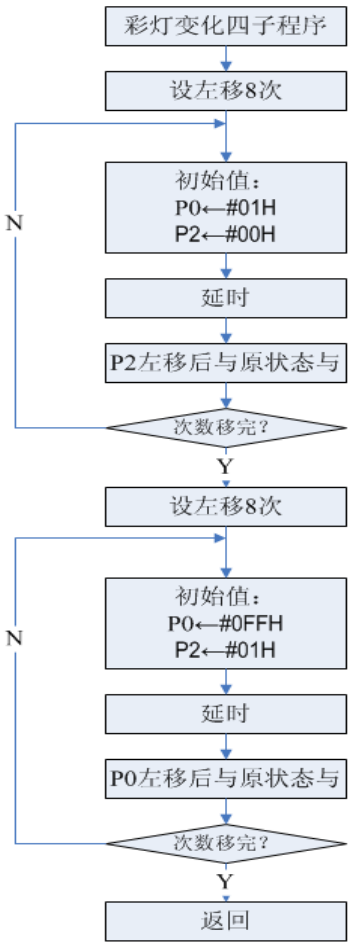
（b）变化一子程序



（c）变化二子程序



（d）变化三子程序



（e）变化四子程序

图 4.9 阵列彩灯控制程序流程图

4.2.3 编写应用程序

```
；程序开始**********************************************************************
*
        ORG 0000H                       ；程序入口地址
        AJMP 主程序                     ；跳转主程序
        ORG 0050H           ；主程序入口地址
；————————————————————————————————————————————————————————
—
主程序：                                ；主程序开始
        MOV R2，#04H            ；给 R2 赋值，设置循环调用次数
；————————————————————————————————————————————————————————
—
彩灯变化一循环程序：
        ACALL 彩灯变化一子程序          ；调用彩灯变化一子程序
        DJNZ R2，彩灯变化一循环程序     ；没完继续调用
        MOV R2，#04H                    ；设置循环调用次数
```

```
; ————————————————————————————————————————————————————————————
—
彩灯变化二循环程序：
    ACALL 彩灯变化二子程序          ；调用彩灯变化二子程序
    DJNZ R2，彩灯变化二循环程序     ；没完继续调用
    MOV R2，#04H                ；设置循环调用次数
; ————————————————————————————————————————————————————————————
—
彩灯变化三循环程序：
    ACALL 彩灯变化三子程序                  ；调用彩灯变化三子程序
    DJNZ R2，彩灯变化三循环程序             ；没完继续调用
    MOV R2，#04H                ；设置循环调用次数
彩灯变化四循环程序：
    ACALL 彩灯变化四子程序                  ；调用彩灯变化三子程序
    DJNZ R2，彩灯变化四循环程序             ；没完继续调用
    AJMP 主程序                             ；  跳转至主程序
; ————————————————————————————————————————————————————————————
—
彩灯变化一子程序：
    MOV R3，#08H                ；设置 P0 口显示次数
    CLR A                       ；累加器 A 清 0
    CPL A                       ；累加器 A 取反
    MOV P2，A                   ；P2 口彩灯全不亮
    MOV P0，#0FEH               ；P0 口第一位发亮
; ————————————————————————————————————————————————————————————
—
彩灯变化一之 P0 口显示程序
    ACALL 延时子程序                        ；延时
    MOV A，P0                   ；P0 口内容送累加器 A
    RL A                        ；累加器 A 中内容向左移
    MOV P0，A                   ；送 P0 口显示
    DJNZ R3，彩灯变化一                     ；是否显示了 8 次?否，跳转彩灯变化
    MOV R3，#08H                ；显示程序继续；是，顺序；执行
    MOV P0，#0FFH               ；设置 P2 口显示次数
    MOV P0,#0FFH                ；P0 口彩灯全部亮
    MOV P2，A                   ；送 P2 口显示
    DJNZ R3，彩灯变化三之 P0 口显示程序     ；是否显示了 8 次？跳转彩灯变化之 P0
                                            ；口，显示程序继续；是，顺序执行
    MOV R3,#08H                 ；设置 P2 口显示次数
    MOV P0，#0FFH               ；P0 口彩灯全不亮
