模块3 认识单片机C语言程序设计

• **技能目标**

(1) 掌握任务 9-1　了解 C51 编程结构；

(2) 掌握任务 10-1　用不同数据类型控制 P2 口的 8 位 LED 闪烁；

(3) 掌握任务 11-1　分别用 P2、P3 口显示“加减”运算结果；

(4) 掌握任务 11-2　用 P1 口显示逻辑“与或”运算结果；

(5) 掌握任务 11-3　分别用 P2、P3 口显示位“与或”运算结果；

(6) 掌握任务 11-4　用 P1 口显示“左右移”运算结果；

(7) 掌握任务 12-1　用按键 S 控制 P1 口 8 只 LED 的显示状态；

(8) 掌握任务 12-2　用 for 语句实现蜂鸣器发出 1kHz 音频；

(9) 掌握任务 12-3　用 while 语句控制 P1 口 8 只 LED 的显示状态；

(10) 掌握任务 12-4　用 do…while 语句控制 P1 口 8 只 LED 的显示状态；

(11) 掌握任务 13-1　用数组控制 P1 口 8 只 LED 的显示状态；

(12) 掌握任务 14-1　用指针数组控制 P1 口 8 只 LED 的显示状态；

(13) 掌握任务 14-2　用指针数组实现多状态显示；

(14) 掌握任务 15-1　用带参数函数控制 8 位 LED 的闪烁时间；

(15) 掌握任务 15-2　用数组作为函数参数控制 8 位 LED 的点亮状态；

(16) 掌握任务 15-3　用指针作为函数参数控制 8 位 LED 的点亮状态；

(17) 掌握任务 15-4　用函数型指针控制 8 位 LED 的点亮状态；

(18) 掌握项目 16　用 P2 口控制 8 只 LED 左循环流水灯亮；

(19) 掌握项目 17　用开关 S 控制实现蜂鸣器报警。

• **知识目标**

学习目的：

(1) 理解 C 语言程序在结构上的特点和书写格式上的要求；

(2) 掌握数据类型的概念，了解 C51 语言能够处理的数据类型；

(3) 了解 C51 语言的基本运算符及其特点，掌握运算符的优先级和结合性；

(4) 理解算术表达式、关系表达式、逻辑表达式的特点，能熟练计算表达式；

(5) 掌握 if 语句、swith 语句的语法，能编写选择结构的程序，掌握 for 语句、while 语句、do…while 语句的使用语法及方法，能进行循环程序设计；

(6) 理解数组的概念，能定义、初始化一维数组、二维数组及字符数组，进行相关程序设计；

(7) 理解函数的概念，能根据需要定义一个函数，能正确调用一个函数；理解主调函数

和被调用函数参数的传递过程，掌握函数形参传递数组元素的方法；

（8）理解指针的概念，能区别指针变量和变量的指针。理解指针与数组的关系，熟练使用指针指向一维、二维数组，理解指针表达数组元素的几种表现形式。

学习重点和难点：

（1）C51 的数据类型、存储类型、C51 的运算符和表达式及其规则；

（2）表达式语句、复合语句、条件语句、while 循环语句、do…while 循环语句、for 循环语句的语法及常用算法；

（3）数组的定义、数组元素的表示方法、数组初始化方法、字符数组和字符串；

（4）指针的定义格式、指针的赋值、指针的运算，使用指针表示数组的元素；

（5）函数的定义格式、函数说明方法、函数的参数、函数的返回值；函数的调用方式。

3.1 项目 9 了解单片机 C 语言

• 技能目标

掌握任务 9-1 了解 C51 编程结构。

• 知识目标

学习目的：

（1）了解 C51 程序开发概述；

（2）掌握 C51 程序结构；

（3）掌握标识符与关键字。

学习重点和难点：

（1）C51 程序结构；

（2）标识符与关键字。

3.1.1 任务 9-1 了解 C51 编程结构

1. 任务要求

（1）掌握 C51 编程结构；

（2）了解 C51 编程基本部分；

（3）了解 C51 编程书写格式；

（4）掌握程序结构特点。

2. 任务描述

下面通过一个程序简单来了解 C51 编程的结构、特点、基本部分和书写格式。

3. 任务实现——C51 编程结构

```
#include<reg51.h>              //C 语言的预编译处理，包含 51 单片机寄存器定义的头文件
void main(void)          //主函数，第一个 void 表示无需返回值，第二个 void 表示没有参数传递
{                              //每个函数必须以花括号"{"开始
```

```
    P2 = 0x00;                    //P1 = 0000 0000B,即赋值语句
}                                 //每个函数必须以花括号"}"结束,而且花括号必须成对
```

4. 程序结构特点

(1)“文件包含”处理

程序的第一行是一个“文件包含”处理,其含义是指一个文件内容将被另外一个文件全部包含。由于单片机不认识端口“P1”(或某些寄存器的名字,自己可以打开 Keil 的安装目录,在 C51 文件夹找到“INC”子文件夹,打开里面的“reg51.h”看看,也可以自己在里面定义),要想让单片机认识“P1”,就必须给“P1”作一个定义。这种定义已经由 KeilC51 完成,无须用户再定义,但是编程时必须将这种定义“包含”进去,才能使单片机认识“P1”等各种寄存器的名字。

注意:

- 每个变量必须先说明后引用,变量名英文大小写是有差别的。
- 预处理指令是以#号开头的代码行。#号必须是该行除了任何空白字符外的第一个字符。#后是指令关键字,在关键字和#号之间允许存在任意个数的空白字符。整行语句构成了一条预处理指令,该指令将在编译器进行编译之前对源代码做某些转换。

(2) 主函数

main()函数被称为主函数,每一个 C 语言程序必须有并且只能有一个主函数,函数后面一定要有一对花括号“{ }”,程序就写到里面。

花括号必须成对,位置随意,可在紧挨函数名后,也可另起一行,多个花括号可以同行书写,也可逐行书写,为层次分明,增加可读性,同一层的花括号须对齐,采用逐层缩进方式书写。

(3) 语句结束标志

C 语言程序一行可以书写多条语句,但每个语句必须以“;”结尾,一个语句也可以多行书写。

(4) 注释

C 语言程序设计中的注释只是为了提高程序的可读性,在编译时,注释内容不会被执行。C 语言的注释有两种,一种是采用/*……*/表示;另一种采用“//”表示。二者的区别是:前一种可以注释多行内容,后一种只能注释一行内容。

3.1.2　任务 9-2　相关知识

1. C51 程序开发概述

单片机的 C 语言采用 C51 编译器(简称 C51)。由 C51 产生的目标代码短、运行速度高、所需存储空间小、符合 C 语言的 ANSI 标准,生成的代码遵循 Intel 目标文件格式,而且可与 A51 汇编语言或 PL/M51 语言目标代码混合使用。在众多的 C51 编译器中,Keil 公司的 C 语言编译/连接器 Keil μVision 软件最受欢迎。

(1) 采用 C51 的优点

采用 C51 进行单片机应用系统的程序设计,编译器能自动完成变量的存储单元的分

配，编程者可以专注于应用软件的设计，可以对常用的接口芯片编制通用的驱动函数，对常用的功能模块和算法编制相应的函数，可以方便地进行信号处理算法和程序的移植，从而加快单片机应用系统的开发过程。

（2）C51 程序的开发过程

C51 程序的开发过程如图 3-1 所示。

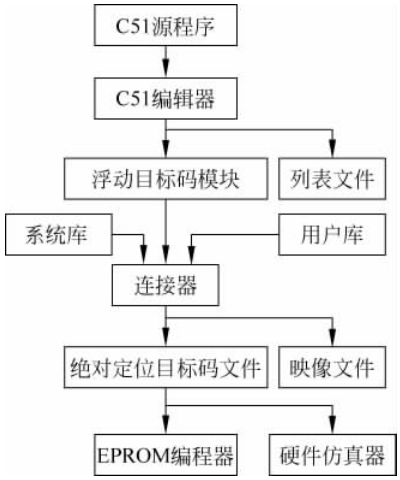

图 3-1　C51 程序开发过程示意图

2. C51 程序结构

与普通的 C 语言程序类似，C51 程序是由若干模块化的函数构成的。函数是 C51 程序的基本模块，常说的“子程序”、“过程”在 C51 中用“函数”这个术语。它们都含有以同样的方法重复地去做某件事情的意思。主程序（main()）可以根据需要用来调用函数。当函数执行完毕时，就发出返回（return）指令，而主程序后面的指令用来恢复主程序流的执行。同一个函数可以在不同的地方被调用，并且函数可以重复使用。

C 语言程序的扩展名为“.c”，如“XM4-1.c”。

C 语言程序的组成结构如下（主函数可以放在功能子函数说明之后的任意位置）。

```
预处理命令   include <>
功能子函数 1   说明
    ⋮
功能子函数 n   说明
功能子函数 1 fun1 ()
    {
        函数体…
    }
    ⋮
功能子函数 n fun ()
    {
        函数体…
    }
main()
```

```
{
    函数体…
}
```

所有函数在定义时都是相互独立的，一个函数中不能再定义其他的函数，即函数不能嵌套定义，但可以互相调用。函数调用的一般原则是：主函数可以调用其他普通函数，普通函数之间也可相互调用，但普通函数不能调用主函数。

一个 C 程序的执行总是从 main()函数开始的，调用其他函数后返回到 main()中，最后在主函数 main()中结束整个 C 程序的运行。

3. 标识符与关键字

(1) 标识符

标识符是一种单词，它用来给变量、函数、符号常量、自定义类型等命名。用标识符给 C 语言程序中各种对象命名时，要用字母、下划线和数字组成的字符序列，并要求首字符是字母或下划线，不能是数字。例如，可以使用 x、y 作为变量的标识符，使用 delay()作为函数的标识符。

字母的大小写是有区别的，如 max 和 MAX 是两个完全不同的标识符。

通常下划线开头的标识符是编译系统专用的，因此在编写 C 语言源程序时一般不使用以下划线开头的标识符，而将下划线用作分段符。C51 编译器规定标识符最长可达 255 个字符，但只有前 32 个字符在编译时有效，因此标识符的长度一般不要超过 32 个字符。

(2) 关键字

关键字是一种已被系统使用过的具有特定含义的标识符。用户不得再用关键字给变量等命名。C 语言的关键字分为如下三类。

① 类型说明符。用来定义变量、函数或其他数据结构的类型，如 unsigned char、long 等。

② 语句定义符。用来标识一个语句功能，如条件判断语句“if”、“while”等。

③ 预处理命令字。表示预处理命令的关键字，如程序开头的“include”。

ANSI C 标准一共规定了 32 个关键字，如表 3-1 所示。

表 3-1　ANSI C 语言的关键字

关键字	用　途	说　明
auto	存储种类说明	用以说明局部变量，缺省值为此
break	程序语句	退出最内层循环
case	程序语句	switch 语句中的选择项
char	数据类型说明	单字节整型数或字符型数据
const	存储种类说明	在程序执行过程中不可更改的常量值
continue	程序语句	转向下一次循环
default	程序语句	switch 语句中的失败选择项
do	程序语句	构成 do…while 循环结构
double	数据类型说明	双精度浮点数
else	程序语句	构成 if…else 选择结构

续表

关键字	用　途	说　明
enum	数据类型说明	枚举类型
extern	存储种类说明	在其他程序模块中说明了的全局变量
float	数据类型说明	单精度浮点数
for	程序语句	构成 for 循环结构
goto	程序语句	构成 goto 转移结构
if	程序语句	构成 if…else 选择结构
int	数据类型说明	基本整型数
long	数据类型说明	长整型数
register	存储种类说明	使用 CPU 内部寄存器的变量
return	程序语句	函数返回
short	数据类型说明	短整型数
signed	数据类型说明	有符号数，二进制数据的最高位为符号位
sizeof	运算符	计算表达式或数据类型的字节数
static	存储种类说明	静态变量
struct	数据类型说明	结构类型数据
switch	程序语句	构成 switch 选择结构
typedef	数据类型说明	重新进行数据类型定义
union	数据类型说明	联合类型数据
unsigned	数据类型说明	无符号数数据
void	数据类型说明	无类型数据
volatile	数据类型说明	该变量在程序执行中可被隐含地改变
while	程序语句	构成 while 和 do…while 循环结构

KeilC51 编译器除了有 ANSI C 标准的 32 个关键字外，还根据 51 单片机的特点扩展了相应的关键字。在 KeilC51 开发环境的文本编辑器中编写 C 程序，系统可以把保留字以不同的颜色显示，缺省颜色为蓝色。表 3-2 为 KeilC51 编译器扩展的关键字。

表 3-2　KeilC51 编译器扩展关键字

关键字	用　途	说　明
bit	位标量声明	声明一个位标量或位类型的函数
sbit	位变量声明	声明一个可位寻址变量
sfr	特殊功能寄存器声明	声明一个特殊功能寄存器(8 位)
sfr16	特殊功能寄存器声明	声明一个 16 位的特殊功能寄存器
data	存储器类型说明	直接寻址的 8051 内部数据存储器
bdata	存储器类型说明	可位寻址的 8051 内部数据存储器
idata	存储器类型说明	简洁寻址的 8051 内部数据存储器
pdata	存储器类型说明	“分页”寻址的 8051 外部数据存储器
xdata	存储器类型说明	8051 外部数据存储器
code	存储器类型说明	8051 程序存储器
interrupt	中断函数声明	定义一个中断函数
reetrant	再入函数声明	定义一个再入函数
using	寄存器组定义	定义 8051 的工作寄存器组

例如：

```
Sfr P1 = 0x90;/ * 定义地址为"0x90"的特殊功能寄存器名字为"P1",对 P1 的操作也就是对地址为
0x90 的寄存器操作. * /
(格式: Sfr 特殊功能寄存器名 = 地址常数.)
Sbit S = P2 ^0;//位定义 S 为 P2.0(P2 口的第 0 位)
(格式: Sbit 位变量名 = 特殊功能寄存器名^位位置.)
```

3.2　项目 10　认识 C51 的数据类型

• 技能目标

掌握任务 10-1　用不同的数据类型控制 P2 口的 8 位 LED 闪烁。

• 知识目标

学习目的：

(1) 掌握 C 语言的数据类型；

(2) 掌握 C51 数据的存储类型；

(3) 掌握 80C51 硬件结构的 C51 定义。

学习重点和难点：

(1) 掌握 C 语言的数据类型；

(2) 掌握 C51 数据的存储类型；

(3) 掌握 80C51 硬件结构的 C51 定义。

3.2.1　任务 10-1　用不同的数据类型控制 P2 口的 8 位 LED 闪烁

1. 任务要求

(1) 了解字符型数据类型的应用；

(2) 了解整型数据类型的应用；

(3) 掌握延时程序的编写；

(4) 掌握主函数调用延时程序的工作原理；

(5) 了解单片机的工作频率。

2. 任务描述

用不同数据类型控制 P2 口的 8 位 LED 闪烁。使用无符号字符型数据和无符号整型数据来设计两个不同的延时时间，来控制 LED0～LED3 和 LED4～LED7 闪烁，可以看出两组灯闪烁时间是不一样的。

3. 任务实现

(1) 程序设计

先建立文件夹“XM10-1”，然后建立“XM10-1”工程项目，最后建立源程序文件“XM10-1.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
/*************************
函数功能：用字符型数据延时一段短时间
*************************/
void delay60(void)
{
    unsigned char i,j;
    for(i=0;i<200;i++)
    for(j=0;j<100;j++)
      ;
}
/*********************
函数功能：用整型数据延时一段长时间
*********************/
void delay150(void)                  //两个 void 意思分别为无需返回值,没有参数传递
{
 unsigned int i;                     //定义无符号整数,最大取值范围为 65 535
for(i=0;i<50000;i++)                 //做 50 000 次空循环
      ;                              //什么也不做,等待一个机器周期
}
/*****************************
函数功能：主函数(C 语言规定必须有一个主函数)
*****************************/
void main(void)
{
 while(1)                            //无限循环
{
    P2=0xf0;                         //P2=1111 0000B,灯 LED0～LED3 亮
    delay60();                       //延时一段短时间
        P2=0xff;                     //P2=1111 1111B, 灯 LED0～LED3 灭
    delay60();                       //延时一段短时间
    P2=0x0f;                         //P2=0000 1111B,灯 LED4～LED7 亮
    delay150();                      //延时一段长时间
        P2=0xff;                     //P2=1111 1111B, 灯 LED0～LED3 灭
    delay150();                      //延时一段长时间
 }
}
```

(2) 用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM10-1.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看出两组灯的闪烁时间是不一样的，效果图如图 3-2 所示。

3.2.2 任务 10-2 相关知识

1. 数据类型

数据是计算机的操作对象。不管使用任何语言，任何算法进行程序设计，最终在计算机

中运行的只有数据流。

数据类型——数据的不同格式叫数据类型。C 语言中常用的数据类型有整型、字符型、实型、指针型和空类型。而根据变量在程序执行中是否发生变化，还可将数据类型分为常量和变量两种。在程序中，常量可以不经说明而直接引用，而变量则必须先定义类型后才能使用。

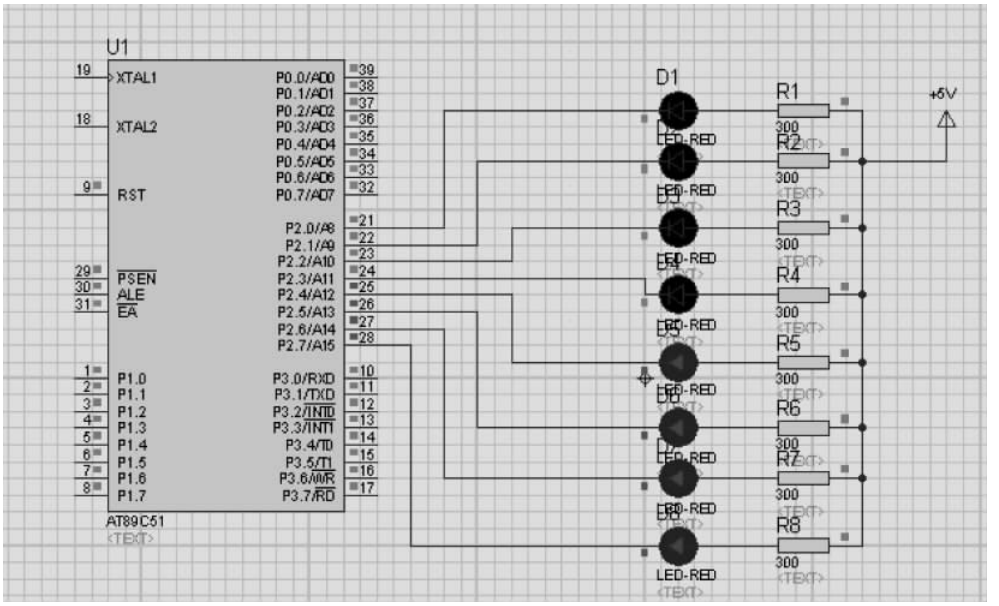

图 3-2　用不同数据类型控制 P2 口的 8 位 LED 闪烁仿真效果图

1）常量与变量

在程序运行过程中，其值不能被改变的量称为常量。常量区分为不同的类型，如 10，0，－4 为整型常量，2.6、－3.15 为实型常量，‘a’，‘d’为字符常量。

C 语言中还有一种符号常量，其定义形式如下。

```
#define 符号常量的标识符 常量
```

例如：

```
#define  price  30
main ()
{
  int a,s;
  a = 10;
 s = a * price;
 printf("total = %d",total);
}
```

程序中用 #define 命令行定义 price 代表常量 30，此后凡在本文件中出现的 price 都代表 30，可以和常量一样进行运算。

使用符号常量的优点是，当程序中有许多地方要用到某个常量，而其值又经常改变时，使用符号常量就可以“一改全改”。

在程序执行中，其值可以改变的量称为变量，变量标识符常用小写字母来表示，变量必须先定义后使用，一般放在程序开头部分。

2）整型数据

整型数据包括整型常量和整型变量。

（1）整型常量

整型常量就是整型常数。在C语言中，整型常量可用以下三种形式表示。

① 十进制整数。用0～9表示，如321，－106等。

② 八进制整数。以0开头的数是八进制数，用0～7表示。如0215表示八进制数215，即等于十进制数141。

③ 十六进制整数。以0x开头的数是十六进制数，用0～9和a～f表示，如0xff，代表十六进制数ff，等于十进制数255。

（2）整型变量

整型数据在内存中是以二进制形式存放的。整型变量可分为基本型和无符号型，基本型类型说明符为int，在内存中占2个字节；无符号型类型说明符为unsigned，同样在内存中占2个字节。

KeilC51软件编译器支持的数据类型如表3-3所示。

表3-3 C51的数据类型

数据类型		长度/位	取值范围
字符型	signed char	8	－128～127
	unsigned char	8	0～255
整型	signed int	16	－32 768～32 767
	unsigned int	16	0～65 535
长整型	signed long	32	－21 474 883 648～21 474 883 647
	unsigned long	32	0～4 294 967 295
浮点型	float	32	±1.754 94e－38～±3.402 823e＋38
位型	bit	1	0，1
	sbit	1	0，1
访问SFR	sfr	8	0～255
	sfr16	16	0～65 535

C规定在程序中所有用到的变量都必须在程序中定义，即“强制类型定义”。整型变量的定义形式。

