

第 3 章



单片机 C 语言开发基础

1.7 节已经介绍了 C 语言的结构、数据类型、运算符、函数,本章将主要通过 C 语言编程控制学习板上的流水灯,学习如何灵活运用 C 语言中的运算符、控制语句、数组、指针、预处理。本章内容可以说是对 C51 知识点的一个完整总结,内容较多,初学者全面掌握有一定难度。初学者对其中一些知识点可做简单了解,在后续章节的学习中再结合具体应用,以加深理解。

3.1 运算符的应用

C 语言中的运算符主要包括算术运算符、关系运算符、逻辑运算符、赋值运算符等。以下就是几个应用运算符来编程的实例。

【例 3-1】 用单片机实现乘法 78×18 的运算,并通过 P2 口的发光二极管分时显示结果的高八位和低八位状态。

分析:先设置两个字符型变量 i 和 j,将它们分别赋值为 78 和 18,可以先计算它们相乘的结果为 1404,等于十六进制数 0x057C,在程序中用变量 s 保存它们相乘的这个结果。因为 i 和 j 的值小于 255,所以用字符型变量保存即可;变量 s 的值大于 255 并小于 65535,所以必须保存为整型变量。相乘的十六位结果在八位并口 P2 上显示,只能把它拆成高八位和低八位分别显示,显示时,为区别高八位和低八位,它们中间让发光二极管全灭,并停顿 1s。变量 s 高八位的二进制数是 0000 0101B,因为发光二极管的状态是并口为高时熄灭,所以高八位送显示时,将有最低位、倒数第二位的灯熄灭,其他灯亮;变量 s 低八位的二进制数是 0111 1100B,当高八位送显示时,将有最高位、最低位两位灯亮,其他灯熄灭。我们可以把以下程序下载到学习板,观察显示状态是否正确。

```
#include <reg51.h>
#define uint unsigned int           //宏定义
#define uchar unsigned char
delay()
{
    uint m,n;
```

```

for(m = 1000; m > 0; m--)
    for(n = 110; n > 0; n--);
}
void main()
{
    uint s;           //保存乘法结果
    uchar i, j;       //保存相乘的因数
    i = 78;
    j = 18;
    s = i * j;
    while(1)
    {
        P2 = s/256;    //取乘积的高八位送 P2 口显示
        delay();
        P2 = 0xff;
        delay();
        P2 = s % 256;  //取乘积的低八位送 P2 口显示
        delay();
    }
}

```

程序中用指令“ $P2 = s/256$ ”取变量 s 的高八位送显示,指令右面的算式变量 s 除以 256 后取整,所以 $P2$ 得到的是乘积的高八位。而用指令 $P2 = s \% 256$ 取变量 s 的低八位送显示,符号“ $\%$ ”表示取 s 和 256 相除的余数,即变量 s 的低八位。通过这个例子可以练习除法和取余运算的用法。如果修改程序中 i 和 j 所赋的初值,还可以得到其他情况下乘法运算的结果。

这个程序里用到了宏定义,宏定义的格式为:

```
#define 新名称 原内容
```

`#define` 命令的作用是:用“新名称”代替后面的“原内容”,一般用于“原内容”比较长,又在程序里反复用到的情况。这样如果在程序中出现“原内容”,就可以用一个比较简短的“新名称”代替,使程序的书写更加简化。例如,本例在程序开始已经做了宏定义“`#define uint unsigned int`”,在此宏定义的后面,所有应该写 `unsigned int` 的地方,都用 `uint` 代替了。同一个程序中,宏定义对一个内容只能定义一次。

【例 3-2】 用 16 个发光二极管显示除法运算结果。

在学习板上除了 $P2$ 口接的 8 个发光二极管以外, $P3$ 口利用串并转换接口芯片 74HC595 也扩展了 8 个发光二极管。发光二极管硬件驱动电路如图 3-1 所示。

其中,芯片 74HC595 是八位串行输入转并行输出移位寄存器。引脚 $SER(14)$ 是串行移位输入引脚,串行数据从低位到高位在该引脚输入;引脚 $SRCLK(11)$ 移位时钟输入引脚,该引脚的上升沿可以使 14 脚的数据输入芯片内,即该引脚的上升沿控制数据串行移入;引脚 $RCLK(12)$ 并行输出时钟端,通常情况下该引脚保持低电平,当串行数据移入完成时,