    MOV P2, #7FH                ；p2 口第一位彩灯发光
; ————————————————————————————————————————————————————————————
—
彩灯变化一之 P2 口显示程序：
```

```
    ACALL 延时子程序                          ；延时
    MOV A，P2                  ；将 P2 口内容送累加器 A
    RL  A                    ；累加器的内容左移一位
    MOV P2，A                  ；送 P2 口显示
    DJNZ  R3，彩灯变化一之 P2 口显示程序      ；是否显示了 8 次?否，跳转彩灯变化
                                          ；之 P2 口显示程序继续；是，顺序执行
    RET                                   ；返回主程序
；————————————————————————————————————————
—
彩灯变化二子程序：
    MOV  R3，#08H              ；设置 P2 口显示次数
    MOV  P2，#7FH              ；P2 口第 8 位发亮
    MOV  P0，#0FFH             ；P0 口全不亮
；————————————————————————————————————————
—
彩灯变化二之 P2 口显示程序：
    ACALL 延时子程序                          ；延时
    MOV、A，P2                 ；将 P2 口内容送累加器 A
    RR  A                    ；累加器的内容左移一位
    MOV  P2，A                 ；送 P2 口显示
    DJNZ  R3，彩灯变化二              ；是否显示了 8 次？否，跳转彩灯变化二
                                          ；P2 口，显示程序继续；是，顺序执行
    MOV R3，#08H               ；设置 P0 口显示次数
    MOV P0，#7FH               ；P0 口第 8 位亮
    MOV P2，#0FFH              ；P2 口全不亮
；————————————————————————————————————————
—
彩灯变化二之 P0 口显示程序
    ACALL 延时子程序                          ；延时
    MOV  A，P0                 ；P0 口的内容送累加器
    RR  A                    ；累加器的内容右移一位
    MOV  P0，A                 ；送 P0 口显示
    DJNZR3，彩灯变化二之 P2 口显示程序        ；是否显示了 8 次？否，跳转彩灯变化
二                                        ；P2 口；显示程序继续；是，顺序执行
    RET                      ；返回主程序
；————————————————————————————————————————
—
彩灯变化三子程序：
    MOV  R3，#08H              ；设置 P2 口显示次数
    MOV  P0，#0FFH                ；P0 口全部亮
    MOV  P2，#7FH              ；P2 口的第 8 位发亮
；————————————————————————————————————————
—
彩灯变化三之 P2 口显示程序：
```

```
    ACALL 延时子程序                          ；延时
    MOV  A，P2                  ；将 P2 口的内容送累加器 A
    RR  A                     ；累加器 A 中的内容向右移一位
    ANL  A，P2                  ；P2 口的内容与累加器 A 的内容逻辑与
                                          ；后送累加器 A
    MOV P2，A                   ；送 P2 口显示
    DJNZR3，彩灯变化三之 P2 口显示程序        ；是否显示了 8 次？否,跳转彩灯变化三
                                          ；P2 口，显示程序继续；是，顺序执行
    MOV R3，#08H                ；设置 P0 口显示次数
    MOV P0，#7FH                ；P0 口第 8 位全亮
    MOV P2，#00H                ；P2 口全亮