```
类型说明符 变量标识符1,变量标识符2,…
```

例如：

```
int a,b;                          //指定变量a、b为整型,各变量名之间用逗号相隔
unsigned short c,d;               //指定变量c、d为无符号短整型
long e,f;                         //指定变量e、f为长整型
```

对变量的定义，一般是放在一个函数开头的声明部分（也可以放在函数中某一分程序内，但作用域只限它所在的分程序）。

(3) 实型数据

实型数据有两种表示形式。

① 十进制小数形式。它由数字和小数点组成(注意必须有小数点)。.123、123.、123.0、0.0都是十进制小数形式。

② 指数形式。如1.23e5或123e3都代表123×10^3。但注意字母e之前必须有数字,且e后面的指数必须为整数,如e3、2.1e3.5、.e3、e等都不是合法的指数形式。

(4) 字符型数据

字符型数据包括字符型常量和字符型变量。

① 字符型常量

C的字符常量是用单引号(即撇号)括起来的一个字符,如'a','x','d','?',''等都是字符常量。字符型常量常用作显示说明。

② 字符型变量

字符型变量用来存放字符常量,请注意只能放一个字符,其说明符是"char",定义形式为

```
char  x,y;                    //它表示x和y为字符型变量,在内存中各占一个字节
```

③ 字符串常量

字符常量是由一对单引号括起来的单个字符。C语言除了允许使用字符常量外,还允许使用字符串常量。字符串常量是一对双引号括起来的字符序列。

例如:

```
"Welcome to China.";          //是字符串常量
'a';                          //是字符常量
```

(5) 指针型数据

出于对变量灵活使用的需要,有时在程序中围绕变量的地址展开操作,这就引入"指针"的概念。变量的地址称为变量的指针,指针的引入把地址形象化了,地址是找变量值的索引或指南,就像一根"指针"一样指向变量值所在的存储单元,因此指针即是地址,是记录变量存储单元位置的正整数。

(6) 位类型数据

位类型数据是C51编译器的一种扩充数据类型,利用它可以定义一个位变量,但不能定义位指针,也不能定义位数组。该类型数据取值为"0"或"1"。

(7) 空类型数据

C语言经常使用函数,当函数被调用完后,无需返回一个函数值,这个函数值称为空类型数据,例如以下程序。

```
/**********************
    函数功能:用整型数据延时一段长时间
**********************/
void delay(void)              //两个void意思分别为无需返回值,没有参数传递
{
    unsigned int i;           //定义无符号整数,最大取值范围为65 535
    for(i=0;i<50 000;i++)     //做50 000次空循环
```

```
        ;                    //什么也不做,等待一个机器周期
}
```

用“void”说明该函数为“空类型”，即无返回值。void 的字面意思是“无类型”。

(8) 变量赋值

程序中常需要对一些变量预先赋值。C 语言允许在定义变量的同时给变量赋值，如下所示。

```
int a = 3;                      /* 指定 a 为整型变量,值为 3 */
float f = 3.56;                 /* 指定 f 为实型变量,值为 3.56 */
char c = 'a';                   /* 指定 c 为字符变量,值为'a' */
```

也可以使被定义的变量的一部分赋值，如：

```
int a,b,c = 5;
```

表示指定 a、b、c 为整型变量，只对 c 赋值，c 的值为 5。

2. C51 数据的存储类型

C51 是面向 80C51 系列单片机的程序设计语言，应用程序中使用的任何数据(变量和常数)必须以一定的存储类型定位于单片机相应的存储区域中。C51 编译器支持的存储类型如表 3-4 所示。

表 3-4 C51 的存储类型与 8051 存储空间的对应关系

存储器类型	长度/位	对应单片机存储器
bdata	1	片内 RAM,位寻址区,共 128 位(也能字节访问)
data	8	片内 RAM,直接址区,共 128B
idata	8	片内 RAM,间接址区,共 256B
pdata	8	片外 RAM,分页间址,共 256B
xdata	16	片外 RAM,间接寻址,共 64KB
code	16	ROM 区域,间接寻址,共 64KB

对于 80C51 系列单片机来说，访问片内的 RAM 比访问片外的 RAM 的速度要快得多，所以对于经常使用的变量应该置于片内 RAM，即用 bdata、data、idata 来定义；对于不常使用的变量或规模较大的变量应该置于片外 RAM 中，即用 pdata、xdata 来定义。

例如：

```
bit bdata my_flag;                  /* item1 */
char data var0;                     /* item2 */
float idata x,y,z;                  /* item3 */
unsigned int pdata temp;            /* item4 */
unsigned char xdata array[3][4];    /* item5 */
```

item1。位变量 my_flag 被定义为 bdata 存储类型，C51 编译器将把该变量定义在 80C51 片内数据存储区(RAM)中的位寻址区(地址为：20H～2FH)。

item2。字符变量 var0 被定义为 data 存储类型，C51 编译器将把该变量定位在 80C51 片内数据存储区中(地址为：00H～FFH)。

item3。浮点变量 x、y、z 被定义为 idata 存储类型，C51 编译器将把该变量定位在 80C51 片内数据区，并只能用间接寻址的方式进行访问。

item4。无符号整型变量 temp 被定义为 pdata 存储类型，C51 译器将把该变量定位在 80C51 片外数据存储区（片外 RAM）。

item5。无符号字符二维数组 unsigned char array[3][4]被定义为 xdata 存储类型，C51 编译器将其定位在片外数据存储区（片外 RAM），并占据 3×4＝12 字节存储空间，用于存放该数组变量。

如果用户不对变量的存储类型进行定义，C51 的编译器采用默认的存储类型。默认的存储类型由编译命令中存储模式指令限制。C51 支持的存储模式如表 3-5 所示。

表 3-5　C51 存储模式

存储模式	默认存储类型	特　　点
Small	data	直接访问片内 RAM；堆栈在片内 RAM 中
Compact	pdata	用 R0 和 R1 间址片外分页 RAM；堆栈在片内 RAM 中
Large	xdata	用 DPTR 间址片外 RAM，代码长，效率低

例如：

```
char var ;                    /* 在 small 模式中，var 定位 data 存储区 */
                              /* 在 compact 模式中，var 定位 pdata 存储区 */
                              /* 在 large 模式中，var 定位 xdata 存储区 */
```

在 Keil μVision 4 平台下，设置存储模式的界面如图 3-3 所示，步骤为：工程建立好后，使用菜单 Project→Options for Target 'Target 1'，或单击 快捷图标即出现如图 3-3 所示的工程对话框，单击 Target 标签，其中的 Memory Model 用于设置 RAM 的使用情况，有三个选项。Small 是所有的变量都在单片机的内部 RAM 中；Compact 变量存储在外部 RAM 里，使用 8 位间接寻址；Large 变量存储在外部 RAM 中，使用 16 位间接寻址，可以使用全部外部的扩展 RAM。

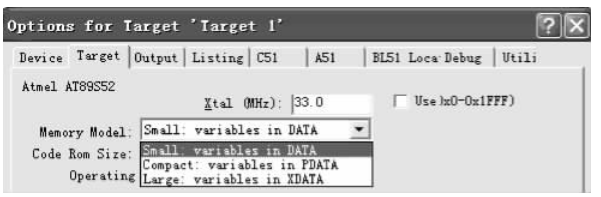

图 3-3　KeilC51 的存储模式界面

3. 80C51 硬件结构的 C51 定义

C51 是适合于 80C51 单片机的 C 语言。它对标准 C 语言（ANSI C）进行扩展，从而具有对 80C51 单片机硬件结构的良好支持与操作能力。

（1）特殊功能寄存器的定义

80C51 单片机内部 RAM 的 80H～FFH 区域有 21 个特殊功能寄存器，为了对它们能够直接访问，C51 编译器利用扩充的关键字 SFR 和 SFR16 对这些特殊功能寄存器进行定义。

SFR 的定义方法是：sfr 特殊功能寄存器名＝地址常数

例如：

```
sfr P0 =  0x80;                    /* 定义 P0 口,地址为 0x80 */
sfr TMOD = 0x89;                   /* 定时/计数器方式控制寄存器地址 89H */
```

注意：关键字 sfr 后面必须跟一个标识符作为特殊功能寄存器名称，名称可以任意选取，但要符合人们的一般习惯。等号后面必须是常数，不允许有带运算符的表达式，常数的地址范围与具体的单片机型号相对应，通常的 80C51 单片机为 0x80～0xFF。

(2) 特殊功能寄存器中特定位的定义

在 C51 中可以利用关键字 sbit 定义可独立寻址访问的位变量，如定义 80C51 单片机 SFR 中的一些特定位，定义的方法有三种。

① sbit 位变量名＝特殊功能寄存器名^位的位置(0～7)

例如：

```
sfr PSW  = 0xD0;                   /* 定义 PSW 寄存器地址为 0xd0h */
sbit OV  =  PSW^2;                 /* 定义 OV 位为 PSW.2,地址为 0xd2 */
sbit CY  =  PSW^7;                 /* 定义 Cy 位为 PSW.7,地址为 0xd7 */
```

② sbit 位变量名＝字节地址^位的位置

例如：

```
sbit OV  = 0xd0^2;                 /* 定义 OV 位的地址为 0xd2 */
sbit CF  = 0xd0^7;                 /* 定义 CF 位的地址为 0xd7 */
```

注意：字节地址作为基地址，必须位于 0x80～0xff 之间。

③ sbit 位变量名＝位地址

例如：

```
sbit OV  = 0xd2;                   /* 定义 OV 位的地址为 0xd2 */
sbit CF  = 0xd7;                   /* 定义 CF 位的地址为 0xd7 */
```

注意：位地址必须位于 0x80～0xFF 之间。

(3) 8051 并行接口及其 C51 定义

① 对于 8051 片内 I/O 口用关键字 sfr 来定义，例如

```
sfr P0 = 0x80;                     /* 定义 P0 口,地址为 80h */
sfr P1 = 0x90;                     /* 定义 P1 口,地址为 90h  */
```

② 对于片外扩展 I/O 口，则根据其硬件译码地址，将其视为片外数据存储器的一个单元，使用 define 语句进行定义，例如

```
#include <absacc.h>
#define PORTA XBYTE [0x78f0];    /* 将 PORTA 定义为外部口,地址为 78f0,长度为 8 位 */
```

一旦在头文件或程序中对这些片内外的 I/O 口进行定义以后，在程序中就可以自由使用这些口了。定义口地址的目的是为了便于 C51 编译器按 8051 实际硬件结构建立 I/O 口变量名与其实际地址的联系，以便使程序员能用软件模拟 8051 硬件操作。

(4) 位变量(bit)及其定义

C51 编译器支持 bit 数据类型。

① 位变量的 C51 定义语法及语义如下。

```
bit dir_bit;                    /* 将 dir_bit 定义为位变量 */
bit lock_bit;                   /* 将 lock_bit 定义为位变量 */
```

② 函数可包含类型为 bit 的参数,也可以将其作为返回值。

```
bit func(bit b0,bit b1)
    {/* … */}
return (b1);
```

③ 对位定义的限制。位变量不能定义成一个指针,如 bit * bit_ptr 是非法的。不存在位数组,如不能定义 bit arr[]。

在位定义中允许定义存储类型,位变量都放在一个段位中,此段总位于 8051 片内 RAM 中,因此存储类型限制为 data 或 idata。如果将位变量的存储类型定义成其他类型,编译时将出错。

④ 可位寻址对象。可位寻址的对象是指可以字节寻址或位寻址的对象,该对象位于 8051 片内 RAM 可位寻址 RAM 区中,C51 编译器允许数据类型为 idata 的对象放入 8051 片内可位寻址的区中。先定义变量的数据类型和存储类型。

```
bdata int ibase;                /* 定义 ibase 为 bdata 整型变量 */
bdata char bary[4];             /* 定义 bary[4]为 bdata 字符型数组 */
```

然后可使用“sbit”定义可独立寻址访问的对象位,即:

```
sbit mybit0 = ibase^0;          /* mybit0 定义为 ibase 的第 0 位 */
sbit mybit15 = ibase^15;        /* mybit15 定义为 ibase 的第 15 位 */
sbit ary01 = bary[0]^1;         /* ary01 定义为 bary[0]的第 1 位 */
sbit ary25 = bary[2]^5;         /* ary25 定义为 bary[2]的第 5 位 */
```

3.3　项目 11　认识 C51 的运算符

• 技能目标

(1) 掌握任务 11-1　分别用 P2、P3 口显示“加减”运算结果;

(2) 掌握任务 11-2　用 P1 口显示逻辑“与或”运算结果;

(3) 掌握任务 11-3　分别用 P2、P3 口显示位“与或”运算结果;

(4) 掌握任务 11-4　用 P1 口显示“左右移”运算结果。

• 知识目标

学习目的:

(1) 掌握算术运算符、关系表达式及优先级;

(2) 掌握关系运算符、关系表达式及优先级;

(3) 掌握逻辑运算符和逻辑表达式及优先级;

（4）掌握 C51 位操作及其表达式。

学习重点和难点：

（1）算术运算符、关系表达式及优先级；

（2）关系运算符、关系表达式及优先级；

（3）逻辑运算符和逻辑表达式及优先级；

（4）C51 位操作及其表达式。

3.3.1 任务 11-1 分别用 P2、P3 口显示“加减”运算结果

1. 任务要求

（1）了解“加”运算及编程；

（2）了解“减”运算及编程；

（3）掌握十进制数、十六进制数、二进制数转换；

（4）掌握无符号字符型定义即“unsigned char a,b;”；

（5）掌握无限循环编程。

2. 任务描述

分别用 P2、P3 口显示“加减”运算结果。把两个数进行“加减”运算，即设“52＋48”和“52－48”，把“加”运算运算结果送 P2 口显示出来，把“减”运算结果送 P3 口显示出来。

3. 任务实现

（1）分析

设置两个无符号字符型变量 a 和 b，并分别赋值十进制数 52 和 48，然后进行 $a+b$ 和 $a-b$ 运算，并把运算结果分别送 P2、P3 口显示。

（2）程序设计

先建立文件夹“XM11-1”，然后建立“XM11-1”工程项目，最后建立源程序文件“XM11-1.c”，输入如下源程序。

```
    #include<reg51.h>          //包含单片机寄存器的头文件
/*****************************
函数功能：主函数(C语言规定必须有一个主函数)
*****************************/
void main(void)
{
  unsigned char a,b;           //定义无符号字符型,最大取值范围为255
  a=52;                        //a赋值为52
  b=48;                        //b赋值为48
  P2=a+b;/*P2=52+48=100=64H=01100100B,结果为P2.7、P2.4、P2.3、P2.1、P2.0接的LED灯亮*/
  P3=a-b;/*P3=52-48=4=00000100B,结果为P3.7、P3.6、P3.5、P3.4、P3.3、P3.1、P3.0接的
LED灯亮*/
  While(1);                    //无限循环
    ;                          //空操作
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM11-1.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看到分别用 P2、P3 口显示“加减”运算结果，效果图如图 3-4 所示。

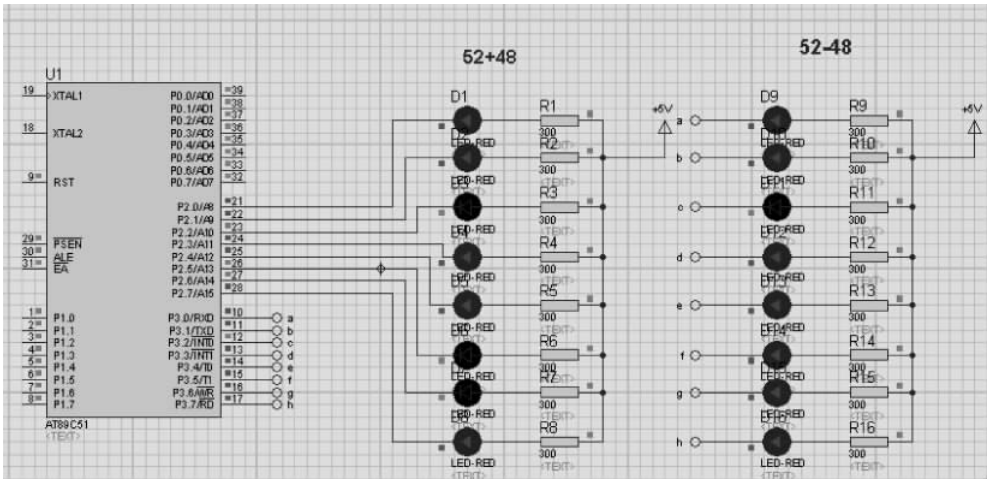

图 3-4　分别用 P2、P3 口显示“加减”运算结果

3.3.2　任务 11-2　用 P1 口显示逻辑“与或”运算结果

1. 任务要求

(1) 掌握“与”运算及编程；

(2) 掌握“或”运算及编程；

(3) 掌握延时程序编程；

(4) 掌握无限循环编程。

2. 任务描述

用 P1 口显示逻辑“与或”运算结果。把“(6>0x0f)&&(8<0xa)”和“(6>0x0f)||(8<0xa)”运算结果送 P1 口显示出来。

3. 任务实现

(1) 分析

把“(6>0x0f)&&(8<0xa)”进行“与”运算，即“(6>0x0f)&&(8<0xa)”=0&&1=0，结果送 P1 口使得 8 只 LED 全亮，然后调延时；再把“(6>0x0f)||(8<0xa)”进行“或”运算，即“(6>0x0f)||(8<0xa)”=0||1=1，结果送 P1 口使得 8 只 LED 全亮灭，然后调延时。

(2) 程序设计

先建立文件夹“XM11-2”，然后建立“XM11-2”工程项目，最后建立源程序文件“XM11-2.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
/**********************
    函数功能：用整型数据延时时间
**********************/
void delay(void)                     //两个 void 意思分别为无需返回值，没有参数传递
{
    unsigned int i;                  //定义无符号整数，最大取值范围为 65 535
    for(i=0;i<50000;i++)             //做 50 000 次空循环
        ;                            //什么也不做，等待一个机器周期
}
/*****************************
函数功能：主函数
*****************************/
void main(void)
{
  while(1)                           //无限循环
    {
        P1=(6>0x0f)&&(8<0xa);        //运算结果送 P1=0000 0000B，灯 LED0～LED7 亮
        int_delay();                 //延时
        P1=(6>0x0f)||(8<0xa);        //运算结果送 P1=1111 1111B，灯 LED0～LED7 灭
        int_delay();                 //延时
    }
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM11-2.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看到用 P1 口显示逻辑“与或”运算结果，效果图如图 3-5 所示。

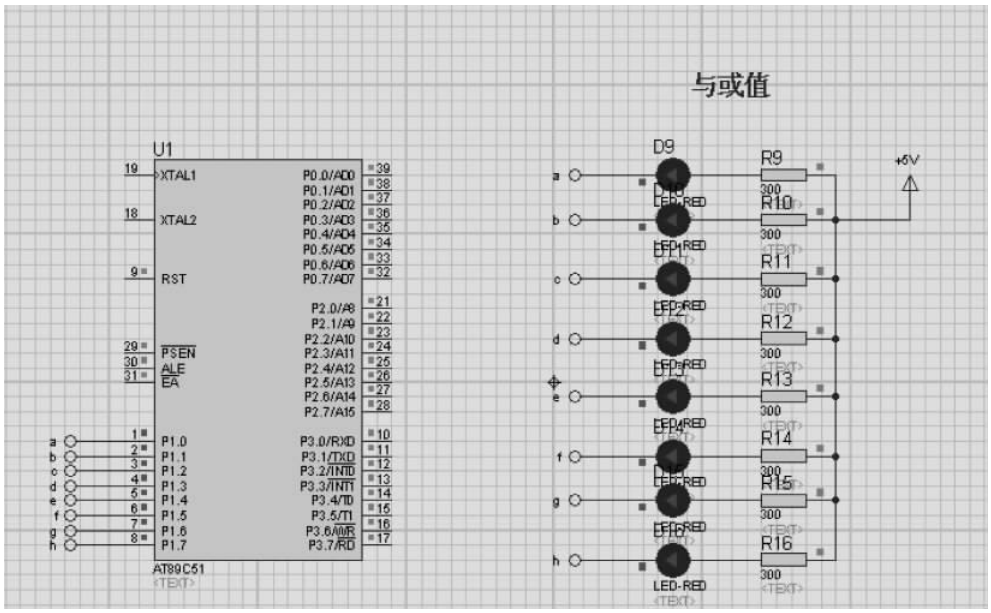

图 3-5 用 P1 口显示逻辑“与或”运算结果

3.3.3　任务 11-3　分别用 P2、P3 口显示位“与或”运算结果

1. 任务要求

(1) 掌握位“与”运算及编程；
(2) 掌握位“或”运算及编程；
(3) 掌握十六进制数、二进制数转换；
(4) 掌握无限循环编程。

2. 任务描述

分别用 P2、P3 口显示位“与或”运算结果。把两个数十六进制数进行位“与或”运算，即设“0x52&0x48”和“0x52|0x48”，把位“与”运算结果送 P2 口显示出来，把位“或”运算结果送 P3 口显示出来。