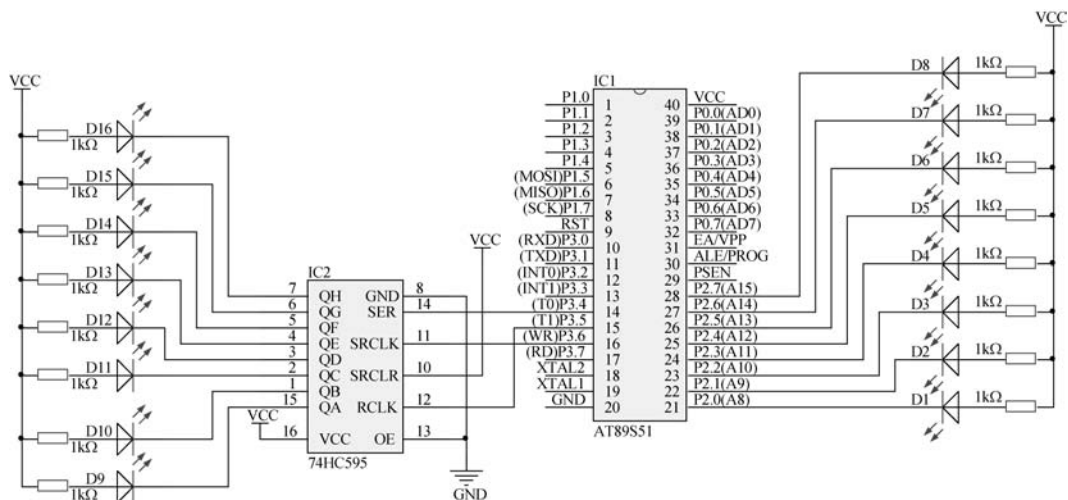


图 3-1 发光二极管硬件驱动电路图①

该引脚产生一个上升沿,将刚才移入的数据在 QA~QH 端并行输出。由 74HC595 的工作原理可知,仅用这一个芯片就可以只占用单片机的 3 个 I/O 口(即 P3.4~P3.6)来驱动 8 个发光二极管,大大节约了硬件资源。感兴趣的读者可以把例 3-1 中乘法运算的程序修改一下,结果在图 3-1 的 16 位发光二极管上显示出来。

假设本例中除法运算为“10÷6”结果只保留一位小数,结果中整数部分可以在 P2 口的发光二极管上显示,小数部分在 74HC595 扩展的发光二极管上显示。程序如下:

```
#include <reg51.h>
#define uint unsigned int
#define uchar unsigned char
sbit data1 = P3^4;           //定义 74HC595 中用到的几个口
sbit iclk = P3^6;
sbit oclk = P3^5;
uchar i, j, k, m;
void delay()
{
    uint a, b;
    for(a = 10; a > 0; a--)
        for(b = 110; b > 0; b--);
}
void xianshi(uchar m)       //74HC595 显示子程序
{
    uchar n;
    n = m;                  //要显示的数存放在 n 里
    oclk = 0;
    iclk = 0;
    for(j = 8; j > 0; j--)   //要显示的数左移串行输入
```

① 接地符号与软件界面截图保持一致,全书同。

```

    {
        n = n << 1;
        data1 = CY;
        iclk = 1;
        delay();
        iclk = 0;
    }
    oclk = 1;           //移位完成并行输出
    delay();
    oclk = 0;
}
void main()
{
    i = 10;              //将被除数和除数分别赋给 i 和 j
    j = 6;
    P2 = i/j;
    k = ((i % j) * 10)/j; //小数位保存在 k 中
    xianshi(k);          //调用显示子程序
    while(1);
}

```

这个程序主要用到了 74HC595 的显示驱动函数,命名为“xianshi()”,它的形参为 m,即要送 D9~D16 显示的数据。在这个子函数里,首先定义了一个无符号字符型变量 n,用来存放要显示的数,然后将 74HC595 的串行输入时钟和并行输出时钟都置为低电平。再开始串行输入:n 左移一位,移出的位移到了 CY 里,再把 CY 的值(第一次移位是显示数据最高位)给 74HC595 串行数据输入端,再将串行输入时钟置高电平,该时钟原来为低电平,因此在该引脚形成了一个上升沿,将串行数据输入端的当前值移入 74HC595,再延时将时钟端置低。串行数据输入的过程要反复移位 8 次,才能将要显示的八位二进制数全部移入,这里用了一个 for 循环,控制移位一共进行 8 次。当跳出循环时移位完成,此时需要将八位二进制数并行输出,输出通过并行输出时钟控制:并行输出时钟原值为低电平,现在置为高电平,形成一个上升沿,这个上升沿控制 74HC595 将刚才移入的数据并行由 QA~QH 输出。输出完成后,再将输出时钟端置为低电平,为下一次显示做准备。

主程序里先将被除数和除数分别赋值给 i 和 j,然后将 i 和 j 相除的结果取整后直接赋值给 P2 口,这是商的整数部分。小数部分的计算方法为:先将 i 和 j 相除后取余数,再扩大 10 倍,再用除数除后取整,这样计算得到的小数部分只有小数点后的第一位,再调用 74HC595 的显示子函数,在发光二极管 D9~D16 显示这个小数位。发光二极管熄灭表示 1,点亮表示 0,通过观察学习板上发光二极管的亮灭状态,即可知道除数运算的结果。

上述方法在单片机控制小数点的显示时非常有用。也可以把本例中被除数和除数的值变化一下,观察发光二极管的状态有什么变化。另外,读者也可以考虑:如果想显示更多的小数位时,程序应如何编写。

【例 3-3】 用自增、自减运算控制 P2 口的流水灯。

如果对变量 i 执行自增运算时写成“i++”,执行完 i 的值被加 1;对变量 i 执行自减运算时写成“i--”,执行完变量 i 的值被减 1。这两个运算符主要用在 for 循环的表达式里,用来修改循环指针。

```
#include <reg51.h>
void delay()           //延时 1s 子程序
{
    unsigned int i,j;
    for(i=1000;i>0;i--)
        for(j=110;j>0;j--);
}
void main()
{
    unsigned char m,n;
    while(1)
    {
        for(m=0;m<10;m++)           //m 从 0 到 9 自增 1,状态从 P2 输出
        {
            P2 = m;
            delay();
        }
        for(n=10;n>0;n--)           //n 从 10 到 1 自减 1,状态从 P2 输出
        {
            P2 = n;
            delay();
        }
    }
}
```

这个程序可以实现 P2 口的流水灯按一定规律变化。端口 P2 驱动的发光二极管在端口数据为 1 时,灯灭,在数据为 0 时,灯亮,亮灭规律都是按照二进制数的形式变化。程序中两个 for 循环使 P2 口输出数据按从 0 到 10,再从 10 到 0 的规则循环变化,相应的发光二极管也按这个规律点亮或熄灭。从观察 P2 口灯亮灭状态的变化,就可以知道自增和自减指令的功能了。