；————————————————————————————————————————
—
彩灯变化三之 P0 口显示程序
    ACALL 延时子程序                          ；延时
    MOV A，P0                   ；将 P0 口的内容送累加器 A
    RR A                      ；累加器 A 的内容右移一位
    ANL A，P0                   ；P0 口的内容与累加器 A 的内容逻辑与
                                ；后送累加器 A
    MOV P0，A                 ；送 P0 口显示
    DJNZ R3，彩灯变化三之 P0 口显示程序       ；是否显示了 8 次？否,跳转彩灯变化之
                                          ；三 P0 口显示程序继续；是，顺序执行
    RET                       ；返回主程序
；————————————————————————————————————————
—
彩灯变化四子程序：
    MOV R3，#08H                  ；设置 P0 口显示次数
    MOV P0，#01H                ；P0 口第一位熄灭
    MOV P2，#00H                  ；P2 口全亮
    MOV A，P0                   ；将 P0 口的内容送累加器 A
    XRL A，P0                     ；P0 口的内容与累加器 A 的内容逻辑异
                                  ；或后送累加器 A
    MOV P2，A                     ；将异或后内容送 P2 口，即 P2 口全亮
；————————————————————————————————————————
—
彩灯变化四之 P0 口显示程序
    ACALL 延时子程序                          ；延时
    MOV A，P0                     ；将 P0 口的内容送累加器 A
    ORL A，P0                     ；P0 口的内容与累加器 A 的内容逻辑或
                                  ；后送累加器 A
    MOV P0，A                     ；送 P0 口显示
    DJNZ R3，彩灯变化四之 P0 口显示程序       ；是否显示了 8 次？否,跳转彩灯变化之
                                          ；四 P0 口显示程序继续；是，顺序执行
    MOV R3，#08H                  ；  设置 P2 口显示次数
```

```
    MOV P0，#0FFH                    ；P0 口全熄灭
    MOV P2，#01H                     ；P2 口第一位熄灭
；————————————————————————————————————————————————
—
彩灯变化四之 P2 口显示程序：
    ACALL 延时子程序                              ；延时
    MOV A，P2                ；P2 口的内容送累加器 A
    RL A                     ；累加器的内容左移一位
    ORL A，P2                ；P2 口的内容与累加器 A 的内容逻辑或
                                                  ；后送累加器 A
    MOV P2，A                ；送 P2 口显示
    DJNZ R3，彩灯变化四之 P2 口显示程序      ；是否显示了 8 次?否，跳转彩灯变化四
                                                  ；之 P2 口显示程序继续；是，顺序执行
    RET                      ；返回主程序
；————————————————————————————————————————————————
—
延时子程序：                                      ；延时子程序
    HOV R5，#64H
延时子程序第一循环：
    MOV R4，#OF8H
延时子程序第二循环：
    DJNZ R4，延时子程序第二循环
    NOP
    DJNZ R5，延时子程序第一循环
    RET
；————————————————————————————————————————————————
—
    END
；程序结束
```