3. 任务实现

(1) 分析

设两个十六进制数进行位“与”运算即“0x52&0x48 = 01010010&01001000 = 01000000”，把运算结果送 P2 口显示出来，把数“0x52 | 0x48 = 01010010 | 01001000 = 01011010”运算结果送 P3 口显示出来。

(2) 程序设计

先建立文件夹“XM11-3”，然后建立“XM11-3”工程项目，最后建立源程序文件“XM11-3.c”，输入如下源程序。

```
    #include<reg51.h>          //包含单片机寄存器的头文件
/*****************************
函数功能：主函数(C语言规定必须有一个主函数)
*****************************/
void main(void)
{
  P2 = 0x52&0x48; /* P2 = 01010010&01001000 = 01000000B,结果为 P2.7、P2.5、P2.4、P2.3、P2.2、P2.1、
P2.0 接的 LED 亮 */
  P3 = 0x52|0x48; /* P3 = 01010010&01001000 = 01011010B,结果为 P3.7、P3.5、P3.2、P3.0 接的 LED
亮 */
  While(1);                    //无限循环
    ;                          //空操作
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM11-3.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看到分别用 P2、P3 口显示位“与或”运算结果，效果图如图 3-6 所示。

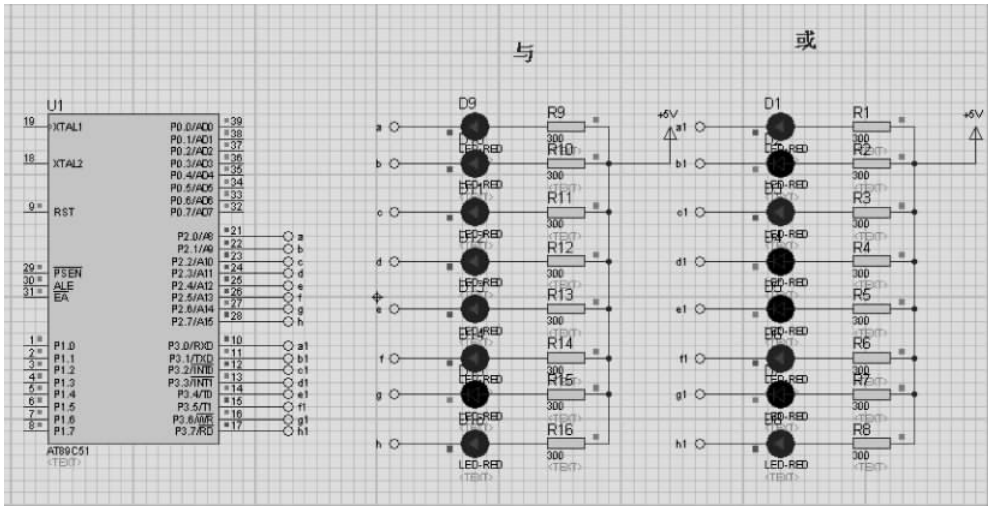

图 3-6　分别用 P2、P3 口显示位“与或”运算结果

3.3.4　任务 11-4　用 P1 口显示“左右移”运算结果

1. 任务要求

（1）掌握“右移”运算及编程；
（2）掌握“左移”运算及编程；
（3）掌握二进移位；
（4）掌握无限循环编程。

2. 任务描述

用 P1 口显示“左右移”运算结果。把数“0xaa”进行左移 1 位运算，即“0xaa＜＜1”，把运算结果送 P1 口显示出来，调延时，P1 再左移 1 位运算结果送显示；然后再把刚刚左移 2 位的数进行右移 2 位运算，分别把运算结果送 P1 口显示出来。

3. 任务实现

（1）分析

设一个十六进制数“0xaa”，展开为二进制数为 10101010B，进行左移 1 位运算“10101010B→01010100”，规则为高位丢掉，低位添 0，把运算结果送 P1 口显示，再进行左移 1 位运算“01010100→**10101000**”，把运算结果送 P1 口显示即 LED6、LED4、LED2、LED1、LED0 亮，其他 LED7、LED5、LED3 灭。然后再把这个数据进行右移 2 位运算，即“10101000→01010100→**00101010**”，再把运算结果送 P1 口显示即 LED7、LED6、LED4、LED2、LED0 亮，其他 LED5、LED3、LED1 灭。

（2）程序设计

先建立文件夹“XM11-4”，然后建立“XM11-4”工程项目，最后建立源程序文件“XM11-4.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
/**********************
    函数功能：用整型数据延时时间
**********************/
void delay(void)                     //两个 void 意思分别为无需返回值,没有参数传递
{
    unsigned int i;                  //定义无符号整数,最大取值范围为 65 535
    for(i=0;i<50000;i++)             //做 50 000 次空循环
            ;                        //什么也不做,等待一个机器周期
}
/*****************************
函数功能：主函数
*****************************/
void main(void)
{
  while(1)                           //无限循环
    {
      P1=0xaa<<1; /*运算结果送 P1=01010100B,LED7、LED5、LED3、LED1、LED0 亮,其他 LED6、
LED4、LED2 灭*/
      int_delay();                   //延时
      P1=P1<<1; /*运算结果送 P1=10101000B,LED6、LED4、LED2、LED1、LED0 亮,其他 LED7、LED5、
LED3 灭*/
      int_delay();                   //延时
      P1=P1>>1; /*运算结果送 P1=01010100B,再把运算结果送 P1 口显示即 LED7、LED5、LED3、
LED1、LED0 亮,其他 LED6、LED4、LED2 灭*/
      int_delay();                   //延时
      P1=P1>>1; /*运算结果送 P1=00101010B,再把运算结果送 P1 口显示即 LED7、LED6、LED4、
LED2、LED0 亮,其他 LED5、LED3、LED1 灭*/
      int_delay();                   //延时
    }
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM11-4.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看到用 P1 口显示“左右移”运算结果，效果图如图 3-7 所示。

3.3.5　任务 11-5　相关知识

1. 算术运算符、关系表达式及优先级

(1) 基本算术运算符

+ 加法运算符，或正值符号；

− 减法运算符，或负值符号；

* 乘法运算符；

/ 除法运算符；

% 模(求余)运算符，例 11%3=2，结果是 11 除以 3 所得余数为 2。

在上述运算符中，加、减和乘法符合一般的算术运算规则。除法运算时，如果是两个整

数相除，其结果为整数；如果是两个浮点数相除，其结果为浮点数。而对于取余运算，则要求两个运算对象均为整型数据。

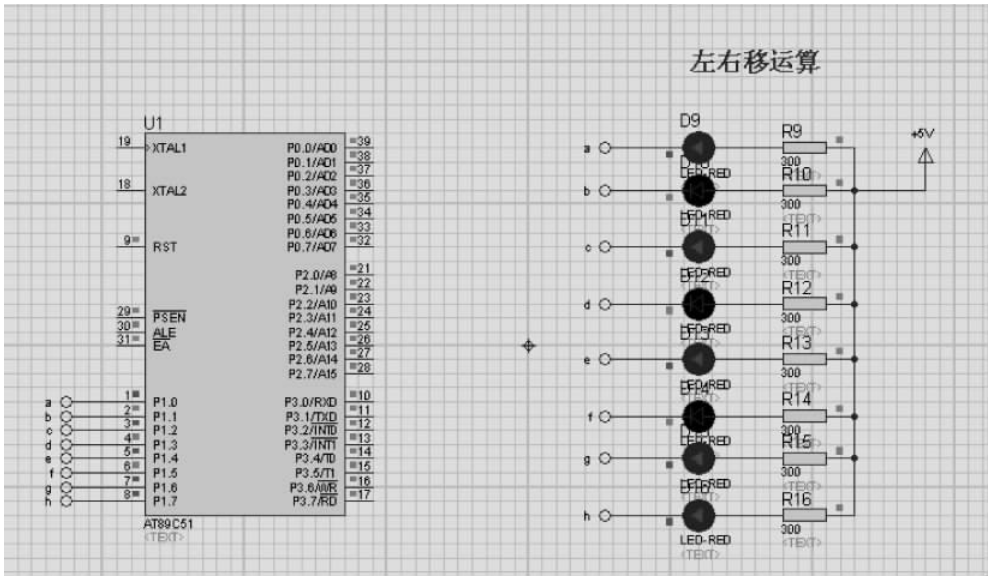

图 3-7　用 P1 口显示“左右移”运算结果

C 语言规定了算术运算符的优先级和结合性。

优先级——指当运算对象两侧都有运算符时，执行运算的先后次序。按运算符优先级别的高低顺序执行运算。

结合性——指当一个运算对象两侧的运算符优先级别相同时的运算顺序。

算术运算符中取负运算的优先级最高，其次是乘法、除法和取余，加法和减法的优先级最低。也可以根据需要，在算术表达式采用括号来改变优先级的顺序。

例如，$a+b/c$，该表达式中，除号优先级高于加号，故先运算 b/c 所得的结果，之后再与 a 相加。

$(a+b)*(c-d)-e$，该表达式中，括号优先级最高，其次是“*”，最后是减号。故先运算$(a+b)$和$(c-d)$，然后再将二者结果相乘，最后减去 e。

(2) 自增减运算符

自增减运算符的作用是使变量值自动加 1 或减 1。

++　自增运算符；

--　自减运算符。

++和--运算符只能用于变量，不能用于常量和表达式，如++(a+1)是错误的。

例如，++i、--i，在使用 i 之前，先使 i 值加(减)1。

i++、i--，在使用 i 之后，再使 i 值加(减)1。

粗略地看，++i 和 i++的作用都相当于 $i=i+1$，但++i 和 i++的不同之处在于++i 先执行 $i=i+1$，再使用 i 的值；而 i++则是先使用 i 的值，再执行 $i=i+1$。

例如，若 i 的值原来为 5，则

$j=++i$，j 的值为 6，i 的值也为 6；

$j=i++$，j 的值为 5，i 的值为 6。

(3) 类型转换

运算符两侧的数据类型不同时，要转换成同种类型。转换的方法有两种，一是自动转换，编译系统在编译时自动进行类型转换，顺序是：bit→char→int→long→float，signed→unsigned。二是强制类型转换，是通过类型转换运算来实现的。

其一般形式为：(类型说明符)(表达式)

功能是：把表达式的运算结果强制转换成类型说明符所表示的类型。

例如，(double)a　将 a 强制转换成 double 类型；

(int)(x+y)　将 $x+y$ 值强制转换成 int 类型；

(float)(5%3)　将模运算 5%3 的值强制转换成 float 类型。

2. 关系运算符、关系表达式及优先级

(1) C51 提供的 6 种关系运算符

< 小于；

<= 小于等于；

> 大于；

>= 大于等于

== 测试等于；

==!= 测试不等于。

(2) 关系运算符的优先级

① <、>、<=、>=的优先级相同，两种==、==!=相同；前 4 种优先级高于后两种。

② 关系运算符的优先级低于算术运算符。

③ 关系运算符的优先级高于赋值运算符。

例如，$c>a+b$　等效于 $c>(a+b)$；$a>b!=c$　等效于$(a>b)!=c$

$a=b>c$　等效于 $a=(b>c)$

(3) 关系运算符的结合性为左结合

例如，$a=4$，$b=3$，$c=1$，则 $f=a>b>c$，则 $a>b$ 的值为 1，$1>c$ 的值为 0，故 $f=0$。

(4) 关系表达式

用关系运算符将两个表达式(可以是算术表达式、关系表达式、逻辑表达式、字符表达式)连接起来的式子。

(5) 关系表达式的结果

真和假。C51 中用 0 表示假，1 表示真。

3. 逻辑运算符和逻辑表达式及优先级

(1) C51 提供了三种逻辑运算符

! 逻辑“非”(NOT)；

&& 逻辑“与”(AND)；

|| 逻辑“或”(OR)。

“&&”和“||”是双目运算符，要求有两个运算对象；而“!”是单目运算符，只要求有一个运算对象。

(2) 逻辑运算符的优先级

在逻辑运算中，逻辑非的优先级最高，且高于算术运算符；逻辑或的优先级最低，低于关系运算符，但高于赋值运算符。

(3) 逻辑表达式

用逻辑运算符将关系表达式或逻辑量连接起来的式子称为逻辑表达式。其值应为逻辑量真和假，逻辑表达式和关系表达式的值相同，以 0 代表假，1 代表真。

(4) 逻辑运算符的结合性为从左到右

例如，如 $a=4$，$b=5$ 则

!a 为假。因为 $a=4$(非 0)为真，所以!a 为假(0)。

$a||b$ 为真。因为 a，b 为真，所以两者相或为真。

$a\&\&b$ 为真。

!$a\&\&b$ 为假(0)。!优先级高于 &&，先执行!a 为假(0)，0&&b=0，结果为假。

4. C51 位操作及其表达式

C51 提供了 6 种位运算符。

& 位与；

| 位或；

^ 位异或；

～ 位取反；

<< 左移；

>> 右移。

除按位取反运算符“～”以外，以上位操作运算符都是双目运算符，即要求运算符两侧各有一个运算对象。

(1) “按位与”运算符“&”

运算规则是：参与运算的两个运算对象，若两者相应的位都为 1，则该位结果为 1，否则为 0，即 0&0=0、0&1=0、1&0=0、1&1=0。

例如，a=45H=0100 0101B，b=0deH=1101 1110B，则表达式 $c=a\&b$=44H。

按位与的主要用途如下。

- 清零。用 0 去和需要清零的位按位与运算。
- 取指定位。

(2) “按位或”运算符“|”

运算规则是：参与运算的两个运算对象，若两者相应的位中有一位为 1，则该位结果为 1，否则为 0，即 0|0=0、0|1=1、1|0=1、1|1=1。

例如，a=30H=00110000B，b=0fH=00001111B，则表达式 $c=a|b$=3fH。

按位或的主要用途是将一个数的某些位置 1，则需要将这些位和 1 按位或，其余的位和 0 进行按位或运算则不变。

(3)“异或”运算符“^”

运算规则是：参与运算的两个运算对象，若两者相应的位相同，则结果为0；若两侧相应的位相异，结果为1，即0^0=0、0^1=1、1^0=1、1^1=0。

例如，a=0a5H，b=3dH，则表达式 $c=a\ \hat{}\ b$=98H。

按位异或的主要用途如下。

- 使特定位翻转(0变1，1变0)。需要翻转的位和1按位异或运算，不需要翻转的位和0按位异或运算。原数和自身按位异或后得0。
- 不用临时变量而交换两数的值。

(4)“位取反”运算符“～”

“～”是一个单目运算符，用来对一个二进制数按位取反，即0变1，1变0。

(5) 位左移和位右移运算符“<<”，“>>”

位左移、位右移运算符“<<”和“>>”，用来将一个二进制位的全部左移或右移若干位；移位后，空白位补零，而溢出的位舍弃。

例如，a=15H，则 $a=a<<2$ =54H；$a=a>>2$=05H。

(6) 赋值和复合赋值运算符

符号“=”称为赋值运算符，其作用是将一个数据的值赋予一个变量。赋值表达式的值就是被赋值变量的值。

在赋值运算符的前面加上其他运算符就可以构成复合赋值运算符。在C51中共有10种复合运算符，这10种赋值运算符均为双目运算符，即

+=，-=，*=，/=，%=，<<=，>>=，&=，|=，^=，～=。

采用这种复合赋值运算的目的，是为了简化程序，提高C程序的编译效率，如

$a+=b$ 相当于 $a=a+b$　　$a\%=b$ 相当于 $a=a\%b$

$a-=b$ 相当于 $a=a-b$　　$a<<=3$ 相当于 $a=a<<3$

$a*=b$ 相当于 $a=a*b$　　$a>>=2$ 相当于 $a=a>>2$

$a/=b$ 相当于 $a=a/b$　　…

(7) 其他运算符(共有10个)

[]是数组的下标。

()是括号。

. 是结构/联合变量指针成员。

& 是取内容。

? 是三目运算符。

，是逗号运算符。

sizeof 是 sizeof 运算符用于在程序中测试某一数据类型占用多少字节。

3.4　项目12　认识C51流程控制语句

- **技能目标**

(1) 掌握任务12-1　用按键S控制P1口8只LED的显示状态；

(2) 掌握任务12-2　用for语句实现蜂鸣器发出1kHz音频；

（3）掌握任务 12-3　用 while 语句实现 P1 口 8 只 LED 的显示状态；

（4）掌握任务 12-4　用 do…while 语句实现 P1 口 8 只 LED 的显示状态。

• **知识目标**

学习目的：

（1）掌握 C51 的顺序结构；

（2）掌握 C51 的选择结构；

（3）掌握 C51 的循环结构。

学习重点和难点：

（1）C51 的选择结构；

（2）C51 的循环结构。

3.4.1　任务 12-1　用按键 S 控制 P1 口 8 只 LED 的显示状态

1. 任务要求

（1）掌握“if”语句功能及编程；

（2）掌握“switch”语句功能及编程；

（3）掌握“while”语句功能及编程；

（4）掌握延时程序的编写。

2. 任务描述

用按键 S 控制 P1 口 8 只 LED 的显示状态。P3.0 接一个按键 S，P1 口接 8 只 LED。设计一个程序实现以下功能。S 按下第一次，LED1 发光；S 按下第二次，LED1、LED2 发光；S 按下第三次，LED1、LED2、LED3 发光；……；S 按下第八次，LED1～LED8 都发光；S 按下第九次，LED1 发光；S 按下第十次，LED1、LED2 发光……依次循环。

3. 任务实现

（1）分析

先设置一个变量 i，当 $i=1$ 时，LED1 发光（被点亮）；当 $i=2$ 时，LED1、LED2 发光；当 $i=3$ 时，LED1、LED2、LED3 发光；……；当 $i=8$ 时，LED1～LED8 都发光。由“switch”语句根据 i 的值来实现 LED 发光。

i 值的改变可以通过按键 S 来控制，每按下 S 按键一次，i 自增 1，当增加到 9 时，将其值重新置为 1。

（2）程序设计

先建立文件夹“XM12-1”，然后建立“XM12-1”工程项目，最后建立源程序文件“XM12-1.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
sbit S=P3^0;                         //定义按键 S 接入 P3.0 引脚
/**************************
函数功能:延时约 30ms
**************************/
```

```
void delay(void)
{
    unsigned char i,j;
      for(i = 0;i<100;i++)
        for(j = 0;j<100;j++)
          ;
}
/*****************************
函数功能:主函数
*****************************/
void main(void)
{
 unsigned char i ;
i = 0 ;                          //i 初始化
while(1)                         //无限循环
{
if(S == 0)//判断 S 按键是否被按下,如果 S = 0 则被按下
{
delay();                         //30ms 延时,消除键盘抖动
if(S == 0)                       //再判断 S 按键是否被按下,如果 S = 0 则确定被按下
i++;                             //i 自增 1
if(i == 9)                       //如果 i = 9,将其值重新置为 1
i = 1 ;
}
switch(i)                        //使用多分支语句
  {
 case 1:P1 = 0xfe;               //LED1 发光
        break;                   //退出 switch 语句
case 2:P1 = 0xfc;                // LED1、LED2 发光
        break;                   //退出 switch 语句
case 3:P1 = 0xf8;                // LED1、LED2、LED3 发光
        break;                   //退出 switch 语句
case 4:P1 = 0xf0;                // LED1、LED2、LED3、LED4 发光
        break;                   //退出 switch 语句
case 5:P1 = 0xe0;                // LED1、LED2、LED3、LED4、LED5 发光
        break;                   //退出 switch 语句
case 6:P1 = 0xc0;                // LED1、LED2、LED3、LED4、LED5、LED6 发光
        break;                   //退出 switch 语句
case 7:P1 = 0x80;                // LED1、LED2、LED3、LED4、LED5、LED6、LED7 发光
        break;                   //退出 switch 语句
case 8:P1 = 0x00;                // LED1～LED8 发光
        break;                   //退出 switch 语句
defauit:                         //默认值,关闭所有 LED
      P1 = 0xff;
}
}
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后,在 Proteus ISIS 编辑环境中绘制仿真电路图,将编译好的“XM12-1.hex”文件加载到 AT89C51 里,然后启动仿真,就可以看到用按键 S 控制 P1 口 8 只 LED 的显示状态,效果图如图 3-8 所示。

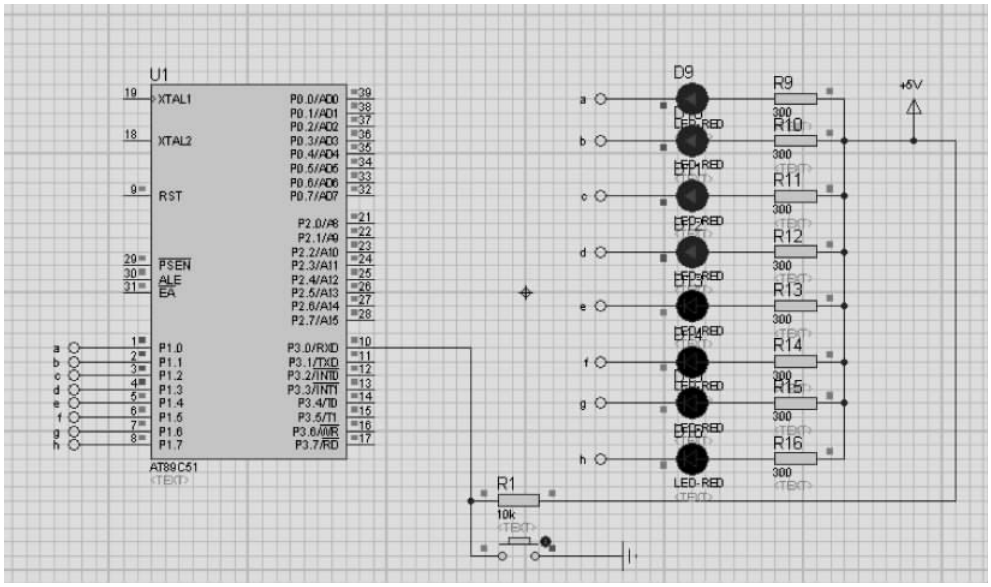

图 3-8　用按键 S 控制 P1 口 8 只 LED 的显示状态

3.4.2　任务 12-2　用 for 语句实现蜂鸣器发出 1kHz 音频

1. 任务要求

（1）掌握“for”语句功能及编程；
（2）掌握延时时间的估算方法；
（3）掌握“while”语句功能及编程；
（4）掌握延时程序的编写。

2. 任务描述

设计一个用 for 语句实现蜂鸣器发出 1kHz 音频的程序，要求如下。
（1）发出频率为 1kHz 音频。
（2）蜂鸣器接到 P1.0 引脚上。

3. 任务实现

（1）分析

设单片机晶振频率为 12MHz，则机器周期为 1μs。只要让单片机的 P1.0 引脚的电平信号每隔音频的半个周期取反一次即可发出 1kHz 的音频。音频的周期为 $T=1/1000\text{Hz}=0.001\text{s}$，即 1000$\mu$s，半个周期为 1000$\mu$s/2＝500$\mu$s，即在 P1.0 引脚上每 500$\mu$s 取反一次即可发出 1kHz 的音频。而延时 500μs 需要消耗机器周期数 $N=500\mu\text{s}/3=167$，即延时每循环 167 次，就可让 P1.0 引脚上取反一次得到 1kHz 的音频。

（2）程序设计

先建立文件夹“XM12-2”，然后建立“XM12-2”工程项目，最后建立源程序文件“XM12-2.c”，

输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
sbit sound=P1^0;                     //将 sound 位定义为 P1.0 引脚
/*************************
函数功能：延时以形成约 1kHz 的音频
*************************/
void delay1000Hz(void)
{
   unsigned char i;
    for(i=0;i<167;i++)
    ;
}
/*****************************
函数功能：主函数
*****************************/
void main(void)
{
  while(1)                           //无限循环
    {
        sound =0;                    //P1.0 引脚输出低电平
        delay1kHz();                 //延时以形成半个周期
        sound =1;                    //P1.0 引脚输出高电平
        delay1kHz();                 //延时以形成约 1kHz 的音频
    }
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM12-2.hex”文件加载到 AT89C51 里，然后启动仿真，就可以听到用 for 语句实现蜂鸣器发出 1kHz 音频，效果图如图 3-9 所示。

小结：消耗机器周期数的计算(近似值)如下。

(1) 一重循环