3.2 C 语言的语句

一个完整的 C 程序是由若干条 C 语句按一定的方式组合而成的。按 C 语句执行方式的不同,C 程序可分为顺序结构、选择结构和循环结构。

- 顺序结构:指程序按语句的顺序逐条执行。
- 选择结构:指程序根据条件选择相应的执行顺序。
- 循环结构:指程序根据某条件的存在重复执行一段程序,直到这个条件不满足为止。如果这个条件永远存在,就会形成死循环。

一般的C程序都是由上述三种结构混合而成的。但要保证C程序能够按照预期的意图运行,还要用到以下5类语句来对程序进行控制。

1. 控制语句

控制语句完成一定的控制功能,C语言中有9种控制语句。

- if...else: 条件语句;
- for: 循环语句;
- while: 循环语句;
- do...while: 循环语句;
- continue: 结束本次循环语句;
- break: 终止执行循环语句;
- switch: 多分支选择语句;
- goto: 跳转语句;
- return: 从函数返回语句。

2. 函数调用语句

调用已定义过的函数,如延时函数。

3. 表达式语句

由一个表达式和一个分号构成,示例如下:

```
z = x + y;
```

4. 空语句

空语句什么也不做,常用于消耗若干机器周期,延时等待。

5. 复合语句

用“{ }”把一些语句括起来就构成了复合语句。

以下重点学习一些控制语句的编程方法。

3.2.1 if 语句

if语句用来判定所给条件是否满足,根据判定的结果(真或假)选择执行给出的两种操作之一。if语句有3种基本形式:当表达式成立时,执行表达式后面的语句,不成立跳过该语句;当表达式不成立时,执行else后面的语句,当表达式成立时,执行if后面的语句;当表达式有多个时,哪个表达式成立,就执行相应表达式后面的语句,都不成立时,就执行最后一个else后面的语句。它们的格式如下:

```
(1) if(表达式)
(2) if(表达式)
    语句 1
    else
    语句 2
```

```

(3) if(表达式 1)
    语句 1
else
    if(表达式 2)
        语句 2
    else
        if(表达式 3)
            语句 3
        ...
    else
        语句 n

```

【例 3-4】 用 if 语句控制 P2 口一个流水灯从低位到高位循环移位点亮。



例 3-4 详解

```

#include <reg51.h>
void delay()
{
    unsigned int i,j;
    for(i = 1000; i > 0; i-- )
        for(j = 110; j > 0; j--);
}
void main()
{
    unsigned char m = 0xfe;           //赋初值最低位的灯亮
    while(1)                          //无限循环
    {
        P2 = m;
        delay();
        m = (m << 1) | 0x01;          //m 的值左移一位并且在最低位填 1
        if(m == 0xff) m = 0xfe;       //当点亮的灯移出最高位时,恢复初值 0xfe
    }
}

```

主程序中的指令“ $m = (m \ll 1) | 0x01$;”是让 m 的值左移一位,此时最高位移出到 CY 中,最低位移入零补位。但我们这里想让移入的位是一,所以后面加了一个“|”(位或)运算,让移位后的结果与“0x01”相或,使最低位保持为一。指令“ $\text{if}(m == 0xff) m = 0xfe$;”的表达式中用到了关系运算符“ $==$ ”(等于),即当变量 m 和 0xff 相等的条件成立时,执行语句“ $m = 0xfe$ ”。当变量 m 和 0xff 相等时,是点亮灯的位从最高位移出时,为了让灯接着循环点亮,后面的语句让点亮灯的状态回到初始状态。这里的 if 语句形式属于 3 种中的第一种。

3.2.2 switch…case 多分支选择语句

if 语句比较适合于从两者之间选择。当要实现从多种选择一种时,采用 switch…case 多分支选择语句,可使程序变得更为简洁。一般格式如下:

```

switch(表达式)
{
case 常量表达式 1:           //如果常量表达式 1 满足,则执行语句 1
    语句 1;
break;                       //执行语句 1 后,用此指令跳出 switch 结构
case 常量表达式 2:           //如果常量表达式 2 满足,则执行语句 2
    语句 2;
break;                       //执行语句 2 后,用此指令跳出 switch 结构
...
case 常量表达式 n:
    语句 n;
break;
default:                     //上述表达式都不满足时,执行语句(n+1)
    语句 n+1;
}

```

用到 switch 语句时要注意,常量表达式的值必须是整型或字符型;当满足某个常量表达式,并执行完它后面的语句时,一定不要忘了写 break 语句,否则程序就会出错。

【例 3-5】 用多分支选择语句 switch…case 实现 P2 口流水灯的控制。



例 3-5 详解

```

#include <reg51.h>
#define uchar unsigned char
#define uint unsigned int
void delay()
{
    uint i,j;
    for(i=1000;i>0;i--)
        for(j=110;j>0;j--);
}
void main()
{
    char m=3;                //m 赋初值为 3
    while(1)
    {
        switch(m--)          //表达式为 m--,第一次执行为 m 的初值
        {
            case 0:           //如果表达式的值为 0,执行这句后面的语句
                P2 = 0x01;
                break;         //执行完前一条指令,跳出 switch
            case 1:           //如果表达式的值为 1,执行这句后面的语句
                P2 = 0x02;
                break;
            case 2:           //如果表达式的值为 2,执行这句后面的语句
                P2 = 0x04;
                break;
            default:          //当表达式的值不等于 0~2,执行这句后面的语句
                P2 = 0x08;
        }
    }
}