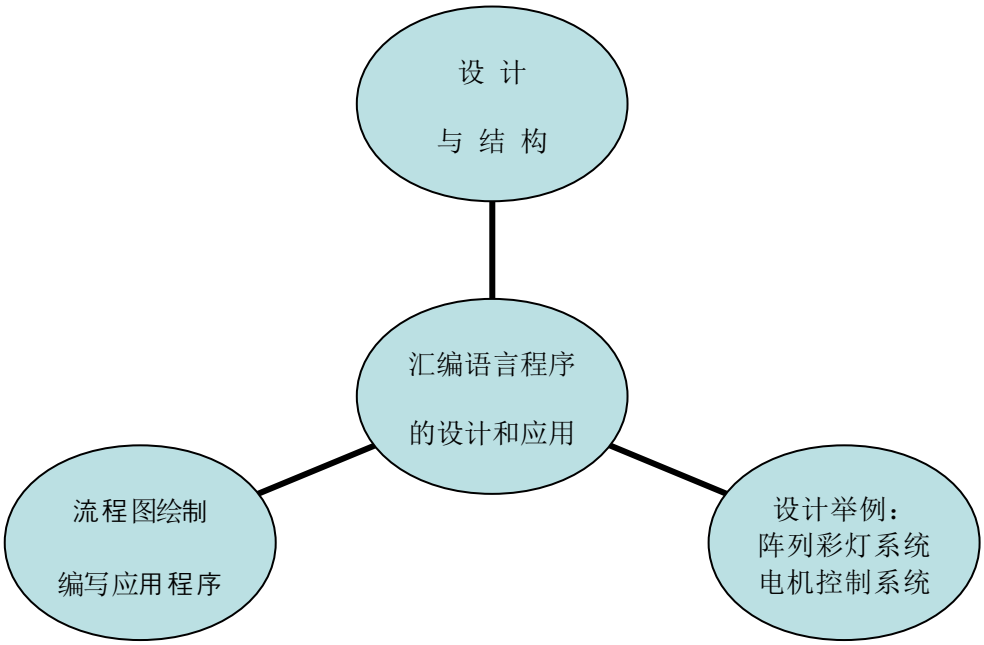
4.2.3 硬件组成

完成上述产品的制作需要以下元件，具体见表 4.1

表 4.1 阵列彩灯硬件电路元器件表

名称	代号	规格或型号	数量
单片机	U1	AT89S51	1
晶振	B	12MHz	1
发光二极管	LED1–LED16	ф5mm	16
按钮	SB	轻触型	1
电阻	R1–R18	470Ω 1/8W	18
电容	C_1	10μF / 16V	1
	C_2、C_3	30pF	2
单片机电源	V_{CC}	+5V	1

重点串联：



基础训练：

一、填空题

1、指令 MOVC A， @A+PC 的功能是_____。

2、指令 JBC CY, LOOP 是_____字节、_____个机器周期指令。

3、调用子程序时，将 PC 当前值保存到_____。

4、MCS-51 单片机堆栈操作的基本原则是_____。

二、简答题

1、若 SP=25H，标号 LABEL 的值为 3456H。指令“LCALL LABEL”的首地址为 2345H,问执行长调用指令“LCALL LABEL”后，堆栈指针和堆栈的内容发生什么变化？PC 的值等于什么？

2、已知 SP=25H，PC=2345H，（24H）=12H，（25H）=34H，（26H）=56H。问执行“RET”指令以后， SP=？ PC=？

3、以下程序段执行后，A= ， （30H）= 。

MOV 30H，#0AH
MOV A，#0D6H
MOV R0，#30H
MOV R2，#5EH
ANL A，R2
ORL A，@R0
SWAP A
CPL A
XRL A， #0FEH
ORL 30H， A

4、比较内部 RAM 中 30H 和 40H 单元的二个无符号数的大小，将大数存入 20H 单元，小数存入 21H 单元，若二数相等，则使位空间的 7FH 位置 1。

5、设变量 X 存在内部 RAM 的 20H 单元中，其取值范围为 0－5，编一查表程序求其平方值，并将结果存放在内部 RAM 21H 单元。

6、将一个字节的二进制数转换成 3 位非压缩型 BCD 码。设该二进制数在内部 RAM 40H 单元，转换结果放入内部 RAM50H,51H,52H 单元中（百位在 50H，十位在 51H,个位在 52H）。

答案：

一、填空题

1、将 A 的内容与 PC 当前值相加作为程序存储器地址，再将该地址单元的内容传送到 A

2、3,2　　3、堆栈　　4、先进后出

二、简答题

1、PC 当前值压入堆栈，并转向子程序，SP=27H, (26H)=48H,(27)=23H,PC=3456H

2、SP=23H,PC=3412H

3、A=E4H （30H）=EEH

4、
```
            ORG 0000H
             LJMP MAIN
             ORG 0100H
      MAIN：MOV  A，30H
             CJNE  A，40H，LOOP1
             SETB  7FH
             SJMP  LOOP3
   LOOP1：   JC    LOOP2
             MOV  20H，A
             MOV  21H，40H
             SJMP  LOOP3
   LOOP2：   MOV  20H，40H
             MOV  21H，A
   LOOP3:    SJMP  $
             END
```

```
5、      ORG 0000H
         LJMP START
         ORG 1000H
START: MOV DPTR, #TABLE
         MOV  A,20H
         MOVC A ,@A+DPTR
         MOV 21H, A
         SJMP $
         ORG 2000H
TABLE: DB 0,1,4,9,16,25
         END
```

```
6、    ORG  0000H
          LJMP  HEXBCD
          ORG   0100H
HEXBCD:  MOV  A, 40H◆
          MOV B, ＃100◆
          DIV  AB◆
          MOV 50H, A◆
          MOV A, ＃10◆
          XCH  A, B◆
          DIV  AB◆
          MOV 51H, A◆
          MOV 52H, B◆
          SJMP $
          END
```