```
for(i=0;i<n;i++)                     //n 必须为无符号字符型数据
        ;
```

消耗机器周期数为：

$$N = 3 \times n$$

式中，N 为消耗机器周期数，n 为需要设置的循环次数(n 必须为无符号字符型数据)。

(2) 二重循环

```
for(i=0;i<n;i++)                     //n 必须为无符号字符型数据
for(i=0;i<m;i++)                     //m 必须为无符号字符型数据
        ;
```

消耗机器周期数为：

$$N = 3 \times n \times m$$

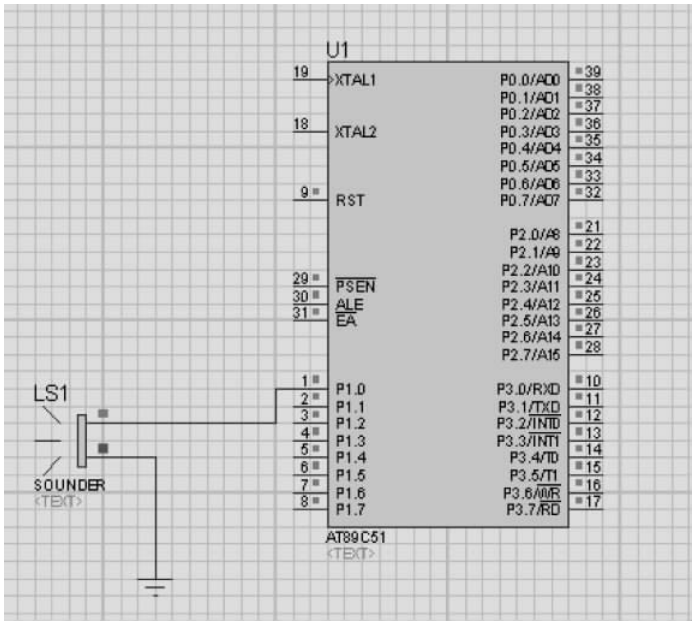

图 3-9　用 for 语句实现蜂鸣器发出 1kHz 音频

3.4.3　任务 12-3　用 while 语句控制 P1 口 8 只 LED 的显示状态

1. 任务要求

(1) 掌握“while”语句功能及编程；
(2) 掌握延时程序的编写。

2. 任务描述

设计一个用 while 语句控制 P1 口 8 只 LED 显示状态的程序，要求如下。
(1) P1 口接 8 只发光二极管，低电平点亮。
(2) 点亮发光二极管间隔为 150ms。
(3) 显示 99 种状态。

3. 任务实现

(1) 分析

设计一个用 while 语句控制 P1 口 8 只 LED 显示状态的程序，根据要求在 while 语句循环中设置一个变量 i，当 i 小于 0x64(十进制数 100)时，将 i 的值送 P1 口显示，并 i 自增 1，当 i 等于 0x64 时，就跳出 while 循环。

(2) 程序设计

先建立文件夹“XM12-3”，然后建立“XM12-3”工程项目，最后建立源程序文件“XM12-3.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
/**************************
```

```
函数功能：延时约 150ms
************************* /
void delay(void)
{
  unsigned char i,j;
    for(i = 0;i < 200;i++)
      for(j = 0;j < 250;j++)
        ;
}
/ ***************************
函数功能：主函数
*************************** /
void main(void)
{
  unsigned char i;
  while(1)                          //无限循环
    {
      i = 0;                        //将 i 置为 0,即初始化
     while(i < 0x64)                //i 小于 100 时执行循环体
       {
        P1 = i;                     //将 i 值送 P1 口显示
       delay;                       //调延时
        i++;                        //i 自增 1
       }
    }
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM12-3.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看到用 while 语句实现控制 P1 口 8 只 LED 的显示状态了，效果图如图 3-10 所示。

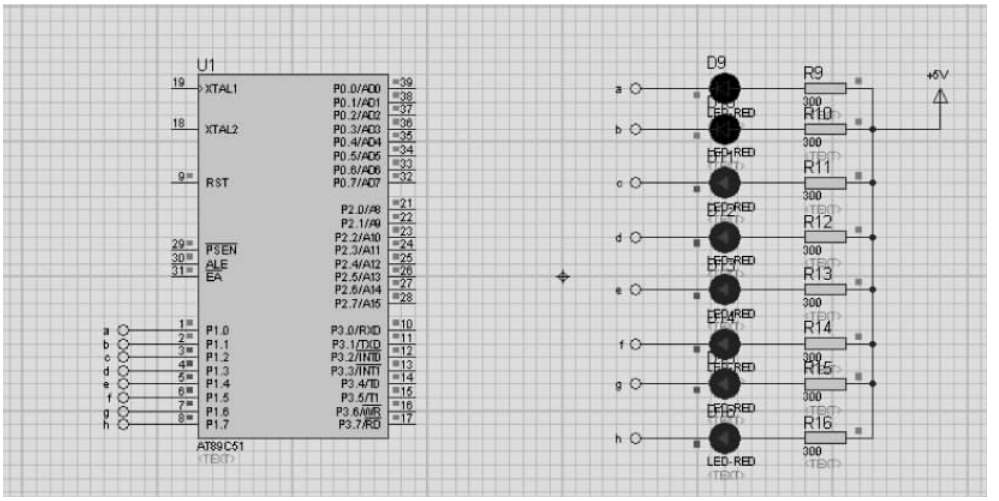

图 3-10　用 while 语句控制 P1 口 8 只 LED 的显示状态

3.4.4 任务 12-4 用 do…while 语句控制 P1 口 8 只 LED 的显示状态

1. 任务要求

（1）掌握“do…while”语句功能及编程；
（2）掌握延时时间的估算方法；
（3）掌握延时程序的编写。

2. 任务描述

设计一个用 do…while 语句控制 P1 口 8 只 LED 显示状态的程序，要求如下。
（1）P1 口接 8 只发光二极管，低电平点亮。
（2）点亮发光二极管间隔为 150ms。
（3）点亮次序为 LED1 发光，LED1、LED2 发光，LED1、LED2、LED3 发光，……，LED1～LED8 都发光，LED1 发光，LED1、LED2 发光……依次循环。

3. 任务实现

（1）分析

只要在循环体中按照点亮次序依次点亮，再将循环条件设置为死循环即可。现在来讨论点亮 LED 的控制码。LED1 发光的控制码为 0xfe；LED1、LED2 发光的控制码为 0xfc；LED1、LED2、LED3 发光的控制码为 0xf8…… LED1～LED8 都发光的控制码为 0x00；LED1 发光的控制码为 0xfe；LED1、LED2 发光的控制码为 0xfc……依次循环。

（2）程序设计

先建立文件夹“XM12-4”，然后建立“XM12-4”工程项目，最后建立源程序文件“XM12-4.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
/**************************
函数功能：延时约 150ms
**************************/
void delay(void)
{
   unsigned char i,j;
    for(i=0;i<200;i++)
      for(j=0;j<250;j++)
      ;
}
/*****************************
函数功能：主函数
*****************************/
void main(void)
{
  do
   {
```

```
        P1 = 0xfe;                  //LED1 点亮
        delay;                      //延时
        P1 = 0xfc;                  //LED1、LED2 点亮
        delay;                      //延时
        P1 = 0xf8;                  //LED1、LED2、LED3 点亮
        delay;                      //延时
        P1 = 0xf0;                  //LED1、LED2、LED3、LED4 点亮
        delay;                      //延时
        P1 = 0xe0;                  //LED1、LED2、LED3、LED4、LED5 点亮
        delay;                      //延时
        P1 = 0xc0;                  //LED1、LED2、LED3、LED4、LED5、LED6 点亮
        delay;                      //延时
        P1 = 0x80;                  //LED1、LED2、LED3、LED4、LED5、LED6、LED7 点亮
        delay;                      //延时
        P1 = 0x00;                  //LED1、LED2、LED3、LED4、LED5、LED6、LED7 点亮
        delay;                      //延时
    }while(1);                      //无限循环,需要注意此句的";"不能少
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后,在 Proteus ISIS 编辑环境中绘制仿真电路图,将编译好的"XM12-4.hex"文件加载到 AT89C51 里,然后启动仿真,就可以看到用 do…while 语句实现控制 P1 口 8 只 LED 的显示状态了,效果图如图 3-11 所示。

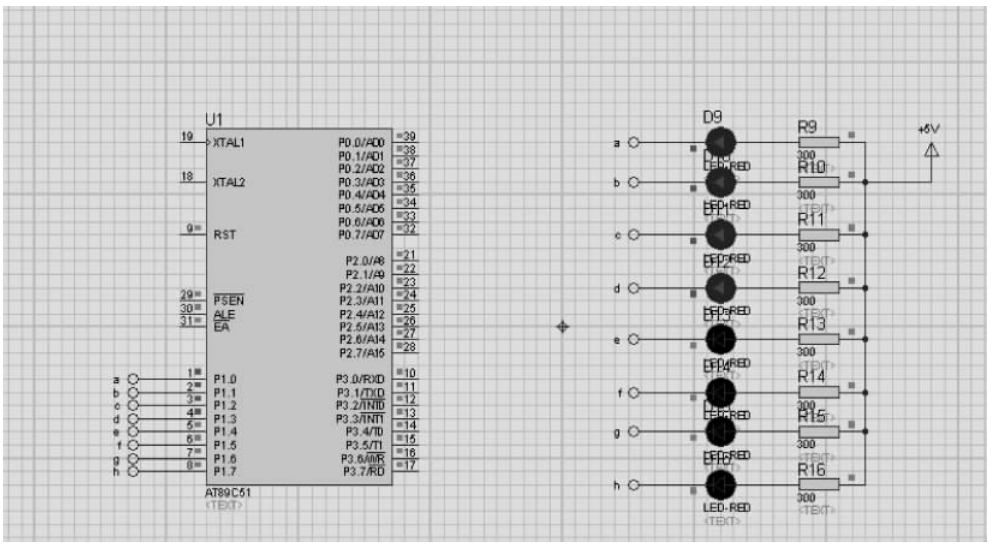

图 3-11　用 do…while 语句控制 P1 口 8 只 LED 的显示状态

3.4.5　任务 12-5　相关知识

1. 概述

顺序结构、选择结构和循环结构是实现所有程序的三种基本结构,也是 C51 语言程序的三种基本构造单元。选择结构体现了程序的逻辑判断能力,分支结构分为简单分支(两分

支)和多分支两种情况。一般采用 if 语句实现简单分支结构的程序,用 switch…case 语句实现多分支结构程序。循环结构解决了重复性的程序段的设计,主要有 for 语句、while 语句以及 do…while 语句。

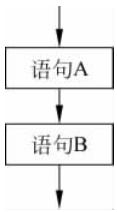

图 3-12　顺序结构

2. C51 的顺序结构

顺序结构是一种基本、最简单的编程结构。在这种结构中,程序由低地址向高地址顺序执行指令代码。如图 3-12 所示,程序先执行 A 操作,再执行 B 操作,两者是顺序执行的关系。

3. C51 的选择结构

选择语句就是条件判断语句,首先判断给定的条件是否满足,然后根据判断的结果决定执行给出的若干选择之一。在 C51 中,选择语句有条件语句和开关语句两种。

(1) 条件语句

条件语句由关键字 if 构成,它的基本结构是:

```
if (表达式)
  {语句};
```

如果括号中的表达式成立(为真),则程序执行花括弧中的语句;否则程序将跳过花括弧中的语句部分,执行下面的语句,C 语言提供了三种形式的 if 语句。

① 形式 1

```
if(表达式)
  {语句}
```

例如:

```
if (x>y)
printf("%d",x);
```

② 形式 2

```
if(条件表达式){语句 1;} else {语句 2}
```

例如:

```
if (x>y) max=x;
else max=y;
```

③ 形式 3

```
if(表达式 1){语句 1;}
    else if(表达式 2){语句 2;}
    else if(条件表达式 3){语句 3;}
         ⋮
    else if(条件表达式 n){语句 n;}
    else {语句 m}
```

例如：

```
if (salary>1000)     index=0.4;
  else if (salary>800)      index=0.3;
    else if (salary>600)     index=0.2;
      else if (salary>400)     index=0.1;
          else     index=0;
```

说明：if 语句的嵌套，在 if 语句中又含一个或多个 if 语句，这种情况称为 if 语句的嵌套。

(2) 开关语句

开关语句主要用于多分支的场合，一般形式为：

```
switch (表达式)
{
case 常量表达式 1: 语句 1;break;
case 常量表达式 2: 语句 2;break;
     ⋮
case 常量表达式 n: 语句 n;break;
default: 语句 n+1;
}
```

当 switch 括号中表达式的值与某一个 case 后面的常量表达式的值相等时，就执行它后面的语句，然后因遇到 break 而退出 switch 语句。当所有 case 中的常量表达式的值都没有与表达式的值相匹配时，就执行 default 后面的语句。

每一个 case 的常量表达式必须是互不相同的，否则就会出现对表达式的同一个值，有两种以上的选择。

如果 case 语句中遗忘了 break，则程序在执行了本行 case 选择之后，不会按规定退出 switch 语句，而是执行后续的 case 语句。

4. C51 的循环结构

程序设计中，常常要求进行有规律的重复操作，如求累加和，数据块的搬移等。几乎所有的实用程序都包含有循环结构。循环结构是结构化程序设计的三种基本结构之一，因此掌握循环结构的概念是程序设计，尤其是 C 程序设计最基本的要求。

在 C51 语言中，实现循环的语句主要有三种。

(1) while 语句的一般形式

```
while(表达式)
  {语句;   /*循环体*/}
```

while 语句的语义是：计算表达式的值，当值为真(非 0)时，执行循环体语句。

使用 while 语句应注意以下几点。

- while 语句中的表达式一般是关系表达式或逻辑表达式，只要表达式的值为真(非 0)即可继续循环。
- 循环体如包含一个以上的语句，则必须用“{}”括起来，组成复合语句。
- while 循环体中，应有使循环趋向于结束的语句，如无此种语句，循环将无休止地继

续下去，一直运行直至关机。

```
while (1)
  { }
```

这个语句的作用就是无限循环，一直运行直至关机。

(2) do…while 语句的一般形式

```
do
  {语句;}     /* 循环体 */
while (表达式);
```

do…while 循环语句的执行过程如下。首先执行循环体语句，然后执行圆括号中的表达式。如果表达式的值为真(非 0 值)，则重复执行循环体语句，直到表达式的值变为假(0 值)时为止。对于这种结构，在任何条件下，循环体语句至少会被执行一次。

(3) for 语句的一般形式

```
for (表达式 1;表达式 2;表达式 3)
  {语句;}     /* 循环体 */
```

有关 for 循环语句的执行过程和 for 循环的几种特殊结构，请读者参考 C 语言教材。

3.5 项目 13 认识 C51 的数组

- **技能目标**

掌握任务 13-1 用数组控制 P1 口 8 只 LED 的显示状态。

- **知识目标**

学习目的：

(1) 掌握一维数组；

(2) 掌握二维数组；

(3) 掌握字符数组；

(4) 掌握查表。

学习重点和难点：

(1) 一维数组；

(2) 二维数组；

(3) 字符数组；

(4) 查表。

3.5.1 任务 13-1 用数组控制 P1 口 8 只 LED 的显示状态

1. 任务要求

(1) 掌握“for”语句功能及编程；

(2) 掌握无符号字符型数组功能及编程；

(3) 掌握“while”语句功能及编程；

（4）掌握延时程序的编写。

2. 任务描述

用数组控制P1口8只LED的显示状态。设计一个程序用无符号字符型数组实现以下功能。先设置一个变量i，当$i=1$时，LED1发光（被点亮）；当$i=2$时，LED1、LED2发光；当$i=3$时，LED1、LED2、LED3发光；……；当$i=8$时，LED1～LED8都发光，当$i=9$时，LED1～LED8都熄灭，当$i=1$时，LED1发光……依次循环。

3. 任务实现

（1）分析

用无符号字符型数组来实现，大大简化了程序设计和节约了存储器空间，关键字为“code”，其定义如下。