```

```
    }  
    delay();  
    if(m<0)m=3;           //如果 m 自减 1 后的结果小于零,重新赋为初值 3  
    }  
}
```

本例先定义了一个有符号字符数 m , 并赋初值为 3。用“ $m--$ ”作为 switch 语句的表达式, 注意这里第一次执行 switch 语句时, 表达式的值为 m 的初值 3。把表达式的值与 case 语句后面的常量表达式比较, 看是否相等。因为第一次执行时, 表达式的值为 3, 不等于 0、1、2, 所以执行 default 后面的语句“ $P2=0x08$ ”, 此时 P2 口倒数第 4 个灯熄灭。延时后, 在循环语句 while(1) 的控制下, 再次进入 switch 语句, 这次表达式为 m 的初值减 1, 等于 2, 则程序跳到 case 后面值为 2 的下一条语句执行“ $P2=0x04$ ”, 此时 P2 口倒数第 3 个灯熄灭。再进入 switch 时表达式值为 1, 转去执行“case 1:”的下一条语句“ $P2=0x02$;”。再进入时, 表达式值为 0, 转去执行“case 0:”后面的语句“ $P2=0x01$;”。当 m 的值减为负值时, “if($m<0$) $m=3$;”语句控制 m 的值返回初值 3。这样, 在 switch 语句的控制下, 我们会看到学习板上 P2 口低四位的有一个灯, 在从高位到低位循环移位被熄灭。

通过这个程序的编写, 我们学习了 switch 指令的用法: 先把 switch 后面的表达式和 case 后面的常量表达式比较, 如果相等, 执行此 case 指令下面的一条指令, 碰到 break 时, 跳出 switch 语句。如果 switch 后面的表达式和 case 后面的常量表达式都不相等, 则执行 default 后面的语句。可见, switch 语句根据表达式的值, 形成了多分支选择的结构。

3.2.3 do…while 循环语句

该循环语句先执行循环体一次, 再判断表达式的值。若为真值, 则继续执行循环, 否则退出循环。一般格式如下:

```
do 循环体语句  
while(表达式);
```

do…while 循环语句的执行过程如下:

先执行一次指定的循环体语句, 然后判断表达式; 当表达式的值为非零时, 返回到第一步重新执行循环体语句; 如此反复, 直到表达式的值等于 0 时, 循环结束。

使用时要注意 while(表达式) 后的分号“;”不能丢, 它表示整个循环语句的结束。

【例 3-6】 用 do…while 循环语句实现流水灯 3 次左移循环点亮。

```
#include <reg51.h>  
#include <intrins.h>           //包含循环左移指令的头文件  
void delay()  
{  
    unsigned int i, j;  
    for(i=1000; i>0; i-- )
```



例 3-6 详解

```

for(j = 110; j > 0; j -- );
}
void main()
{
    unsigned char m, n;
    n = 2;
    do                                     //do... while 语句先执行循环体,再判断循环条件
    {
        P2 = 0xfe;                       //P2 口的初始状态,最低位的灯亮
        delay();
        for(m = 7; m > 0; m -- )         //循环左移 7 次
        {
            P2 = _crol_(P2, 1);          //循环左移语句
            delay();
        }
    }while(n -- );                       //n 减 1 不为 0 时,返回执行循环体
    while(1);                           //n 等于 0 时,跳出循环,原地踏步
}

```

上述程序中,我们控制一个流水灯从最低位开始,间隔 1s 左移点亮一次,直到最高位,共循环移位 3 次,最后停在最高位上。这里用变量 n 控制循环移位次数, n 的初值取为 2,是因为当 do...while 第一次执行完循环体时, n 取值为 2; 每执行完一次循环体, n 取值减 1,直到第 3 次执行完, n 的值减为 0,跳出循环。

3.3 C 语言的数组

数组是同类型的一组变量,引用这些变量时可用同一个标志符,借助于下标来区分各个变量。数组中的每一个变量称为数组元素。数组由连续的存储区域组成,最低地址对应于数组的第一个元素,最高地址对应于最后一个元素。数组可以是一维的,也可以是多维的。

3.3.1 一维数组

一维数组的表达式如下:

```
类型说明符  数组名  [常量];
```

方括号中的常量称为下标。C 语言中,下标是从 0 开始的。示例如下:

```
int  a[10];    //定义整型数组 a,它有 a[0]~a[9]共 10 个元素,每个元素都是整型变量
```

一维数组的赋值方法有以下几种。

(1) 在数组定义时赋值,示例如下:

```
int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

数组元素的下标从 0 开始,赋值后, $a[0]=0$, $a[1]=1$,依次类推,直至 $a[9]=9$ 。

(2) 对于一个数组也可以部分赋值,示例如下:

```
int b[10] = {0,1,2,3,4,5};
```

这里只对前 6 个元素赋值。对于没有赋值的 $b[6] \sim b[9]$,默认的初始值为 0。

(3) 如果一个数组的全部元素都已赋值,可以省去方括号中的下标,示例如下:

```
int a[] = {0,1,2,3,4,5,6,7,8,9};
```

数组元素的赋值与普通变量相同,可以把数组元素像普通变量一样使用。

3.3.2 二维数组

C 语言允许使用多维数组,最简单的多维数组是二维数组。其一般表达式形式如下:

```
类型说明符 数组名[下标 1][下标 2];
```

示例如下:

```
unsigned char x[3][4];           //定义无符号字符型二维数组,有  $3 \times 4 = 12$  个元素
```

二维数组以行列矩阵的形式存储。第一个下标代表行,第二个下标代表列。上一数组中各数组元素的顺序排列如下:

```
x[0][0]、x[0][1]、x[0][2]、x[0][3]
x[1][0]、x[1][1]、x[1][2]、x[1][3]
x[2][0]、x[2][1]、x[2][2]、x[2][3]
```

二维数组的赋值方法可以采用以下两种方式。

(1) 按存储顺序整体赋值,这是一种比较直观的赋值方式,示例如下:

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

如果是全部元素赋值,可以不指定行数,即

```
int a[][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

(2) 按每行分别赋值。为了更直观地给二维数组赋值,可以按每行分别赋值,这时要用 `{ }` 标明,没有说明的部分默认为 0,示例如下:

```
int a[3][4] = { {0,1,2,3},
                {4,5,6,7},
                {8} };           //最后 3 个元素,没有赋值的被默认为 0
```

3.3.3 字符数组

用来存放字符型数据的数组称为字符数组。与整型数组一样,字符数组也可以在定义时进行初始化赋值。示例如下:

```
char a[8] = {'B', 'e', 'i', '-', 'j', 'x', 'l', 'd'};
```

上述语句定义了字符型数组,它有 a[0]~a[7]共 8 个元素,每个元素都是字符型变量。还可以用字符串的形式来对全体字符数组元素进行赋值,示例如下:

```
char str[] = {"Now, Temperature is:"};
```

或者写成更简洁的形式:

```
char str[] = "Now, Temperature is:";
```

要特别注意的是:字符串是以'\0'作为结束标志的。所以,当把一个字符串存入数组时,也把结束标志'\0'存入了数组。因此,上面定义的字符数组“str[20]”最后一个元素不是“:”,而是'\0'。数组必须先定义,才能使用。

3.3.4 数组的应用

流水灯的控制方法有许多种,一种比较常用的方法是:把流水灯的控制代码按顺序存入数组,再依次引用数组元素,并送到发光二极管的接口显示。无论流水灯的控制逻辑多么复杂,用这种方法都可以很容易地通过调用控制代码实现。

例如,定义一个无符号字符型数组如下:

```
unsigned char code Tab[] = {0x7f, 0xbf, 0xdf, 0xef, 0xf7, 0xfb, 0xfd, 0xfe};
```

上述数组定义中用到了关键字“code”,因为数组的各个元素在使用过程中不发生变化,所以可以用这个关键字定义数组元素的存放方式,减小数组数据的存储空间。假设这个数组中的元素是要送给 P2 口的控制代码,程序如果把它们按顺序送给 P2 口,并间隔一定的延时,就可以实现一个流水灯的右移点亮。

【例 3-7】 用数组控制 P2 口一个流水灯间隔 1s 右移点亮。

```
#include <reg51.h>
#define uchar unsigned char
#define uint unsigned int
uchar code tab[] = {0x7f, 0xbf, 0xdf, 0xef, 0xf7, 0xfb, 0xfd, 0xfe}; //定义一个数组
uchar i = 0;
void delay()
```



例 3-7 详解

```

{
    uint m,n;
    for(m=1000;m>0;m--)
        for(n=110;n>0;n--);
}
void main()
{
    for(;i<8;i++)           //循环给 P2 口送数 8 次
    {
        P2 = tab[i];        //引用数组元素,并送 P2 口显示
        delay();            //调用延时 1s 子程序
    }
    while(1);               //显示结束,程序停在这里
}

```

本例程序中我们可以看到数组元素的用法：在使用数组之前要先定义一个数组，包含数组元素类型、数组名和具体的数组元素，这里定义的数组是无符号字符型的，数组中的元素都是要给 P2 口的控制代码，并且是按顺序存放的。在程序中引用数组时用到语句“P2=tab[i];”，把数组中的第 i 个元素取出，直接赋值给 P2 显示。并且这个语句又是 for 循环内部语句，for 循环控制取数组元素从 tab[0]直到 tab[7]。其中，for 循环中第一个表达式省略，因为程序前面在定义变量 i 时，已经给 i 赋了初值 0。这个程序里，我们可以试着把数组元素改成其他形式，就可以形成不同逻辑的流水灯控制，所以数组的应用在花式流水灯控制上，是一种非常高效的编程控制方法。

另外，通过这个例子我们还可以明确局部变量和全局变量的概念。在函数内部定义的变量称为局部变量，如本例中延时函数内部定义的变量 m、n，就是延时函数的局部变量。局部变量只在该函数内有效。例如，一个函数定义了变量“x”为整型数据，另一个函数则把变量“x”定义为字型数据，两者之间互不影响。全局变量也称为外部变量，它定义在函数的外部，最好在程序的顶部。它的有效范围为从定义开始的位置到源文件结束。例如，本例中在程序最开始定义的语句“uchar i=0;”，就定义了一个全局变量 i，从这条语句开始以下的程序中都可以使用此变量。全局变量可以被函数内的任何表达式访问。如果全局变量和某一函数的局部变量同名时，在该函数内，只有局部变量被引用，全局变量被自动“屏蔽”。例如，我们在主函数中用同样的语句再定义一个变量 i 时，此时在主函数中只有主函数定义的内部变量 i 是有效的。

3.3.5 数组作为函数参数

一个数组的名字表示该数组的首地址，所以用数组名作为函数的参数时，被传递的就是数组的首地址，被调用的函数的形式参数必须定义为指针型变量。

用数组名作为函数的参数时，应该在主调函数和被调函数中分别进行数组定义，而不能只在一方定义数组。并且，两个函数中定义的数组类型必须一致。如果不一致，将导致编译出错。实参数组和形参数组的长度可以一致，也可以不一致，编译器不检查形参数组的长