技能训练:

【课时建议】 4 课时

实训 4.1 流水灯控制

1.实训目的

⑴ 掌握汇编语言的基本结构。

⑵ 了解汇编语言程序设计思维方法。

⑶ 设计两个开关，使 CPU 可以察知两个开关组合出的 4 种不同状态。然后对应每种状态，使 8 个 LED 显示出不同的亮灭模式。

2. 实训设备与器件

微机、Proteus 和 Keil 软件

3. 实训步骤

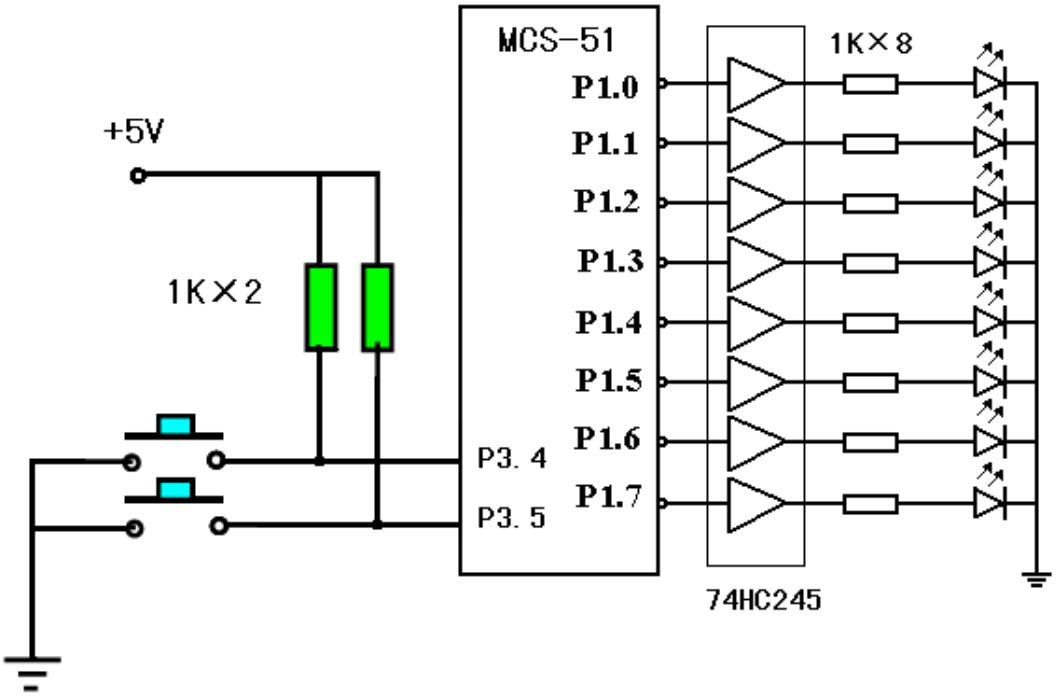
⑴ 在 keil C51 uVision3 软件开发平台上建立工程项目、编写程序