```
unsigned char code Tab[ ] = {0xfe,0xfc,0xf8,0xf,0xe0,0xc0,0x80,0x00,0xff};/ * 定义无符号字符型数组,数组元素为点亮 LED 的状态控制码 * /
```

（2）程序设计

先建立文件夹“XM13-1”，然后建立“XM13-1”工程项目，最后建立源程序文件“XM13-1.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
/**************************
函数功能: 延时约 150ms
**************************/
void delay(void)
{
   unsigned char i,j;
    for(i = 0;i<200;i++)
      for(j = 0;j<250;j++)
        ;
}
/*****************************
函数功能: 主函数
*****************************/
void main(void)
{
  unsigned char i ;
  unsigned char code Tab[ ] = {0xfe,0xfc,0xf8,0xf,0xe0,0xc0,0x80,0x00,0xff};
                                  /* 定义无符号字符型数组,数组元素为点亮 LED 的状态控制码 * /
  while(1)                          //无限循环
    {
      for(i = 0;i<9;i++)
        {
          P1 = Tab[i];              //引用数组元素,送 P1 口点亮 LED
          Delay();                  //延时
        }
```

```
    }
}
```

（3）用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM13-1.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看到用数组控制 P1 口 8 只 LED 的显示状态了，效果图如图 3-13 所示。

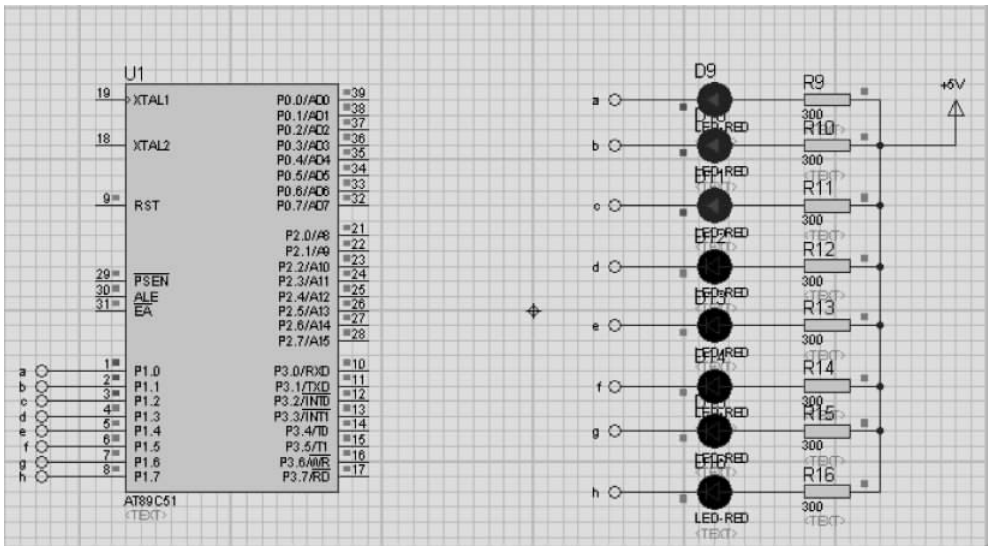

图 3-13　用数组控制 P1 口 8 只 LED 的显示状态

3.5.2　任务 13-2　相关知识

1. 概述

在程序设计中，为了处理方便，把具有相同类型的若干变量按有序的形式组织起来。这些按序排列的同类型数据元素的集合称为数组。

在 C 语言中，数组属于构造数据类型。一个数组可以分解为多个数据元素，这些数据元素可以是基本的数据类型或是构造类型。因此按数组元素的类型不同，数组又可分为数值数组、字符数组、指针数组、结构数组等各种类别。

2. 一维数组

（1）一维数组的定义方式

类型说明符 数组名［整型常量表达式］；例如，

```
int a[10];
```

它表示数组名为 **a**，此数组有 10 个元素。

说明：

① 数组名的命名规则和变量名相同，遵循标识符命名规则。

② 数组名后是用方括号括起来的常量表达式，不能用圆括弧。

③ 常量表达式表示元素的个数，即数组的长度。例如在 int **a**[10]中，10 表示数组 **a** 有 10 个数据元素，下标从 0 开始，这 10 个元素是 **a**[0]，**a**[1]，**a**[2]，**a**[3]，**a**[4]，**a**[5]，**a**[6]，**a**[7]，**a**[8]，**a**[9]，注意不能使用 **a**[10]。

④ 常量表达式中可以包括常量和符号常量，不能包含变量。也就是说，C51 不允许对数组的大小作动态定义，即数组大小不依赖于程序运行过程中变量的值。

(2) 一维数组的初始化

对数组元素的初始化可以用以下方法实现。

① 在定义数组时对数组元素赋予初值。

例如，int a[10]={0,1,2,3,4,5,6,7,8,9};

将数组元素的初值依次放在一对花括弧内。经过上面的定义和初始化之后，**a**[0]=0，**a**[1]=1，**a**[2]=2，**a**[3]=3，**a**[4]=4，**a**[5]=5，**a**[6]=6，**a**[7]=7，**a**[8]=8，**a**[9]=9。

② 可以只给一部分元素赋值。

例如，int a[10]={0,1,2,3,4};

定义数组 **a** 有 10 个元素，但花括弧内只提供 5 个初值，这表示只给前 5 个元素赋初值，后面的 5 个元素值为 0。

③ 在对全部数组元素赋初值时，可以不指定数组的长度。

例如，int a[5]={1,2,3,4,5};

也可以写成 int a[]={1,2,3,4,5}。

(3) 一维数组元素的引用

数组必须先定义，后使用。C51 语言规定只能逐个引用数组元素而不能一次引用整个数组。数组元素的表示形式为：

数组名[下标] 下标可以是整型常量或整型表达式，如 a[0]=a[5]+a[7]-a[2*3]。

3. 二维数组

(1) 二维数组定义的一般形式

类型说明符 数组名[常量表达式][常量表达式]

例如，int a[3][4]，b[5][10];

定义 **a** 为 3×4(3 行 4 列)的数组，**b** 为 5×10(5 行 10 列)的数组，数组元素为 int 型数据。

注意不能写成：int a[3,4]，b[5,10];

C51 语言对二维数组采用这样的定义方式，使我们可以把二维数组看作一种特殊的一维数组，它的元素又是一维数组。例如把 **a** 看作一个一维数组，它有三个元素：**a**[0]、**a**[1]、**a**[2]，每一个元素又是一个包含 4 个元素的一维数组，如图 3-14(a)、图 3-14(b)所示。

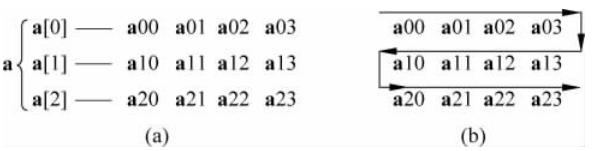

图 3-14　二维数组

(2) 二维数组的初始化

① 按行赋初值

数据类型 数组名[行常量表达式][列常量表达式]={{第零行初值表},{第一行初值表},……,{最后一行初值表}}。

② 按二维数组在内存中的排列顺序给各元素赋初值

数据类型 数组名 [行常量表达式][列常量表达式]={初值表}。

(3) 二维数组元素的引用

数组名 [行下标表达式][列下标表达式]

① “行下标表达式”和“列下标表达式”,都应是整型表达式或符号常量。

② “行下标表达式”和“列下标表达式”的值,都应在已定义的数组大小的范围内。

③ 对基本数据类型的变量所能进行的操作,也适合于相同数据类型的二维数组元素。

4. 字符数组

字符数组就是元素类型为字符型(char)的数组,字符数组是用来存放字符的。在字符数组中,一个元素存放一个字符,可以用字符数组来存储长度不同的字符串。

(1) 字符数组的定义

字符数组的定义和数组定义的方法类似。

如 char str[10],定义 str 为一个有 10 个字符的一维数组。

(2) 字符数组置初值

最直接的方法是将各字符逐个赋给数组中的各元素,如

```
char str[10] = {'M','I','A','N',' ','Y','A','N','G','\0'};   /* '\0'表示字符串的结束标志 */
```

C 语言还允许用字符串直接给字符数组置初值,其方法有以下两种形式。

```
char str[10] = {"Cheng Du"}; char str[10] = "Bei Jing"
```

5. 查表

在 C51 编程中,数组的一个非常有用的功能之一就是查表。

在实际单片机应用系统中,希望单片机能进行高精度的数学运算,但这并非单片机的特长,也不是完全必要的。许多嵌入式控制系统的应用中,人们更愿意用表格而不是数学公式,特别是在 A/D 转换中对模拟量的标定,使用表格查找法避免数值计算。在 LED 数码显示、LCD 的汉字显示系统中,一般将字符或汉字的点阵信息存放在表格中,表格可事先计算好装入 EPROM 中。

如一个摄氏温度转换成华氏温度的例子。

```
#define uchar unsigned char
uchar code tempt[ ] = {32,34,36,37,39,41};   /* 数组,设置在 EPROM 中,长度为实际输入的值 */
uchar f2c(uchar degr)
    { …
        return tempt(degr);}     /* 返回华氏温度值 */
```

```
void main()
    { uchar x;
    x = f2c(5);                    /* 得到 5℃ 相应的华氏温度 */
    }
```

3.6　项目 14　认识 C51 的指针

• 技能目标

(1) 掌握任务 14-1　用指针数组控制 P1 口 8 只 LED 的显示状态；

(2) 掌握任务 14-2　用指针数组实现多状态显示。

• 知识目标

学习目的：

(1) 了解指针的基本概念；

(2) 掌握指针变量的使用；

(3) 掌握数组指针和指向数组的指针变量；

(4) 掌握指向多维数组的指针和指针变量；

(5) 掌握关于 KeilC51 的指针类型。

学习重点和难点：

(1) 指针变量的使用；

(2) 数组指针和指向数组的指针变量；

(3) 指向多维数组的指针和指针变量；

(4) 关于 KeilC51 的指针类型。

3.6.1　任务 14-1　用指针数组控制 P1 口 8 只 LED 的显示状态

1. 任务要求

(1) 掌握指针的概念；

(2) 掌握指针运算符“*”功能及编程；

(3) 掌握无符号字符型数组功能及编程；

(4) 掌握“while”语句功能及编程。

2. 任务描述

用指针数组控制 P1 口 8 只 LED 的显示状态。设计一个程序用指针数组实现以下功能。先设置一个变量 i，当 $i=1$ 时，LED1 发光(被点亮)；当 $i=2$ 时，LED1、LED2 发光；当 $i=3$ 时，LED1、LED2、LED3 发光；……；当 $i=8$ 时，LED1～LED8 都发光，当 $i=9$ 时，LED1～LED8 都熄灭，当 $i=1$ 时，LED1 发光……依次循环。

3. 任务实现

（1）分析

用无符号字符型数组来定义控制码，其控制码值如下。

```
unsigned char code Tab[ ] = {0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80,0x00,0xff};
```

将其元素的地址依次存入如下指针数组。

```
unsigned char * p[ ] = {&Tab[0],&Tab[1],&Tab[2],&Tab[3],&Tab[4],&Tab[5],&Tab[6],&Tab[7],
&Tab[8]};
```

然后，利用指针运算符“*”取得各指针所指元素的值，送 P1 口 8 只 LED 显示。

（2）程序设计

先建立文件夹“XM14-1”，然后建立“XM14-1”工程项目，最后建立源程序文件“XM14-1.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
/**************************
函数功能：延时约 150ms
**************************/
void delay(void)
{
   unsigned int i;
    for(i=0;i<50000;i++)
            ;
}
/*****************************
函数功能：主函数
*****************************/
void main(void)
{
  unsigned char i ;                  //定义无符号字符型数据
  unsigned char code Tab[ ] = {0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80,0x00,0xff};/*定义无符号
字符型数组，数组元素为点亮 LED 的状态控制码*/
  unsigned char * p[ ] = {&Tab[0],&Tab[1],&Tab[2],&Tab[3],&Tab[4],&Tab[5],&Tab[6],&Tab[7],
&Tab[8]};                          //取点亮 LED 状态控制码地址，初始化指针数组
  while(1)                         //无限循环
    {
      for(i=0;i<9;i++)
        {
          P1 = * p[i];             //将指针所指数组元素值，送 P1 口点亮 LED
          delay();                 //延时
        }
    }
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM14-1.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看到用指针数组实现控制 P1 口 8 只 LED 的显示状态了，效果图如图 3-15 所示。

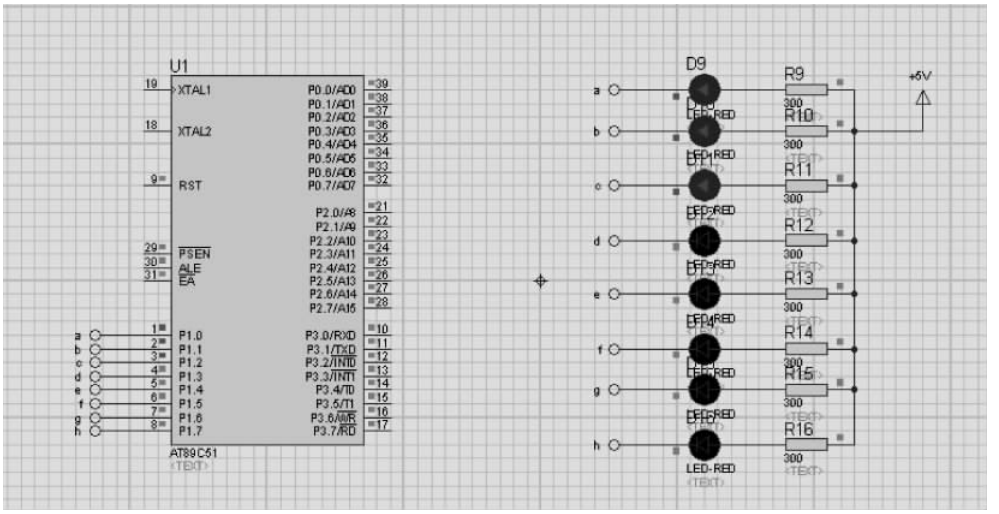

图 3-15　用指针数组控制 P1 口 8 只 LED 的显示状态

3.6.2　任务 14-2　用指针数组实现多状态显示

1. 任务要求

(1) 掌握指针运算符“*”的功能及编程；
(2) 掌握多状态显示利用指针数组编程的优点；
(3) 掌握数组关键字“code”的功能及编程；
(4) 掌握“while”语句功能及编程。

2. 任务描述

用指针数组实现多状态显示。任务要求①利用 P1 口 8 只 LED 的显示状态。②设计一个程序用指针数组实现以下功能。先设置一个变量 i，当 $i=1$ 时，LED1 发光(被点亮)，当 $i=2$ 时，LED1、LED2 发光，当 $i=3$ 时，LED1、LED2、LED3 发光，……；当 $i=8$ 时，LED1～LED8 都发光，当 $i=9$ 时，LED1～LED8 都熄灭；当 $i=10$ 时，LED1 发光，当 $i=11$ 时，LED2 发光，当 $i=12$ 时，LED3 发光，当 $i=13$ 时，LED4 发光，当 $i=14$ 时，LED5 发光，当 $i=15$ 时，LED6 发光，当 $i=16$ 时，LED7 发光，当 $i=17$ 时，LED8 发光；当 $i=18$ 时，LED1～LED4 发光，当 $i=19$ 时，LED5～LED8 发光；当 $i=20$ 时，LED1、LED3、LED5、LED7 发光。

3. 任务实现

(1) 分析

用无符号字符型数组来定义控制码，其控制码值如下。

```
unsigned char code Tab[ ] = {0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80,0x00,0xff,0xfe,0xfd,0xfb,
0xf7,0xef,0xdf,0x7f,0xf0,0x0f,0xaa};
```

将其元素的首地址赋给指针，通过指针引用数组元素值，送 P1 口点亮 LED。

（2）程序设计

先建立文件夹“XM14-2”，然后建立“XM14-2”工程项目，最后建立源程序文件“XM14-2.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
/**************************
函数功能：延时约 150ms
**************************/
void delay(void)
{
   unsigned int i;
    for(i=0;i<50000;i++)
            ;
}
/*****************************
函数功能：主函数
*****************************/
void main(void)
{
  unsigned char i ;                  //定义无符号字符型数据
  unsigned char code Tab[ ] = {0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80,0x00,0xff,0xfe,0xfd,0xfb,
0xf7,0xef,0xdf,0x7f,0xf0,0x0f,0xaa}; /*定义 20 个无符号字符型数组，数组元素为点亮 LED 的
状态控制码*/
 unsigned char  *p ;                 //定义无符号字符型指针
 p=Tab ;                             //将数组首地址存入指针 p
  while(1)                           //无限循环
    {
      for(i=0;ii<20;i++)//共有 20 个控制码
        {
          P1=*(p+i); /*[p+i]的值等于 a[i]，通过指针引用数组元素值，送 P1 口点亮 LED*/
          delay();                   //延时
        }
    }
}
```

（3）用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM14-2.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看到用指针数组实现多状态显示，效果图如图 3-16 所示。

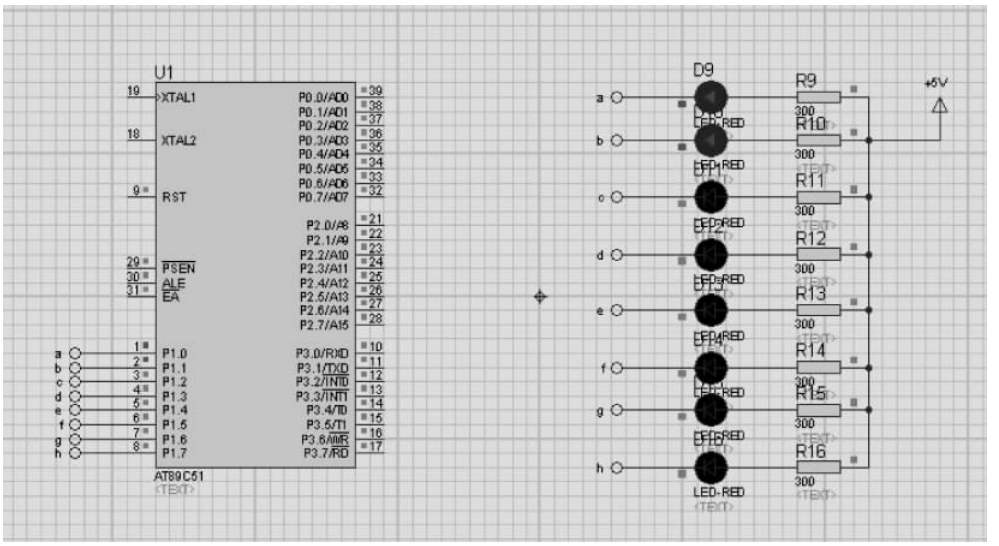

图 3-16　用指针数组实现多状态显示

3.6.3　任务 14-3　相关知识

1. 指针的基本概念

（1）地址

在程序中定义的变量都会在编译时分配对应的存储单元，变量的值存放在存储单元中，而存储单元都有相应的地址，访问变量首先要得到变量的存储单元地址，找到对应的存储单元地址后，再进一步对其中的值进行访问。除了得到变量单元的起始地址之外，还要根据变量的类型决定其存储字节数，将两者结合起来正确地访问变量。

对于变量，实际存在三个基本要素，即变量名、变量的地址和变量的值。变量名是变量的外在表现形式，方便用户对数据进行引用；变量的值是变量的核心内容，是设置变量的目的，设置变量就是为了对其中的值进行读写访问，变量的值存放在内存单元中；变量的地址则起到纽带的作用，把变量名和变量的值联系起来，通过变量名得到变量的地址，再通过变量地址在内存中寻址找到变量值。例如，通过通讯地址，可以确定居住区内的每个住户；知道了教室的门牌编号我们就能准确地找到要去的教室。

对于内存单元，也要明确两个概念，一个是内存单元的地址，一个是内存单元的内容。前者是内存对该单元的编号，它表示该单元在整个内存中的位置。后者指的是在该内存单元中存放着的数据。

（2）指针

变量存储单元的分配、地址的记录以及寻址过程虽然是在系统内部自动完成的，一般用户不需要关心其中的细节，但是出于对变量灵活使用的需要，有时在程序中围绕变量的地址展开操作，这就需要引入“指针”的概念。变量的地址称为变量的指针，指针的引入把地址形象化了，地址是找变量值的索引或指南，就像一根“指针”一样指向变量值所在的存储单元，因此指针即是地址，是记录变量存储单元位置的正整数。

（3）指针变量

指针是反映变量地址的整型数据，可以把指针值存放在另一个变量中，以便通过这个变量对存放在其中的指针进行操作，这个变量被称为“指针变量”。指针变量是专门存放其他变量地址的变量。指针变量虽然属于变量的范畴，但却不同于其他类型的变量。其他类型的变量用于存放被处理的数据即操作对象，可以对这些数据以“直接访问”的方式进行访问；而使用指针变量的目的并非针对存于其中的指针进行操作，而是为了通过这个指针对其指向的变量进行操作，因此这种访问被称为“间接访问”。

如图 3-17 所示，反映了指针变量与指针、指针与指针所指变量之间的关系。变量 n 是一般变量，变量 n 的指针（地址）又存放在指针变量 p 中，因此要存取变量 n 的值可以通过指针变量 p 以“间接访问”的方式进行。先从指针变量 p 中得到存放在其中的指针，即变量 n 的地址，在根据这个指针（地址）寻址，找到对应的存储单元，实现对变量 n 的访问。

图 3-17　指针与指针变量

2．指针变量的使用

（1）指针变量的定义

C 语言规定，所有的变量在使用前必须定义，以确定其类型。指针变量也不例外，由于它是专门存放地址的，因此必须将它定义为“指针类型”。

指针定义的一般形式为：类型识别符 * 指针变量名；

例如，