度,只是将实参数组的首地址传递给形参数组。为保证两者长度一致,最好在定义形参数组时,不指定长度,只在数组名后面跟一个空的方括号[]。编译时,系统会根据实参数组的长度为形参数组自动分配长度。

【例 3-8】 使用数组作参数控制 P2 口八位流水灯点亮。

先定义流水灯控制码数组,再定义流水灯点亮函数,使其形参为数组,数据类型要和实参数组的类型一致。



例 3-8 详解

```
#include <reg51.h>
void delay()
{
    unsigned int i, j;
    for(i = 1000; i > 0; i--)
        for(j = 110; j > 0; j--);
}
void xianshi(unsigned char a[]) //定义显示子函数,形参为字符型数组首地址
{
    unsigned char m;
    for(m = 0; m < 8; m++)
    {
        P2 = a[m]; //取数组的第 m 个元素送 P2 口显示
        delay();
    }
}
void main()
{
    unsigned char code tab[] = {0x7f, 0xbf, 0xdf, 0xef, 0xf7, 0xfb, 0xfd, 0xfe}; //定义流水灯控制码
    while(1)
    {
        xianshi(tab); //调用显示子函数
    }
}
```

本例中先定义了一个显示子函数,子函数的形参是无符号字符型数组 tab 的首地址,该子函数的功能是顺序地取数组中的元素,送 P2 口显示。在主函数中,先定义一个流水灯控制码数组,该数组类型要和显示子函数形参类型相同,这里都是无符号字符型。数组中的元素是要送到 P2 口的显示代码,顺序把它们送显示,可以实现 P2 口一个灯右移循环点亮。然后在主函数中调用刚才定义的显示函数,这时要注意,调用时函数的实参是刚才定义的控制码数组名。

3.4 C 语言的指针

指针是 C 语言中的一个重要概念,也是 C 语言的一个重要特色。正确而灵活地运用指针,可以有效地表示复杂的数据结构,动态地分配内存,方便地使用字符串,有效地使用数组。利用指针引用数组元素速度更快,占用内存更少。

3.4.1 指针的定义和引用

1. 指针的概念

一个数据的“指针”就是它的地址。通过变量的地址能找到该变量在内存中的存储单元,从而能得到它的值。指针是一种特殊类型的变量。它具有一般变量的三要素:名字、类型和值。指针的命名与一般变量是相同的,它与一般变量的区别在于值和类型上。

1) 指针的值

指针存放的是某个变量在内存中的地址值。被定义过的变量都有一个内存地址。如果一个指针存放了某个变量的地址值,就称这个指针指向该变量。由此可见,指针本身具有一个内存地址。另外,它还存放了它所指向的变量的地址值。

2) 指针的类型

指针的类型就是该指针所指向的变量的类型。例如,一个指针指向 int 型变量,该指针就是 int 型指针。

3) 指针的定义格式

指针变量不同于整型或字符型等其他类型的数据,使用前必须将其定义为“指针类型”。指针定义的一般形式如下:

类型说明符 * 指针名字

示例:

```
int i;           //定义一个整型变量 i
int * pointer;   //定义整型指针,名字为 pointer
```

可以用取地址运算符“&”使一个指针变量指向一个变量,例如:

```
pointer = &i;    // "&i"表示取 i 的地址,将 i 的地址存放在指针变量 pointer 中
```

在定义指针时要注意两点:

- (1) 指针名字前的“*”表示该变量为指针变量。
- (2) 一个指针变量只能指向同一个类型的变量,如整型指针不能指向字符型变量。

2. 指针的初始化

在使用指针前必须进行初始化,一般格式如下:

类型说明符 指针变量 = 初始地址值;

示例:

```
unsigned char * p;    //定义无符号字符型指针变量 p
unsigned char m;      //定义无符号字符型数据 m
p = &m;               //将 m 的地址存在 p 中(指针变量 p 被初始化了)
```

严禁使用未经初始化的指针变量,否则将引起严重后果。

3. 指针数组

指针可以指向某类变量,也可以指向数组。以指针变量为元素的数组称为指针数组。这些指针变量应具有相同的存储类型,并且指向的数据类型也必须相同。

指针数组定义的一般格式如下:

```
类型说明符 * 指针数组名[元素个数];
```

示例:

```
int * p[2];          //p[2]是含有 p[0]和 p[1]两个指针的指针数组,指向 int 型数据
```

指针数组的初始化可以在定义时同时进行,示例如下:

```
unsigned char a[ ] = {0,1,2,3};
unsigned char * p[4] = {&a[0],&a[1],&a[2],&a[3]}; //存放的元素必须为地址
```

4. 指向数组的指针

一个变量有地址,一个数组元素也有地址,所以可以用一个指针指向一个数组元素。如果一个指针存放了某数组的第一个元素的地址,就说该指针是指向这一数组的指针。数组的指针即数组的起始地址。示例如下:

```
unsigned char a[ ] = {0,1,2,3};
unsigned char * p;
p = &a[0];          //将数组 a 的首地址存放在指针变量 p 中
```

经上述定义后,指针 p 就是数组 a 的指针。

C 语言规定:数组名代表数组的首地址,也就是第一个元素的地址。例如,下面两个语句等价:

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

C 语言规定: p 指向数组 a 的首地址后, p+1 就指向数组的第二个元素 a[1], p+2 指向 a[2],依次类推, p+i 指向 a[i]。

引用数组元素可以用下标(如 a[3]),但使用指针速度更快,且占用内存少。这正是使用指针的优点和 C 语言的精华所在。

对于二维数组, C 语言规定:如果指针 p 指向该二维数组的首地址(可以用 a 表示,也可以用 &a[0][0]表示),那么 p[i]+j 指向的元素就是 a[i][j]。这里 i、j 分别表示二维数组的第 i 行和第 j 列。