⑵ 在 Proteus 设计电路

⑶ Proteus 和 Keil 软件联合仿真，观察程序运行情况等。

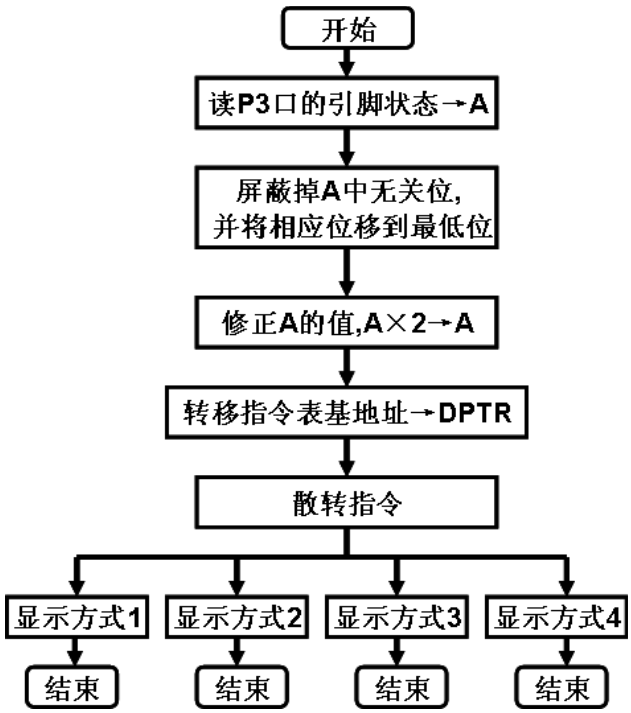
4. 实训电路

学生按照要求自己设计



电路图

5. 程序流程图



电路图

6. 程序清单

```
        ORG   0000H
        MOV   P3,#00110000B
        MOV   A,P3
        ANL   A ,#00110000B
        SWAP  A
        RL    A
        MOV   DPTR,#TABLE
        JMP   @A+DPTR
ONE:    MOV   P1,#00H
        SJMP  $
TWO:    MOV   P1,#55H
        SJMP  $
THREE:  MOV   P1,#0FH
        SJMP  $
FOUR:   MOV   P1,#0F0H
        SJMP  $
TABLE:  AJMP  ONE
        AJMP  TWO
        AJMP  THREE
        AJMP  FOUR
        END
```

实训 4.2　数码管的控制

1.实训目的

⑴ 掌握 BCD 码调整程序的设计。

⑵ 掌握用软件设计的延时子程序。

⑶ 用两个 8 段 LED 数码管组成电梯轿厢数码管指示层。两个数码管显示层数（十进制），显示十位数的数码管的小数点作为上行指示，显示个位数的数码管的小数点作为下行指示。

当轿厢停在某层时，数码管显示该层的层数，显示 m 秒,上行或下行指示灭；轿厢在两层之间运行时,数码管显示前方的层数, 上行或下行指示灯亮, 显示 m 秒。

2. 实训设备与器件

微机、Proteus 和 Keil 软件

3. 实训步骤

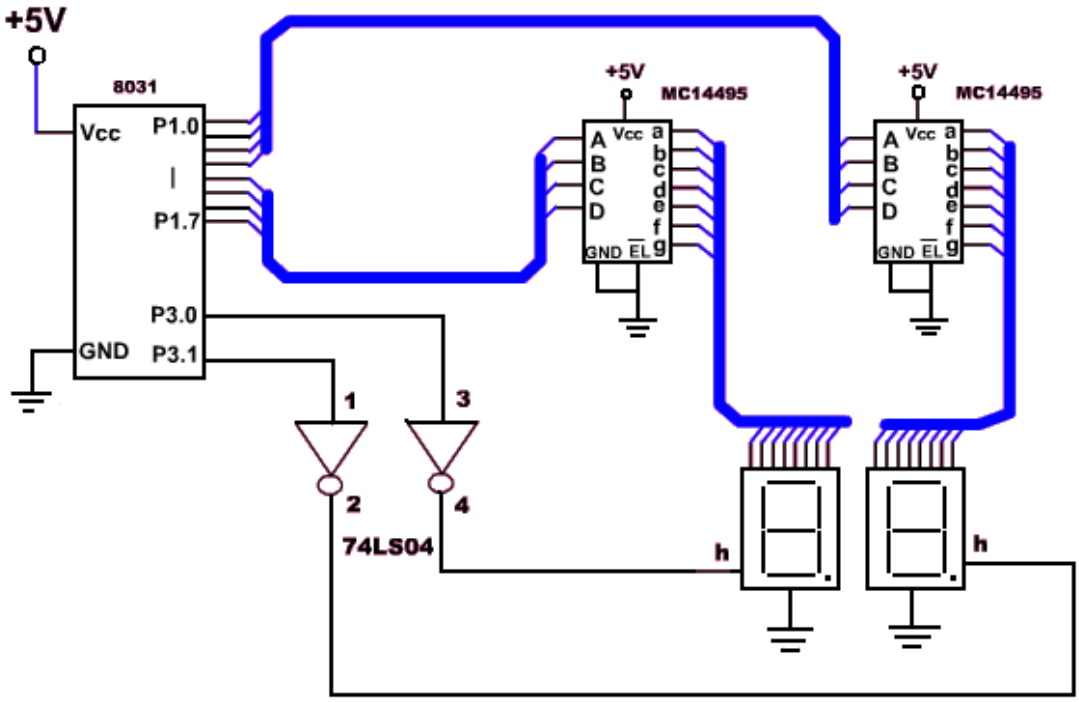
⑴ 在 keil C51 uVision3 软件开发平台上建立工程项目、编写程序