```
int *ap;
float *pointer;
```

注意：指针变量名前的“*”号表示该变量为指针变量。但指针变量名应该是 ap，pointer，而不是 *ap 和 *pointer。

（2）指针变量的赋值——取地址运算符“&”

指针变量既然是通过存放在其中的指针指向另外一个变量的，它就建立了与另外一个变量的联系。对指针变量的赋值实质就是要确定指向关系，即指针变量中到底存放了哪个变量的地址。

将指针变量指向某个变量的赋值格式通常是指针变量名＝& 所指向的变量名；

如要建立如图 3-17 所示的指针变量 p 与一般变量 n 的指向关系，则需要进行以下的定义和赋值。

```
int *p,n=10;
    p=&n;
```

(3) 指针变量的引用——指针运算符“*”

在进行了变量和指针变量的定义之后，如果对这些语句进行编译，C 编译器就会为每个变量和指针变量在内存中安排相应的内存单元，如

定义变量和指针变量。

```
int x = 1, y = 2, z = 3;          /* 定义整型变量 x,y,z */
int *x_point;                     /* 定义指针变量 x_point */
int *y_point;                     /* 定义指针变量 y_point */
int *z_point;                     /* 定义指针变量 z_point */
```

通过编译，C 编译器就会在变量 x、y、z 对应的地址单元中装入初值 1、2、3。如图 3-18(a)所示。但仍然没有对指针变量 x_point、y_point、z_point 赋值，所以它们所对应的地址单元仍为空白，即仍然没有被装入指针，它们没有指向。当执行 x_point＝&x、y_point＝&y、z_point＝z 后，指针 x_point 指向 x，即指针变量 x_point 所对应的内存地址单元中装入了变量 x 所对应的内存单元地址 1000；指针变量 y_point 所对应的内存地址单元中装入了变量 y 所对应的内存单元地址 1002；指针变量 z_point 所对应的内存地址单元中装入了变量 z 所对应的内存单元地址 1004，如图 3-18(b)所示。

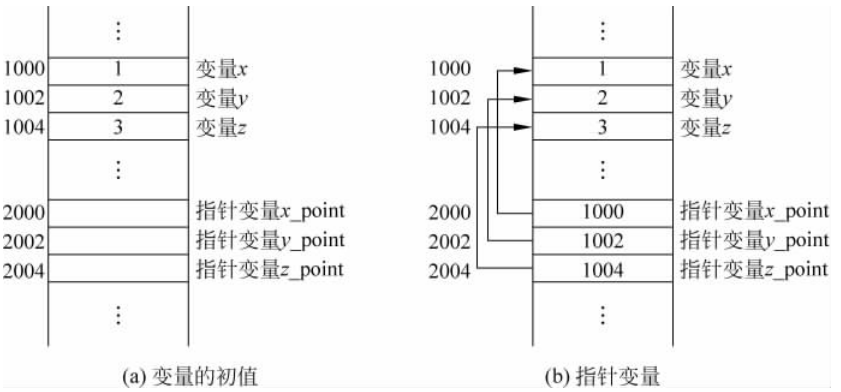

图 3-18　指针变量的引用

在完成了变量、指针变量的定义以及指针变量的引用之后，就可以通过指针和指针变量来对内存进行间接访问了。这时就要用到指针运算符(又称间接运算符)“*”。

如要把整型变量 x 的值赋给整型变量 a。

- 用直接访问方式，则用 $a=x$；
- 使用指针变量 x_point 进行间接访问，则用 $a=*x$_point。

应当特别注意的是：“*”在指针变量定义时和在指针运算时所代表的含义是不同的。在指针定义时，*x_point 中的“*”是指针变量的类型说明符。进行指针运算时，$a=*x$_point 中的“*”是指针运算符。

3. 数组指针和指向数组的指针变量

指针既然可以指向变量，当然也可以指向数组。所谓数组的指针，就是数组的起始地址。若有一个变量用来存放一个数组的起始地址(指针)，则称它为指向数组的指针变量。

(1) 指向数组的指针变量的定义、引用和赋值

首先定义一个数组 **a**[10]和一个指向数组的指针变量 array_ptr。

```
int a[10];                    /* 定义 a 为包含 10 个整型元素的数组 */
int *array_ptr;               /* 定义 array_ptr 为指向整型数据的指针 */
```

为了将指针变量指向数组 **a**[10]，需要对 array_ptr 进行引用，有如下两种引用方法。

① array_ptr=&a[0]

此时数组 **a**[10]的第一个元素 **a**[0]的地址就赋给了指针变量 array_ptr，也就是将指针变量 array_ptr 指向数组 **a**[ ]的第零号元素 **a**[0]。

② array_ptr =a

这种方法和①的作用完全相同，但形式上更简单。C 语言规定，数组名可以代表数组的首地址，即第一个元素的地址，因此下面两个语句是等价的。

```
array_ptr =&a[0];
array_ptr =a;
```

(2) 通过指针引用数组元素

引用数组元素，可以使用数组下标法如 **a**[4]，也可以使用指针法。与数组下标法相比，使用指针法引用数组元素能使目标代码效率高(占用内存少，运行速度快)。

通过指针引用数组元素。设指针变量 array_ptr 的初值为 &**a**[0]，如图 3-19 所示。从图 3-19 可以看出：

① array_ptr+i 和 $a+i$ 就是数组元素 **a**[i]的地址，它指向数组 **a**[ ]的第 i 个元素，由于 a 代表数组的首地址，则 $a+i$ 和 array_ptr+i 等价。

② *(array_ptr+i)和 *($a+i$)是 array_ptr+i 或 $a+i$ 所指向的数组元素，即 **a**[i]。

③ 指向数组的指针变量可以带下标，如 array_ptr[i]与 *(array_ptr+i)等价。

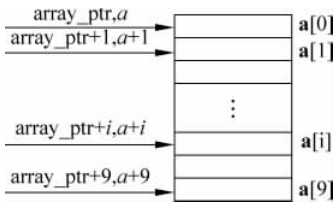

图 3-19 指针引用数组

【例 3-1】 设一个整型数组 **a**，有 10 个元素，要求输出全部的值。

解 要输出数组的全部元素的值有三种方法。

① 下标法。

```
#include <stdio.h>
   void main()
   {
    int a[10] = {12,3,45,6,20,30,78,50,66,81};
    int i;
    for (i = 0;i<10;i++)
    printf("%4d",a[i]);
```

```
    printf("\n");
}
```

② 通过数组名计算数组元素的地址，找出元素的值。

```
#include <stdio.h>
  void main()
  {
   int a[10] = {12,3,45,6,20,30,78,50,66,81};
   int i;
   for (i = 0;i<10;i++)
     printf(" %4d", *(a + i));
     printf("\n");
  }
```

③ 指针变量指向数组元素。

```
#include <stdio.h>
  void main()
  {
   int a[10] = {12,3,45,6,20,30,78,50,66,81};
   int *p;
   for (p = a;p<a + 10;p++)
  printf(" %4d", *p);
  printf("\n");
  }
```

(3) 关于指针变量的运算

若先使指针变量 p 指向数组 **a**[ ](即 p=**a**)，则

① p++（或者 p+=1）

该操作将使指针变量 p 指向数组 **a**[ ]的下一个元素，即 **a**[1]。若再执行 x=*p，则将 **a**[1]的值赋给变量 x。

② *p++

由于 * 和++运算符优先级相同，而结合方向是从右到左，故 *p++等价于 *(p++)。其作用是先得到 p 所指向的变量的值(即 *p)，再执行 p 自加运算。

③ *p++ 和 *++p 作用不同

*p++ 先取 *p 的值，后使 p 自加 1；*++p 先使 p 自加 1，再取 *p 的值。

④ (*p)++

表示 p 所指向的元素值加 1，而不是指针变量值加 1。若 p=a，即 p 指向 &**a**[0]，且 **a**[0]=12，则(*p)++等价于(**a**[0])++，此时 **a**[0]=13。

⑤ 若 p 当前指向数组的第 i 个元素 **a**[i]，则

*(p--)与 **a**[i--]等价，相当于先执行 *p，然后再使 p 自减 1。

*(++p)和 **a**[++i]等价，相当于先执行 p 自加 1，再执行 *p 运算。

*(--p)与 **a**[--i]等价，相当于先执行 p 自减 1，再执行 *p 运算。

4. 指向多维数组的指针和指针变量

以二维数组为例来说明指向多维数组的指针和指针变量的使用方法。

现在定义一个三行四列的二维数组 **a**[3][4]。

同时，定义这样一个(*p)[4]。它的含义是：*p* 是一个指针变量，指向一个包含 4 个元素的一维数组。下面使指针变量 *p* 指向 **a**[3][4]的首地址 *p*＝**a**[0]或者 *p*＝&**a**[0]。则此时 *p* 和 *a* 等价，均指向数组 **a**[3][4]的第零行首址(**a**[0][0])。

p＋1 和 *a*＋1 等价，均指向数组 **a**[3][4]的第一行首址(**a**[1][0])。

p＋2 和 *a*＋2 等价，均指向数组 **a**[3][4]的第二行首址(**a**[2][0])。

……

而 (*p*＋1)＋3 与 &**a**[1][3]等价，指向 **a**[1][3]的地址。

((*p*＋1)＋3) 与 **a**[1][3]等价，表示 **a**[1][3]的值。

一般，对于数组元素 **a**[*i*][*j*]来讲，有

(*p*＋*i*)＋*j* 就相当于 &**a**[*i*][*j*]，表示数组第 *i* 行第 *j* 列元素的地址。

((*p*＋*i*)＋*j*) 就相当于 **a**[*i*][*j*]，表示数组第 *i* 行第 *j* 列元素的值。

5. 关于 KeilC51 的指针类型

KeilC51 支持“基于存储器的”指针和一般指针两种指针类型。基于存储器的指针类型由 C 源代码中存储器的类型决定，并在编译时确定。由于不必为指针选择存储器，这类指针的长度可以为一个字节(idata *，data *，pdata *)或两个字节(code *，xdata *)，用这种指针可以高效访问对象。

(1) 基于存储器的指针

定义指针变量时，若指定了它所指向的对象的存储类型，该变量就被认为是基于存储器的指针，例如：

```
char xdata *px;
```

定义了一个指向 xdata 存储器中字符类型(char)的指针。指针本身在默认存储器(决定于编译模式)，长度为 2 字节(值为 0～0xffff)。

```
char xdata * data pdx;
```

除了确定指针位于 8051 内部存储区(data)中外，其他同上例，它与编译模式无关。

```
data char xdata *pdx;
```

与

```
char xdata * data pdx;
```

完全相同。存储器类型定义既可以放在定义的开头，也可以直接放在定义的对象名前。

还可以在定义时指定指针本身的存储空间位置，例如：

```
int xdata * idata i_ptr;
```

表示 i_ptr 指向的是 xdata 区中的 int 型变量，i_ptr 在片内 RAM 中。

```
long code * xdata l_ptr;
```

表示指向的是 code 区中的 long 型变量，l_ptr 在片外存储区 xdata 中。

(2) 一般指针

定义一般指针变量时,若未指定它所指向的对象的存储类型,该指针变量就认为是一个一般指针。一般指针包括三个字节:2 字节偏移和 1 字节存储器类型,如表 3-6 所示。

表 3-6　一般指针的字节内容

地址	+0	+1	+2
内容	存储器类型	偏移量高位	偏移量低位

其中,第一个字节代表了指针的存储类型,存储类型编码如表 3-7 所示。

表 3-7　指针的存储类型

存储类型	idata / data / bdata	xdata	pdata	code
编码值	0x00	0x01	0xFE	0xFF

(3) KeilC51 指针含义的汇编表示

①

```
unsigned char xdata * x;
x = 0x0456;
*x = 0x34 ;
```

等价于如下汇编程序段。

```
mov dptr,#456h
mov a,#34h
movx @dptr,a
```

②

```
unsigned char pdata * x;
x = 0x045;
*x = 0x34;
```

等价于如下汇编程序段。

```
mov r0,#45h
mov a,#34h
movx @r0,a
```

③

```
unsigned char data * x;
x = 0x30;
*x = 0x34
```

等价于如下汇编程序段。

```
mov a,#34h
mov 30h ,a
```

3.7 项目 15 认识 C51 的函数

• 技能目标

(1) 掌握任务 15-1 用带参数函数控制 8 位 LED 的闪烁时间；
(2) 掌握任务 15-2 用数组作为函数参数控制 8 位 LED 的点亮状态；
(3) 掌握任务 15-3 用指针作为函数参数控制 8 位 LED 的点亮状态；
(4) 掌握任务 15-4 用函数型指针控制 8 位 LED 的点亮状态。

• 知识目标

学习目的：
(1) 了解 C51 的函数概述；
(2) 了解函数的分类；
(3) 掌握函数的参数传递和函数值；
(4) 掌握函数的调用；
(5) 掌握 C51 函数的定义。

学习重点和难点：
(1) 掌握函数的参数传递和函数值；
(2) 掌握函数的调用；
(3) 掌握 C51 函数的定义。

3.7.1 任务 15-1 用带参数函数控制 8 位 LED 的闪烁时间

1. 任务要求

(1) 掌握“参数函数”的应用及编程；
(2) 掌握 LED 灯控制码设置；
(3) 掌握延时程序的编程与循环次数的计算；
(4) 掌握无限循环编程。

2. 任务描述

用带参数函数控制 P1 口 8 位 LED 的闪烁时间，相邻 LED 快速闪烁时的点亮间隔为 90ms，慢速闪烁时的点亮间隔为 300ms。

3. 任务实现

(1) 分析

设晶振频率为 12MHz，一个机器周期为 1μs，如果把内层循环次数设为 $m=100$ 时，则要延时 90ms，外循环次数为

$$n = \frac{90\,000}{3 \times 100} = 300$$

如果要延时 300ms，用上面公式同样可以计算出，外循环次数应为 $n=1000$。

（2）程序设计

先建立文件夹“XM15-1”，然后建立“XM15-1”工程项目，最后建立源程序文件“XM15-1.c”，输入如下源程序。

```
#include<reg51.h>
 /*************************
函数功能：用整型参数延时一段时间
 *************************/
void delay(unsigned int y)          //有参数传递
{
  unsigned int n,m;
 for(m=0;m<y;m++)
 for(n=0;n<100;n++)
;
}
/*************************
 函数功能:主函数(C语言规定必须有一个主函数)
 *************************/
 void main(void)
{
 unsigned char i;
unsigned char code Tab[ ] = {0x7f,0xbf,0xdf,0xef,0xf7,0xfb,0xfd,0xfe,0xaa,0xfe,0xfd,0xfb,
0xf7,0xef,0xdf,0xbf,0x7f}; /*LED灯控制码*/
while(1)
{
 for(i=0;i<17;i++)                  //共17个LED灯控制码

{
 P1=Tab[i];
delay(300);                         //延时约90ms,3×300×100=90 000(μs)=90(ms)
 }
        for(i=0;i<17;i++)           //共17个LED控制码

        {
          P1=Tab[i];
        delay(1000);                //延时约300ms,3×1000×100=300 000(μs)=300(ms)
        }
    }
}
```

（3）用Proteus软件仿真

经过Keil软件编译通过后，在Proteus ISIS编辑环境中绘制仿真电路图，将编译好的“XM15-1.hex”文件加载到AT89C51里，然后启动仿真，就可以看出用带参数函数控制8位LED的闪烁时间是不一样的，效果图如图3-20所示。

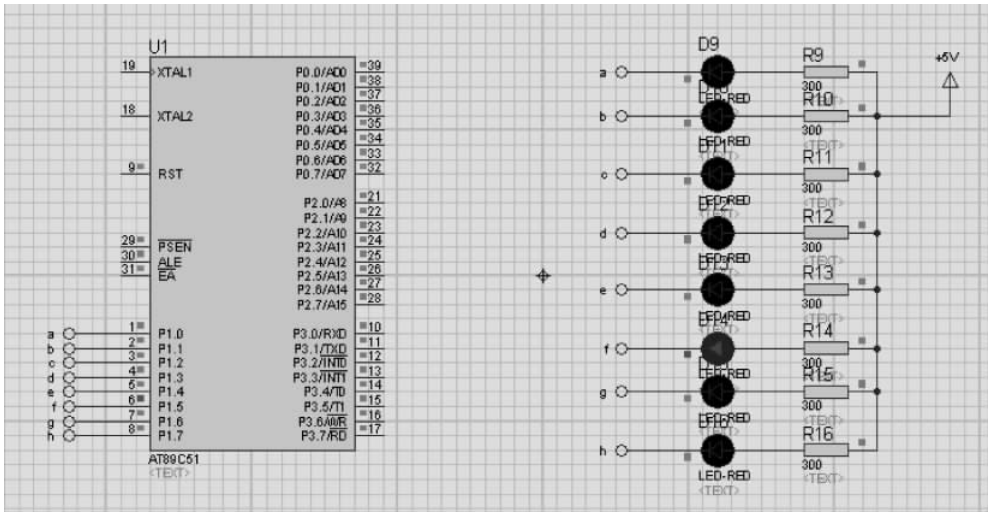

图 3-20　用带参数函数控制 8 位 LED 闪烁时间的仿真效果图

3.7.2　任务 15-2　用数组作为函数参数控制 8 位 LED 的点亮状态

1. 任务要求

（1）掌握“参数函数”的应用及编程；
（2）掌握数组的应用及编程；
（3）掌握 LED 灯控制码设置。

2. 任务描述

用数组作为函数参数控制 8 位 LED 的点亮状态，要求如下。
（1）用单片机的 P1 口；
（2）使用数组作为参数；
（3）设置 17 种 LED 灯控制码；
（4）延时采用 150ms。

3. 任务实现

（1）分析

先定义 17 种 LED 灯控制码数组，再定义 LED 的点亮函数，使其形参为数组，并且数据类型和实参数组（LED 灯控制码数组）的类型一致。

（2）程序设计

先建立文件夹“XM15-2”，然后建立“XM15-2”工程项目，最后建立源程序文件“XM15-2.c”，输入如下源程序。

```
#include<reg51.h>
/**************************
函数功能：延时约 150ms
**************************/
```

```
void delay(void)                    //两个 void 意思分别为无需返回值,没有参数传递
{
    unsigned int n;                 //定义无符号整数,最大取值范围为 65 535
    for(n = 0;n < 50000;n++)        //做 50 000 次空循环
            ;                       //什么也不做,等待一个机器周期
}
/ **************************
函数功能: 点亮 P1 口 8 位 LED
************************** /
    void led_flow(unsigned char a[17])
     {
        unsigned char i;
        for(i = 0;i < 17;i++)
          {
              P1 = a[i];            //取值送 P1 口显示
            delay();
          }
     }
     / **************************
    函数功能: 主函数
     ************************** /
    void main(void)
     {
       unsigned char code
       Tab[ ] = {0x7f, 0xbf, 0xdf, 0xef, 0xf7, 0xfb, 0xfd, 0xfe, 0xaa, 0xfe, 0xfd, 0xfb, 0xf7, 0xef,
0xdf, 0xbf, 0x7f}; / * LED 灯控制码 * /
       led_flow(Tab);               //将数组名作实参传给被调函数
     }
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后,在 Proteus ISIS 编辑环境中绘制仿真电路图,将编译好的"XM15-2.hex"文件加载到 AT89C51 里,然后启动仿真,就可以看到用数组作为函数参数控制 8 位 LED 的点亮状态,效果图如图 3-21 所示。