3.4.2 指针的应用

【例 3-9】 用指针数组控制 P2 口八位流水灯点亮。

指针数组中元素是变量的地址,在本例中就应该流水灯控制码的地址。可先定义流水灯的控制码数组为:

```
unsigned char code Tab[ ] = {0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
```

然后将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]};
```

最后利用指针运算符“*”取得各指针所指元素的值,并送入 P2 口显示即可。程序如下:

```
#include <reg51.h>
#define uchar unsigned char
#define uint unsigned int
void delay()
{
    uint i,j;
    for(i=0;i<1000;i++)
        for(j=0;j<110;j++);
}
void main()
{
    uchar code tab[ ] = {0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
    uchar * p[ ] = {&tab[0],&tab[1],&tab[2],&tab[3],&tab[4],&tab[5],&tab[6],&tab[7]};
    uchar m = 0;
    for(;m<8;m++)          //循环控制取数组元素 8 次
    {
        P2 = * p[m];        //将指针数组中第 i 个元素送 P2 口
        delay();
    }
    while(1);              //取数完毕程序停在这里
}
```

本例采用指针数组来控制 P2 口流水灯,以下再看一个用指向数组的指针来控制流水灯的例子。这里同样要定义流水灯控制码数组,再将数组名(数组的首地址)赋给指针。然后即可通过指针引用数组的元素,从而控制八位流水灯点亮。引用指针时要注意和上例的区别。

【例 3-10】 用指向数组的指针来控制流水灯。

```

#include <reg51.h>
void delay()
{
    unsigned int i, j;
    for(i = 1000; i > 0; i--)
        for(j = 110; j > 0; j--);
}
void main()
{
    unsigned char code Tab[] = {0x7f, 0xbf, 0xdf, 0xef, 0xf7, 0xfb, 0xfd, 0xfe}; //流水灯控制码数组
    unsigned char * p, m; //定义无符号字符型指针和控制循环的变量
    p = Tab; //指针指向数组首地址
    for(m = 0; m < 8; m++) //循环显示 8 个状态
    {
        P2 = * (p + m); //通过指针引用数组元素, 送到 P2 显示
        delay();
    }
    while(1); //显示完毕, 程序停在这里
}

```

我们可以从以上两个例子比较一下两种用法的区别。指针数组需要先将指针数组元素都赋值(初始化)再使用, 通过指针取数组元素的方法是“* p[m]”(其中 m 是数组元素下标)。而指向数组的指针使用前要先定义一个指针, 并将数组的首地址给指针, 取数组元素的方法是“* (p+m)”。因此, 这两种方法使用时是有一定的区别的, 一定要注意区分。

3.4.3 指针作函数参数的应用

函数的参数不仅可以是数据, 也可以是指针, 它的作用是将一个变量的地址传送到另一个函数中。

【例 3-11】 用指针作函数参数控制 P2 口八位流水灯点亮。

首先, 定义一个指针指向存储流水控制码的数组的首地址, 然后以这个指针作为实际参数传递给被调函数的形参, 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的方法就可以控制 P2 口八位流水灯点亮。

```

#include <reg51.h>
void delay()
{
    unsigned int i, j;
    for(i = 1000; i > 0; i--)
        for(j = 110; j > 0; j--);
}
void xianshi(unsigned char * p) //显示子程序, 形参为无符号字符型指针
{

```

```

unsigned char i;
while(1)                                //无限循环
{
    i = 0;
    while( * (p + i) != '\0')           //当数组中元素未取完时,接着取数送显示
    {
        P2 = * (p + i);                 //取数组中第 i + 1 个元素送 P2 口显示
        delay();                        //修改循环计数变量
        i++;
    }
}
void main()
{
    unsigned char code tab[] = {0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
                                     //定义显示码数组

    unsigned char * pin;
    pin = tab;                         //指针指向数组首地址
    xianshi(pin);                      //调用显示子程序
}

```

此程序中首先定义了一个显示子程序,它的形参为无符号字符型指针。子程序的功能为:循环取数组中的元素,并送 P2 口显示。如果把指针 p 指向数组的首地址,则 p+i 指向的是数组中的第 i+1 个元素,* (p+i)则表示取数组中第 i+1 个元素的值。程序中“'\0'”表示数组元素的结束标志,数组元素在存储的时候,不仅要存储各个元素值,在最后还要存储结束标志。如果读数组元素时,读到结束标志就表示全部元素已经读完。所以指令行 while(* (p+i) != '\0')的功能是:当读数组元素不是结束标志时,循环继续,即接着取数并送显示。在本例主程序中,首先定义了显示控制码数组,该数组元素按顺序取数时,可以实现一个流水灯向左移位循环点亮。主程序中又定义了和数组同类型的指针 pin,并让该指针指向数组首地址,再调用刚定义的显示子程序时,就可以用指针 pin 作为它的实参。

3.4.4 函数型指针的应用

在 C 语言中,指针变量除了能指向数据对象外,也可以指向函数。一个函数在编译时,分配了一个入口地址,这个入口地址就称为函数的指针。可以用一个指针变量指向函数的入口地址,然后通过该指针变量调用此函数。

定义指向函数的指针变量的一般形式如下:

类型说明符 (* 指针变量名)(形参列表)

函数的调用可以通过函数名调用,也可以通过函数指针来调用。要通过函数指针调用函数,只要把函数的名字赋给该指针就可以了。

【例 3-12】 用函数型指针控制 P2 口八位流水灯点亮。

先定义流水灯点亮函数,再定义函数型指针,然后将流水灯点亮函数的名字(入口地址)

赋给函数型指针,就可以通过该函数型指针调用流水灯点亮函数。注意:函数型指针的类型说明符必须和函数的类型说明符一致。