⑵ 在 Proteus 设计电路

⑶ Proteus 和 Keil 软件联合仿真，观察程序运行情况等。

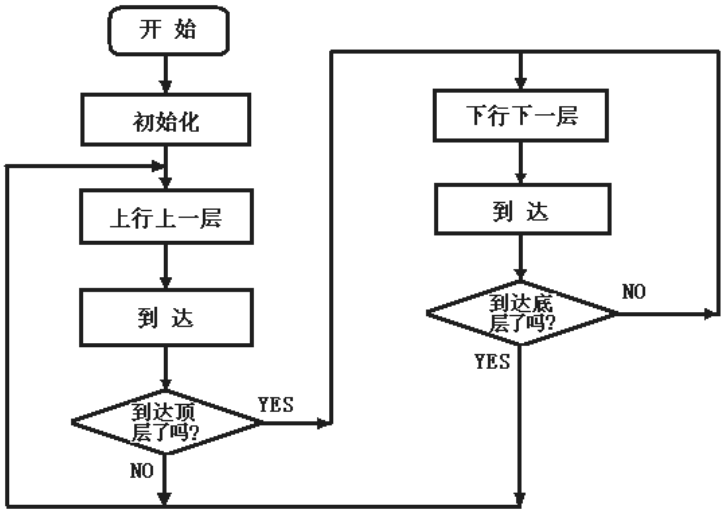
4. 实训电路

学生按照要求自己设计



电路图

5. 程序流程图



6. 程序清单

主程序:

```
        ORG   0000H
        AJMP  MAIN     ;程序入口
        ORG   0100H
MAIN:   MOV  R7,#7      ;循环次数(大循环)
```

```
        MOV  P1,#1       ;起始 1 层(显示 01)
        ACALL  DELAY   ;延时(停在第一层)
LOOP:  MOV  R5,#2     ;准备上行
UP:    ACALL  HTT     ;转换显示
        CLR  P3.0       ;上行中
        ACALL  DELAY    ;延时
        SETB  P3.0      ;到达停顿
        ACALL  DELAY    ;延时
        INC  R5         ;上一层
        CJNE  R5,#13,UP   ;最高层?
        MOV  R5,#11      ;准备下行
DN:    ACALL  HTT
        CLR  P3.1       ;下行中
        ACALL  DELAY
        SETB  P3.1      ;到达停顿
        ACALL  DELAY
        DEC  R5         ;下一层
        CJNZ  R5,#0,DN    ;底层?
        DJNZ  R7,LOOP    ;循环结束
        SJMP  $
```

转换显示子程序:

```
HTT:  MOV  A,R5   ;子程序的入口，
        MOV  B,#10
        DIV  AB    ；相除，商 A=01、余 B=00，
        SWAP  A    ；交换，A=10
        ORL  A,B    ；相加，A=10，
        MOV  P1,A    ；数码管显示 10（显示 12）
        RET
```

延时 3 秒子程序(12MHZ)

```
DELAY:  MOV  R2,#200
LP2:    MOV  R1,#30
LP1:    MOV  R0,#7DH
LP0:    NOP
        NOP
        DJNZ  R0,LP0
        DJNZ  R1,LP1
        DJNZ  R2,LP2
```

RET
END