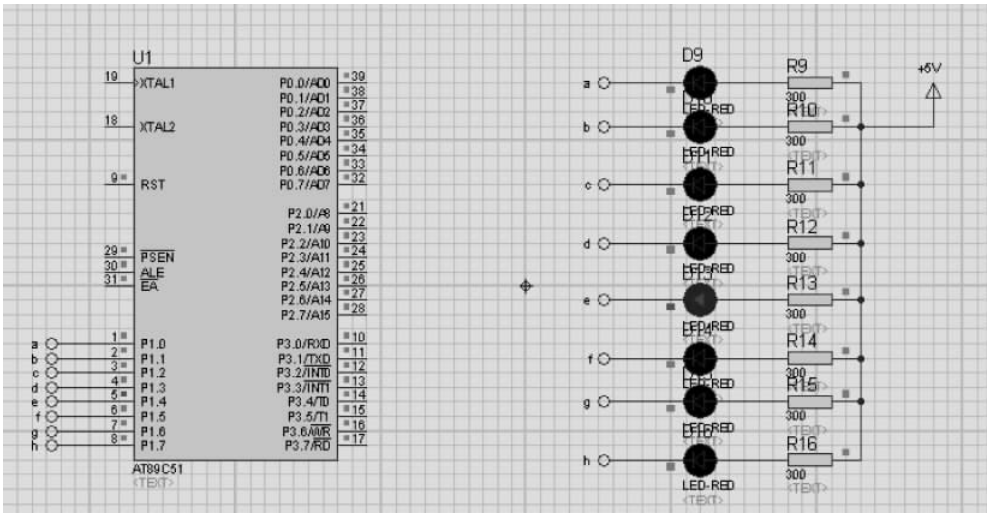

图 3-21　用数组作为函数参数控制 8 位 LED 点亮状态的仿真效果图

3.7.3 任务 15-3 用指针作为函数参数控制 8 位 LED 的点亮状态

1. 任务要求

（1）掌握“参数函数”的应用及编程；
（2）掌握“指针”的应用及编程；
（3）掌握“数组”的应用及编程；
（4）掌握 LED 灯控制码设置。

2. 任务描述

用指针作为函数参数控制 P1 口 8 位 LED 的点亮状态，要求如下。
（1）用单片机的 P1 口；
（2）使用指针作为函数参数；
（3）设置 20 种 LED 灯控制码；
（4）延时采用 150ms。

3. 任务实现

（1）分析

因为存储 LED 控制码的数组名即表示该数组的首地址，所以可以定义一个指针指向该首地址，然后用这个指针作为实际参数传递给被调用函数的形参。因为该形参也是一个指针，该指针也指向流水控制码的数组，所以只要用指针引用数组元素就可以实现控制 P1 口 8 位 LED 的点亮状态了。

（2）程序设计

先建立文件夹“XM15-3”，然后建立“XM15-3”工程项目，最后建立源程序文件“XM15-3.c”，输入如下源程序。

```
#include<reg51.h>
/**************************
函数功能：延时约 150ms
**************************/
void delay(void)                    //两个 void 意思分别为无需返回值，没有参数传递
{
    unsigned int n;                 //定义无符号整数，最大取值范围为 65 535
    for(n=0;n<50000;n++)            //做 50 000 次空循环
        ;                           //什么也不做，等待一个机器周期
}
/**************************
  函数功能：点亮 P1 口 8 位 LED
**************************/
  void led_flow(unsigned char *p)   //形参为无符号字符型指针
{
  unsigned char i;
    while(1)
      {
        i=0;                        //将 i 置为 0，指向数组第一个元素
        while(*(p+i)!='\0')         //只要没有指向数组的结束标志，就继续
          {
```

```
                P1 = *(p + i);          //取的指针所指数组元素的值,送 P1 口显示
                delay();                //调用 150ms 延时函数
                i++;                    //指向下一个数组元素
              }
        }
}
 /*************************
函数功能: 主函数
*************************/
 void main(void)
{
  unsigned char code Tab[ ] = {0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80,0x00,0xff,0xfe,0xfd,0xfb,
0xf7,0xef,0xdf,0x7f,0xf0,0x0f,0xaa}; /*定义 20 个无符号字符型数组,数组元素为点亮 LED 的
状态控制码*/
    unsigned char *pointer;          //定义无符号字符型指针 pointer
    pointer = Tab;                   //将数组的首地址赋给指针 pointer
  led_flow(pointer);                 //调用 LED 灯控制函数,指针为实际参数
  }
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后,在 Proteus ISIS 编辑环境中绘制仿真电路图,将编译好的"XM15-3.hex"文件加载到 AT89C51 里,然后启动仿真,就可以看到用指针作为函数参数控制 8 位 LED 的点亮状态了,效果图如图 3-22 所示。

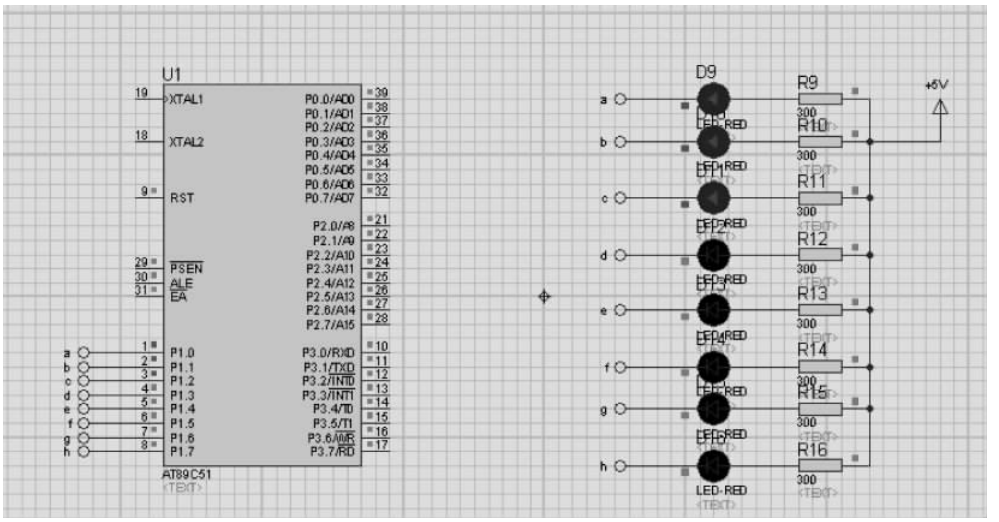

图 3-22　用指针作为函数参数控制 8 位 LED 点亮状态的仿真效果图

3.7.4　任务 15-4　用函数型指针控制 8 位 LED 的点亮状态

1. 任务要求

(1) 掌握 LED 灯作为点亮函数的设置;
(2) 掌握"函数型指针"的应用及编程;
(3) 掌握 LED 灯控制码设置。

2. 任务描述

用函数型指针控制 8 位 LED 的点亮状态，要求如下。

（1）用单片机的 P1 口；

（2）使用 LED 灯作为点亮函数；

（3）用函数型指针控制 8 位 LED 点亮状态；

（4）延时采用 150ms。

3. 任务实现

（1）分析

先定义 LED 灯点亮函数，再定义函数型指针，然后将 LED 灯点亮函数的名字（入口地址）赋给函数型指针，就可以通过该函数型指针调用 LED 灯点亮函数了。

注意：函数型指针的类型说明必须和函数的类型说明一致。

（2）程序设计

先建立文件夹“XM15-3”，然后建立“XM15-3”工程项目，最后建立源程序文件“XM15-3.c”，输入如下源程序。

```
#include<reg51.h>                    //包含 51 单片机寄存器定义的头文件
  unsigned char code Tab[] = {0xfe,0xfc,0xf8,0xf0,0xe0,0xc0,0x80,0x00,0xff,0xfe,0xfd,0xfb,
0xf7,0xef,0xdf,0x7f,0xf0,0x0f,0xaa}; /*定义 20 个无符号字符型数组，数组元素为点亮 LED 状
态控制码，该数组被定义为全局变量*/

/**************************
函数功能：延时约 150ms
**************************/
void delay(void)                     //两个 void 意思分别为无需返回值，没有参数传递
{
    unsigned int n;                  //定义无符号整数，最大取值范围为 65 535
    for(n=0;n<50000;n++)             //做 50 000 次空循环
            ;                        //什么也不做，等待一个机器周期
}

/*********************************
函数功能：点亮 P1 口 8 位 LED *********************************/
void led_flow(void)
  {
    unsigned char i;
    for(i=0;i<20;i++)                //20 位 LED 控制码
        {
          P1=Tab[i];                 //取数组值送 P1 口显示
          delay();                   //延时 150ms
        }
  }
  /*********************************
函数功能：主函数 *********************************/
void main(void)
{
```

```
    void ( * p)(void);                    //定义函数型指针,所指函数无参数,无返回值
    p = led_flow;                         //将函数的入口地址赋给函数型指针 p
    while(1)
    ( * p)();                             //通过函数的指针 p 调用函数 led_flow()
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后,在 Proteus ISIS 编辑环境中绘制仿真电路图,将编译好的"XM15-4.hex"文件加载到 AT89C51 里,然后启动仿真,就可以看到用函数型指针控制 8 位 LED 的点亮状态了,效果图如图 3-23 所示。

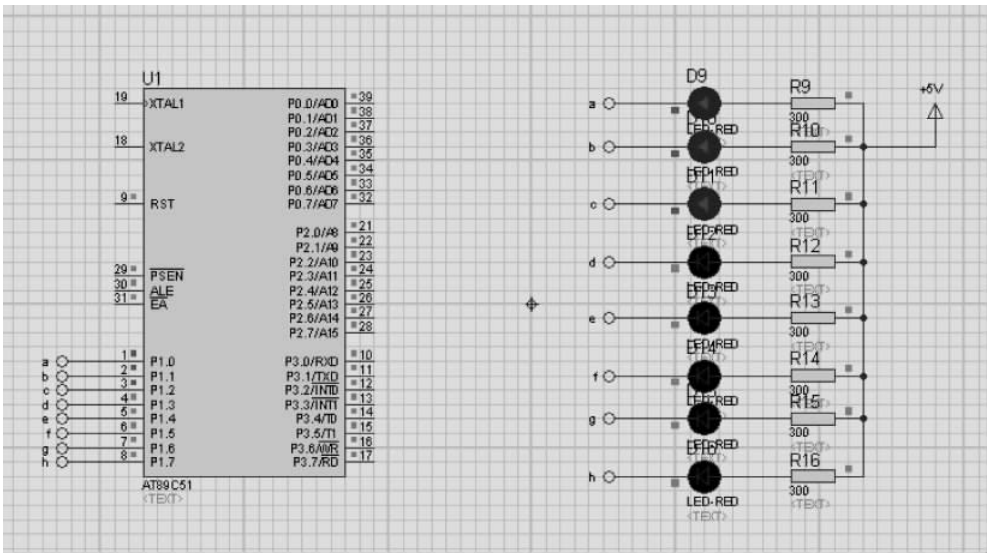

图 3-23　用函数型指针控制 8 位 LED 点亮状态的仿真效果图

3.7.5　任务 15-5　相关知识

1. C51 的函数概述

与普通的 C 语言程序类似,C51 程序是由若干模块化的函数构成的。函数是 C51 程序的基本模块,常说的"子程序"、"过程"在 C51 中用"函数"这个术语。它们都含有以同样的方法重复地去做某件事情的意思。主程序(main())可以根据需要用来调用函数。当函数执行完毕时,就发出返回(return)指令,而主程序后面的指令则用来恢复主程序流的执行。同一个函数可以在不同的地方被调用,并且函数可以重复使用。

前面的程序举例中可以看出,C 语言是由一个个函数构成的。在构成 C 语言程序的若干函数中,必有一个主函数(main())。如下所示是 C 语言程序的一般组成结构。

```
全程变量说明
main() /* 主函数 */
{
局部变量说明
执行语句
    }
```

主程序

```
function_1（数据类型 形式参数，数据类型 形式参数…）
    {
    局部变量说明
    执行语句
    …}
function_n（数据类型 形式参数，数据类型 形式参数…）
    {
    局部变量说明
    执行语句
    }
```

函数

所有函数在定义时都是相互独立的，一个函数中不能再定义其他的函数，即函数不能嵌套定义，但可以互相调用。函数调用的一般原则是：主函数可以调用其他普通函数。普通函数之间也可相互调用，但普通函数不能调用主函数。

一个 C 程序的执行总是从 main()函数开始的，调用其他函数后返回到 main()中，最后在主函数(main())中结束整个 C 程序的运行。

2. 函数的分类

从用户使用的角度划分，函数有两种：一种是标准库函数，一种是用户自定义函数。

(1) 标准库函数

标准库函数是由 C 编译系统的函数库提供的。早在 C 编译系统设计过程中，系统的设计者事先将一些独立的功能模块编写成公用函数，并将它们集中存放在系统的函数库中，供系统的使用者在设计应用程序时使用。故把这些函数库称做库函数或标准库函数。这类函数，用户无需定义，也不必在程序中作类型说明，使用时只需要在程序的开始用编译命令 #include 将头文件包含进来即可，就可以在程序中直接调用。因此，作为系统的使用者，在进行程序设计过程中，应该善于充分利用这些功能强大、资源丰富的标准库函数资源，可以大大提高编程的效率，节省时间。

C 编译系统提供的几类重要库函数如下。

- 专用寄存器 include 文件。
  例如 8031、8051 均为 reg51.h 其中包括了所有 8051 的 SFR 及其位定义，一般系统都必须包括本文件。
- 绝对地址 include 文件 absacc.h。
  该文件中实际只定义了几个宏，以确定各存储空间的绝对地址。
- 动态内存分配函数，位于 stdlib.h 中。
- 缓冲区处理函数位于“string.h”中，其中包括复制、比较、移动等函数，如 memccpy、memchr、memcmp、memcpy、memmove、memset ，这样能很方便地对缓冲区进行处理。
- 输入输出流函数，位于“stdio.h”中。

(2) 用户自定义函数

用户自定义函数，即用户根据自己的需要编写的函数。

从函数定义的形式上划分可以有三种形式：无参函数、有参函数和空函数。

无参函数。此类函数在被调用时，既无输入参数，也不返回结果给调用函数。它是完成

某种操作而编写的。

有参函数。在调用此类函数时，必须提供实际的输入参数。此种函数在被调用时，必须说明与实际参数一一对应的形式参数，并在函数结束时返回结果，供调用它的函数使用。

空函数。此种函数体内无语句，是空白的。调用此类函数时，什么工作也不做，不起任何作用。定义这种函数的目的是为了以后程序功能的扩充。

① 无参函数的定义

无参函数的定义形式如下。

```
数据类型 函数名()
    {
        函数体语句;
      }
```

其中，数据类型为函数返回值类型，如果函数不需要任何返回值，则需要定义成 void ；如果省略返回类型，则默认返回类型为 int，如

```
void PrintHello()
    {
        printf("Hello!\n");          /* 函数 PrintHello 将不返回值 */
    }
```

② 有参函数的定义

有参函数的定义形式如下。

```
返回值类型说明符 函数名(数据类型 变量名 1, 数据类型 变量名 2,…)
    {
        函数体语句;
    }
```

圆括号内的内容为函数的形式参数，形式参数之间必须用逗号隔开，它们构成了形参表，如求三个整数的最大数。

```
#include <stdio.h>
int max_abc (int a,int b,int c)
  {
    int d;
    d = (a > b)?(a > c?a:c):(b > c?b:c);
return (d);
  }
  void main(){
    int x = 12,y = -23,z = 43;
    int max;
    max = max_abc(x,y,z);
    printf("max = %d\n",max);
  }
```

③ 空函数的定义方法

空函数的定义形式为

```
返回值类型说明符 函数名()
        {  }
```

如

```
void DisplayLCD ()
        {  }
```

3. 函数的参数传递和函数值

函数之间的参数传递，通过主调用函数的实际参数与被调用函数的形式参数之间进行数据传递实现的。被调用函数的最后结果由调用函数的 return 语句返回给主调用函数。

(1) 形式参数。在定义函数时，函数名后面括号中的变量名称为“形式参数”，简称形参。

(2) 实际参数。在函数调用时，主调用函数名后面括号中的表达式称为“实际参数”，简称实参。需要注意以下两点。

- 在 C 语言的函数调用中，实际参数与形式参数之间的数据传递是单向进行的，只能由实际参数传递给形式参数，而不能由形式参数传递给实际参数。
- 实际参数和形式参数的类型必须一致，否则会发生类型不匹配的错误。被调用函数的形式参数在调用前，并不占用实际内存单元。只有当函数调用发生时，被调用函数的形式参数才被分配给内存单元，此时内存单元中调用函数的实际参数和被调用函数的形式参数位于不同的单元中。在调用结束后，形式参数所占用的内存被释放，而实际参数所占有的内存单元仍然保留并维持原值。

(3) 函数的返回值

函数返回值通过函数中的 return 语句来获得。

主调函数即 main()在调用有参函数 max_abc()时，将实际参数 x、y、z 传给被调用函数的形式参数 a、b、c。然后，被调用函数 max_abc 使用形式参数 a、b、c 作为输入变量进行计算，所得结果通过返回语句 return(d)返回给主函数，并在主函数的 max=max_abc(x,y,z)语句中赋值给变量 max。这个 return(d)中的 d 变量值就是被调用函数的返回值，简称函数的返回值。

函数调用时，主调函数与被调用函数之间的参数传递及函数值返回的全部过程示意图如图 3-24 所示。

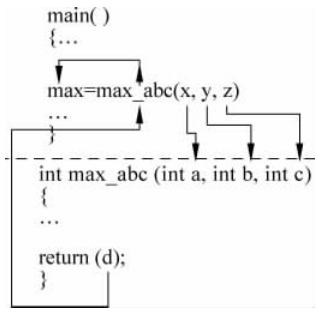

图 3-24　函数调用的参数传递过程

(4) return 语句的使用格式为：

```
return (表达式); 或 return 表达式; 或 return ;
```

使用时注意以下两点。

- 函数类型与 return 语句中表达式的类型尽量保持一致，若不一致，则以函数类型为准自动进行类型转换。
- 若被调用函数中没有 return 语句，函数没有返回值，提倡将函数的类型说明为 void 型，这样可使函数使用者明确函数的类型，避免调用时错误的产生。

4. 函数的调用

(1) 函数调用的一般形式

函数调用的一般形式为函数名(实际参数表列)；

对于有参函数，若包含多个参数，则各参数之间用逗号分隔开。主调函数的实参和被调函数的形参数目应该相等，且按顺序一一对应。如调用的是无参函数，则实际参数表可省略，但函数名后必须有一对空括号。

(2) 函数的调用方法

主调函数对被调函数的调用有以下三种方式。

① 函数调用语句

把被调用函数名作为主调函数的一个语句，如 PrintHello()；此时并不要求被调用函数返回结果数值，只要求函数完成某种操作。

② 函数结果作为表达式的一个运算对象

此时被调用函数以一个运算对象的身份出现在一个表达式中。这就要求被调用函数带有 return 语句，以便返回一个明确的数值参加表达式的运算，如 max=2 * max_abc(x,y,z)。

③ 函数参数

被调用函数作为另一个函数的实际参数。

如 printf("x=%d,y=%d,z=%d,max=%d",x,y,z,max_(x,y,z))；max_(x,y,z) 是一次函数调用。它的值作为另一个函数 printf()的实际参数之一。

在一个函数中调用另一个函数必须具有以下条件。

- 被调用函数必须是已经存在的函数(库函数或用户自定义函数)。

若程序中使用了库函数，或使用了不在同一个文件中的另外的自定义函数，则应在该程序的开头处使用 #include 包含语句，将所用的函数信息包含进来。如

```
#include<stdio.h>     /* 将标准的输入、输出头文件包含到程序中 */
#include<reg51.h>     /* 将包括了所有 8051 的 SFR 及其位定义的头文件包含到程序来 */
```

- 如果程序中使用自定义函数，且该函数与调用它的函数在同一个文件中，则应根据主调用函数和被调用函数在文件中的位置，决定是否对被调用函数做出说明。
- 如果被调用函数出现在主调用函数之后，一般应在主调用函数中，在对被调用函数调用之前，对被调用函数的返回值类型做出说明。

一般形式为 返回值类型说明符 被调用函数的函数名();

• 如果被调用函数出现在主调用函数之前，可以不对被调用函数加以说明。

（3）函数的嵌套和递归调用

在C语言中，尽管C语言中函数不能嵌套定义，但允许嵌套调用，即在调用一个函数的过程中，允许调用另一个函数。就80C51单片机而言，对函数的调用次数是有限制的，是由于其片内RAM中缺少大型堆栈空间所致。然而即便是使用80C51片内堆栈，倘若不传递参数，那么5～10层的函数嵌套调用也是不成问题的。所以对小规模程序而言，即使忽略嵌套调用的层次和深度，通常也是安全的。在调用一个函数的过程中，又直接或间接地调用函数本身。这种情况称为函数的递归调用。

5. C51函数的定义

C51函数的一般定义形式为：

```
返回值类型 函数名(形式参数列表)[编译模式][reentrant][interrupt m][using n]
    {
        函数体
    }
```

当函数没有返回值时，应用关键字void明确说明返回值类型。

形式参数的类型要明确说明，对于无形参的函数，括号也要保留。

编译模式为Small、Compact或Large，用来指定函数中局部变量参数和参数在存储器的空间。

reentrant用于定义可重入函数。

interrupt m用于定义中断函数，m为中断号，可以为0～31，但具体的中断号要取决于芯片的型号，像AT89C51实际上就使用0～4号中断。每个中断号都对应一个中断向量，具体地址为$8n+3$，中断源响应后处理器会跳转到中断向量所在的地址执行程序，编译器会在这地址上产生一个无条件跳转语句，转到中断服务函数所在的地址执行程序。

using n用于确定中断服务函数所使用的工作寄存器组，n为工作寄存器组号，取值为0～3。这个选项是指定选用51单片机芯片内部4组工作寄存器中的哪个组。开始学习者不必去做工作寄存器设定，而由编译器自动选择。

3.8 拓展项目实训

3.8.1 项目16 用P2口控制8只LED左循环流水灯亮

1. 项目要求

（1）掌握“左移”运算及编程；

（2）掌握二进移位；

（3）掌握循环次数设置及编程；

（4）掌握无限循环、延时编程。

2. 项目描述

用 P2 口控制 8 只 LED 左循环流水灯亮。把数“0xff”进行左移 8 位运算，实现 8 只 LED 左循环流水灯亮。

3. 项目实现

(1) 分析

设一个十六进制数“0xff”，展开为二进制数为 11111111B，进行左移 1 位“P2＝P2＜＜1”运算，即“11111111B→11111110”，规则为高位丢掉，低位添 0，把运算结果送 P2 口显示，使 LED0 亮，再进行左移 1 位运算“11111110→11111100”，把运算结果送 P2 口显示即 LED0、LED1 亮……经过 8 次左移后，P2＝00000000B，8 只 LED 灯全亮。然后重新使 P2＝0xff，如此循环，就可以实现 8 只 LED 左循环流水灯亮。

(2) 程序设计

先建立文件夹“XM16”，然后建立“XM16”工程项目，最后建立源程序文件“XM16.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
/*************************
函数功能：延时约 150ms
*************************/
void delay(void)
{
   unsigned char i,j;
    for(i=0;i<200;i++)
      for(j=0;j<250;j++)
        ;
}
/*****************************
函数功能：主函数
*****************************/
void main(void)
{
  unsigned char i ;
  while(1)                           //无限循环
    {
      P2=0xff;                       //设置初始值
      delay();                       //150ms 延时
      for(i=0; i<8; i++)             //循环次数设置为 8
        {
          P2=P2<<1;                  //左移一次
          delay();                   //150ms 延时
        }
    }
}
```

（3）用 Proteus 软件仿真

经过 Keil 软件编译通过后，在 Proteus ISIS 编辑环境中绘制仿真电路图，将编译好的“XM16.hex”文件加载到 AT89C51 里，然后启动仿真，就可以看到用 P2 口控制 8 只 LED 左循环流水灯亮，效果图如图 3-25 所示。

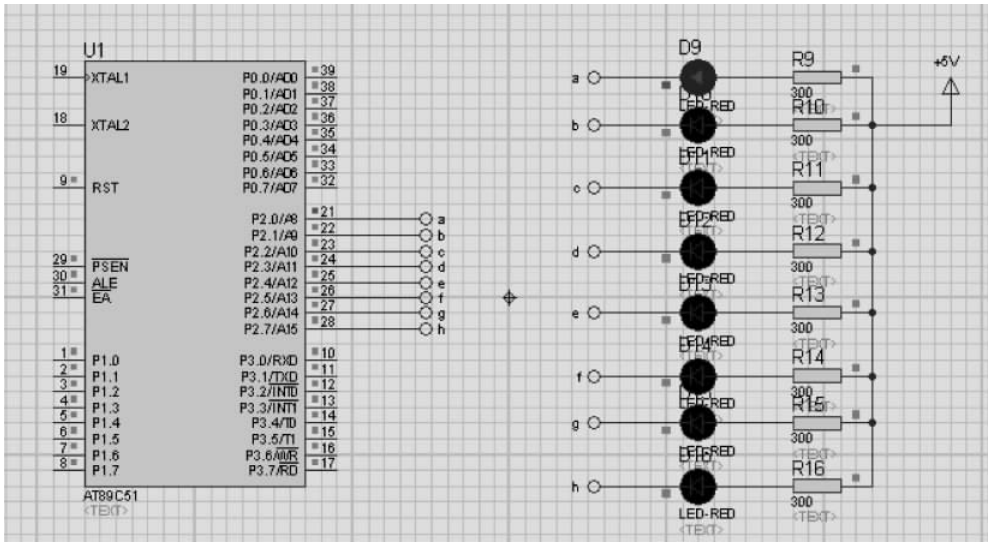

图 3-25　用 P2 口控制 8 只 LED 左循环流水灯亮

3.8.2　项目 17　用开关 S 控制蜂鸣器报警

1. 项目要求

（1）掌握“for”语句功能及编程；
（2）掌握延时时间的计算方法；
（3）掌握声音时间的计算方法；
（4）掌握“while”语句功能及编程；
（5）掌握“switch”语句功能及编程；
（6）掌握延时程序编写。

2. 项目描述

设计用两个开关 S1、S2 控制 P1.0 引脚实现蜂鸣器报警的程序，要求如下。
（1）开关 S1、S2 分别接到 P3.0、P3.1 引脚上。
（2）蜂鸣器接到 P1.0 引脚上。
（3）开关 S1 闭合发出频率为 1kHz 的声，发声时间为 1s。
（4）开关 S2 闭合发出频率为 500Hz 的声，发声时间为 0.5s。

3. 项目实现

（1）分析

设单片机晶振频率为 12MHz，则机器周期为 1μs。开关 S1、S2 闭合讨论如下。

① 开关 S1 闭合要求发出频率为 1kHz 的声，只要让单片机 P1.0 引脚的电平信号每隔音频的半个周期取反一次即可发出 1kHz 音频。音频的周期为 $T=1/1000\text{Hz}=0.001\text{s}$，即 1000μs，半个周期为 1000μs/2＝500μs，即在 P1.0 引脚上每 500μs 取反一次即可发出 1kHz 音频。而延时 500μs 需要消耗机器周期数 $N=500\mu\text{s}/3=167$，即延时每循环 167 次，就可让 P1.0 引脚上取反一次得到 1kHz 的音频。

发声时间的控制。1kHz 音频要求发声时间为 1s＝1000ms，而 1kHz 音频的周期为 1/1000Hz＝0.001s＝1ms，则需要 1000ms/1ms＝1000 个声音周期。

② 开关 S2 闭合要求发出频率为 500Hz 的音，只要让单片机 P1.0 引脚的电平信号每隔音频的半个周期取反一次即可发出 500Hz 音频。音频的周期为 $T=1/500\text{Hz}=0.002\text{s}$，即 2000μs，半个周期为 2000μs/2＝1000μs，即在 P1.0 引脚上每 1000μs 取反一次即可发出 500Hz 的音频。而延时 1000μs 需要消耗机器周期数 $N=1000\mu\text{s}/3=333$，即延时每循环 333 次，就可让 P1.0 引脚上取反一次得到 500Hz 的音频。

发声时间的控制。500Hz 音频要求发声时间为 0.5s＝500ms，而 500Hz 音频的周期为 1/500Hz＝0.002s＝2ms，则需要 500ms/2ms＝250 个声音周期。

(2) 程序设计

先建立文件夹“XM17”，然后建立“XM17”工程项目，最后建立源程序文件“XM17.c”，输入如下源程序。

```
#include<reg51.h>                    //包含单片机寄存器的头文件
sbit S1=P3^0;                        //定义按键 S1 接入 P3.0 引脚
sbit S2=P3^1;                        //定义按键 S2 接入 P3.1 引脚
sbit sound=P1^0;                     //将 sound 位定义为 P1.0 引脚
unsigned int keyval;
/**************************
函数功能：延时 30ms
**************************/
void delay(void)
{
   unsigned int i;
    for(i=0;i<1000;i++)
            ;
}
/**************************
函数功能：延时以形成约 1kHz 的音频
**************************/
void delay1kHz(void)
{
   unsigned char i;
    for(i=0;i<167;i++)
            ;
}
/**************************
/**************************
函数功能：延时以形成约 500Hz 的音频
**************************/
void delay500Hz(void)
```

```
{
   unsigned int i;
    for(i = 0;i < 333;i++)
            ;
}
/ ****************************
/ *************************
函数功能：发声 1s 时间的控制
************************* /
   void sound1s(void)
   {
    unsigned int i;
for(i = 0;i < 1000;i++)
{
sound  = 0;                              //P1.0 引脚输出低电平
delay1kHz();                             //延时以形成半个周期
sound  = 1;                              //P1.0 引脚输出高电平
delay1kHz();                             //延时以形成约 1kHz 的音频
}
}
/ *************************
函数功能：发声 0.5s 时间的控制
************************* /
   void soundBans(void)
   {
   unsigned char i;
for(i = 0;i < 250;i++)
{
sound  = 0;                              //P1.0 引脚输出低电平
delay500Hz();                            //延时以形成半个周期
sound  = 1;                              //P1.0 引脚输出高电平
delay500Hz();                            //延时以形成约 500Hz 的音频
}
}
/ *************************
函数功能：键盘扫描子程序
************************* /
void key_scan(void)
{

        if(S1 == 0)                      //按键 S1 被按下
        keyval = 1;                      //每个按键设置一个按键值
        delay();
        if(S2 == 0)                      //按键 S2 被按下
        keyval = 2;
        delay();
}

/ *************************
函数功能：主函数
************************* /
```

```
void main(void)                          //主函数
{
    keyval = 0;                          //按键值初始化为 0
    while(1)                             //无限循环
        {
            key_scan();                  //调用键盘扫描子程序
            switch(keyval)               //设置 switch 语句的条件表达式 keyval
                {
                    case 1: sound1s();   //如果 keyval = 1,则为发声 1s 时间的控制
                        break;           //跳出 switch 语句
                    case 2: soundBans(); //如果 keyval = 2,则为发声 0.5s 时间的控制
                        break;
                }
        }
}
```

(3) 用 Proteus 软件仿真

经过 Keil 软件编译通过后,在 Proteus ISIS 编辑环境中绘制仿真电路图,将编译好的"XM17.hex"文件加载到 AT89C51 里,然后启动仿真,就可以听到用开关 S 实现控制蜂鸣器报警了,效果图如图 3-26 所示。

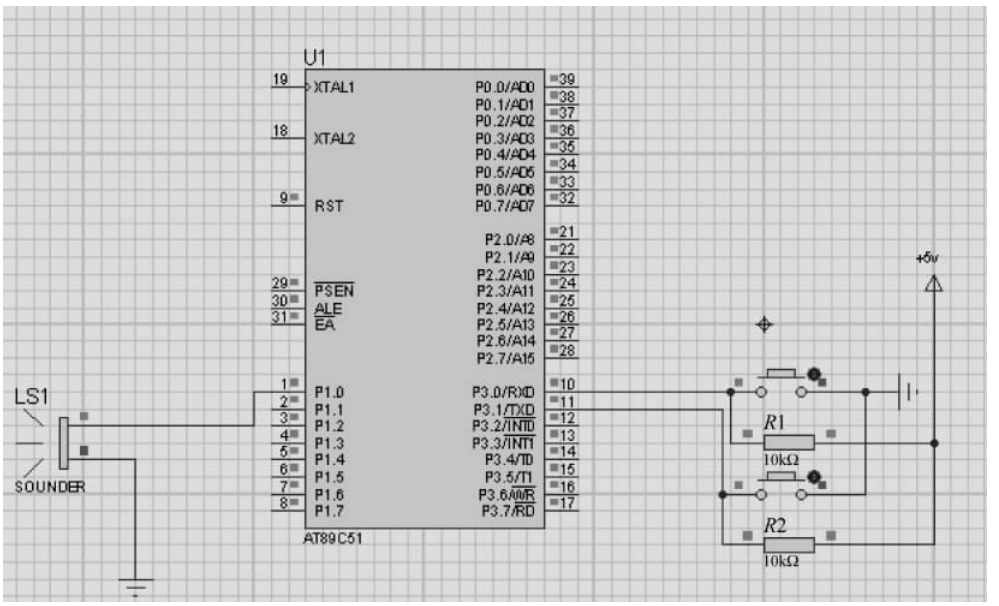

图 3-26　用开关 S 控制实现蜂鸣器报警

模块小结

C 语言单词包括标识符、关键字、运算符、分隔符、常量和注释等。标识符是一种单词,它用来给变量、函数、符号常量、自定义类型等命名。关键字是一种已被系统使用过的具有特定含义的标识符。用户不得再用关键字给变量等命名。

数据的不同格式叫数据类型。C51 的基本数据类型有字符型、整型、长整型、浮点型、位型及 SFR 型。存储类型是指变量被放在机器存储器中的位置。在 80C51 单片机中，C51 支持的存储类型有：bdata、data、idata、pdata、xdata、code。

80C51 硬件结构的 C51 定义；SFR 及其特定位的定义；并行接口及片外扩展 I/O 口的定义；位变量的定义。

C 语言常用运算符按功能可分为 6 类。算术运算符有＋，－，＊，/，％，＋＋，－－；关系运算符有＞，＜，＞＝，＜＝，＝＝，！＝；逻辑运算符有！，&&，||；位操作运算符有～，&，|，^，＞＞，＜＜；赋值运算符有＝，＋＝，－＝，＊＝，/＝，％＝，&＝，|＝，^＝，＞＞＝，＜＜＝；其他类运算符有[]，()，.，－＞，&，?：，，(逗号运算符)。

学习时着重理解每种运算符的功能、用法。每种运算符的优先级和结合性。优先级即执行运算的先后次序。结合性是指当一个运算对象两侧的运算符优先级别相同时的运算顺序。

(1) C51 的数组

① 数组是一种自定义的数据类型。这种数据类型的特点是它是数目固定、类型相同的若干变量的有序集合。

② 如何定义数组？定义数组的格式为

[<存储类>] <数据类型><数组名> [<大小>]…；

具有一个[＜大小＞]的是一维数组，具有两个[＜大小＞]的是二维数组。

③ 数组元素如何表示？

数组元素的表示方法为

<数组名> [<下标表达式>]…

具有一个[＜下标表达式＞]的为一维数组元素，具有两个[＜下标表达式＞]的为二维数组元素。

④ 如何对数组进行初始化？

定义或说明数组时，可以使用初始化表对数组进行初始化。

⑤ 如何给数组赋值？

给数组赋值是指给数组的每一个元素赋值，方法是使用赋值表达式语句对来给数组元素赋值。

⑥ 字符数组在初始化时有何特点？

字符数组可以用来存放若干字符，也可用来存放一个或多个字符串。一维字符数组只能放一个字符串，二维数组可存放多个字符串。用来存放字符串的数组可以直接使用字符串常量进行初始化。

(2) 指针

① 指针是一种用来存放某个变量或对象的地址值的特殊变量，指针存放了哪个变量或对象的地址值，该指针就指向哪个变量或对象，指针的类型是指指针所指变量或对象的类型。

② 指针的定义格式

<类型> *<指针名>=<初值>;

其中<指针名>同标识符，* 是修饰符，说明其后的标识符是指针名，不是一般变量，指针的数据类型不能省略，定义指针时可以初始化，也可以不进行初始化。

③ 指针的赋值

指针可以被赋初值，也可以被赋值。

给指针赋初值或赋值都必须是地址值，各种不同类型变量的地址值的表示方法不同。

在说明语句中，标识符 p 前边的 * 号是修饰符，用它来说明标识符 p 是指针名。赋值表达式中语句中的指针名 p 前边的 * 号是单目运算符，用它表示取指针 p 的内容，即指针 p 所指向的变量的值。

④ 使用指针表示数组元素

一维、二维数组的元素除了可用下标表示以外，还可以使用指针表示。熟悉指针表示各种数组元素的形式。

(3) 函数

① 函数的基本概念。函数的定义格式。函数的说明方法。函数的参数。函数的返回值。

② 函数的调用方法。传值调用方法和传址调用方法、嵌套调用、递归调用。

学习时注意：函数的定义和函数的说明是两回事。函数的实参和形参问题，调用函数的参数为实参，被调用函数的参数为形参。

函数的调用方式有：传递变量本身值的调用称为传值调用。传递变量地址值的调用称为传址调用。注意二者调用的机制和特点。

课后练习题

(1) C51 的数据存储类型有哪几种？几种数据类型各自位于单片机系统的哪一个存储区？

(2) Small、Compact、Large 三种编译模式的区别是什么？

(3) 定义一个可位寻址的变量 flag，该变量位于 23H 单元，用 sbit 指令定义该变量的 8 个位，变量名为 flag0、falg1、…、flag7。

(4) 求表达式的值，(float)(a+b)/2+(int)x%(int)y，设 a=3，b=4，x=3.5，y=4.5。

(5) 写出下列表达式运算后 a 的值，设运算前 a=10，n=9，a 和 n 已定义为整型变量。

① a+=a

② a *=2+3

③ a%=(n%=2)

④ a+=a-=a *=a

⑤ ++a+++a++

(6) 写出下列各逻辑表达式的值，设 a=3，b=4，c=5。

① a+b>c&&a=c

② $a||b+c\&\&a-c$

③ $!(a>b)\&\&!c||1$

④ $!(a+b)+c-1\&\&b+c/2$

(7) 输入三个数 x,y,z，请输出第二大的数。

(8) 求 $\sum_{n=1}^{10} n!$（即 $1!+2!+3!+\cdots+10!$）。

(9) 用三种循环语句分别实现 1 到 10 的平方和。

(10) 10 个元素的 int 数组需要多少字节存放？若数组在 2000H 单元开始存放，在哪个位置可以找到下标为 5 的元素？

(11) 用数组 math[10]存储从键盘上输入的一个班 10 名同学的数学成绩，①编程求其平均值；②将学生成绩按从高到低的顺序输出。

(12) 已知数组 str[255]中存放着一串字符，统计其中英文字母、数字及其他字符的个数。

(13) 求把字符串 s 逆转的函数 reverse(s)。

(14) 写出下列数组用 * 运算的替换形式。

① data[2]；　② num[i+1]；　③ man[5][3]。

(15) 用不同数据类型控制 P1 口 8 位 LED 的闪烁。

(16) 分别用 P2、P3 口显示“乘法”、“除法”的运算结果。

(17) 用 P1 口显示逻辑“与”运算结果。

(18) 用 P1 口显示逻辑“条件”运算结果。

(19) 用 P1 口显示逻辑“异或”运算结果。

(20) 分别用 P1、P2 口显示位“与或”运算结果。

(21) 用 P2 口显示“右移”运算结果。

(22) 用 if 语句控制 P1 口 8 只 LED 的显示状态。

(23) 用 for 语句实现蜂鸣器发出 2kHz 音频。

(24) 用 swtich 语句控制 P1 口 8 只 LED 的显示状态。

(25) 用 while 语句控制 P1 口 8 只 LED 的显示状态。

(26) 用 do…while 语句控制 P1 口 8 只 LED 的显示状态。

(27) 用数组控制 P2 口 8 只 LED 的显示状态。

(28) 用指针数组控制 P2 口 8 只 LED 的显示状态。

(29) 用指针数组控制实现 30 种 P1 口 8 只 LED 的显示状态。

(30) 用带参数函数控制 P0 口 8 位 LED 的闪烁时间。

(31) 用数组作为函数参数控制 P0 口 8 位 LED 的点亮状态。

(32) 用指针作为函数参数控制 P2 口 8 位 LED 的点亮状态。

(33) 用函数型指针控制 P2 口 8 位 LED 的点亮状态。

(34) 用 P1 口控制 8 只 LED 左循环流水灯亮。

(35) 用开关 S1、S2、S3、S4 控制蜂鸣器实现 4 个频率报警。