```
#include <reg51.h>
unsigned char code tab[] = {0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f}; //定义显示码数组
void delay()
{
    unsigned int i,j;
    for(i=1000;i>0;i--)
        for(j=110;j>0;j--);
}
void xianshi()                //定义显示子程序
{
    unsigned char i;
    for(i=0;i<8;i++)
    {
        P2 = tab[i];          //取数组中的第 i + 1 个元素送 P2 口
        delay();
    }
}
void main()
{
    void( *p)(void);          //定义函数型指针 p
    p = xianshi;               //将函数入口地址赋给指针 p
    while(1)( *p)();          //通过指针 p 调用显示函数,并循环
}
```

本例中定义的显示子程序没有什么特别的地方,就是顺序取数组中的元素,并送 P2 口显示。在主程序中,需要先定义一个函数型指针,因为该指针所指向的函数没有参数和返回值,所以该指针的参数和类型说明符都为空。再用指令“p=xianshi;”把显示子程序名赋给指针,使指针指向函数入口地址。最后用指令“while(1)(*p)();”循环调用显示子程序,其中,“(*p)();”是循环内部语句,功能是通过指针 p 调用所指向的函数。

3.5 C 语言的编译预处理

编译预处理是 C 语言编译器的一个组成部分。在 C 语言中,通过一些预处理命令可以在很大程度上为 C 语言本身提供许多功能和符号等方面的扩充,增强 C 语言的灵活性和方便性。预处理命令可以在编写程序时加在需要的地方,但它只在程序编译时起作用,并且通常是按行进行处理的,因此又称为**编译控制行**。编译器在对整个程序进行编译之前,先对程序中的编译控制进行预处理,然后在将预处理的结果与整个 C 语言源程序一起进行编译,以产生目标代码。常用的预处理命令有宏定义、文件包含和条件编译。为了与一般 C 语言语句区别,预处理命令以“#”开头。

1. 宏定义

C 语言允许用一个标志符来表示一个字符串,称为宏。被定义为宏的标志符为宏名。

在编译预处理时,程序中的所有宏名都用宏定义中的字符串代替,这个过程称为宏代换。宏定义分为不带参数的宏定义和带参数的宏定义。

(1) 不带参数的宏定义的一般形式如下:

```
#define 标志符 字符串
```

这种用法在前面的程序里已经用过了。如:

```
#define uchar unsigned char
```

对于不带参数的宏定义说明如下:

- ① 宏定义不是 C 语句,不能在行末加分号,如果加了会连分号一起替代。
- ② 宏名的有效范围为定义命令之后到本源文件结束。通常, #define 命令写在文件开头,在函数之前,作为文件的一部分,在此文件范围内有效。
- ③ 可以用 #undef 命令终止宏定义的作用域,在该语句之后,原来定义的宏将不起作用。

(2) 带参数的宏定义不是进行简单的字符串替换,还要进行参数替换,其一般形式如下:

```
#define 宏名(参数表) 字符串
```

字符串中包含在括号中所指定的参数,示例如下:

```
#define PI 3.1415926
#define S(r) PI * (r) * (r)
main()
{
    float a,area;
    a = 5.6;
    area = S(a);
}
```

经预处理后,程序在编译时如果遇到带参数的宏,则按照指定的字符串从左到右进行置换。关于带参数的宏定义说明如下:

宏定义如果写成 #define S(r) PI * r * r 可能引发歧义。如果参数不是 r 而是 a+b 时,S(a+b)将被替换为 PI * a+b * a+b,这显然与编程的意图不一致。为此,应当在定义时在字符串中的形参外面加上一个括号,即 #define S(r) PI * (r) * (r)。宏名与参数表之间不能有空格,否则将空格以后的字符都作为替代字符串的一部分。

2. 文件包含

文件包含是指一个程序将另一个指定的文件的全部内容包含进来。文件包含的命令一般格式如下:

```
#include <文件名>
```

文件包含命令的功能是用指定文件的全部内容替换该预处理行。例如,在每个程序的最开始都要写上一行“#include <reg51.h>”,就是在此行用 reg51.h 头文件替换该行,因为在后面的程序中要用到该头文件中定义的内容。在进行较大规模程序设计时,文件包含命令十分有用。为了使用模块化编程,可以将组成 C 语言程序的各个功能函数分散到多个程序文件中,分别由若干人员完成,最后再用 #include 命令将它们嵌入一个总的程序文件中。需要注意的是:一个文件包含命令只能指定一个被包含文件。如果程序中要包含多个文件,则需要使用多个包含命令。当程序中需要调用 C51 编译器提供的各种库函数时,必须在程序的开头使用 #include 命令将相应的函数说明文件包含进来。

3. 条件编译

一般情况下,对 C 语言程序进行编译时,所有的程序都参加编译,但有时希望对其中一部分内容只在满足一定条件时才进行编译,这就是所谓的条件编译。条件编译可以选择不同的编译范围,从而产生不同的代码。C51 编译器的预处理提供的条件编译命令可以分为以下 3 种形式。

1) 形式一

```
# ifdef 标志符
    程序段 1
# else
    程序段 2
# end if
```

如果指定的标志符已被定义,则程序段 1 参加编译,并产生有效代码,而忽略掉程序段 2; 否则,程序段 2 参加编译并产生有效代码,而忽略掉程序段 1。

2) 形式二

```
# if 常量表达式
    程序段 1
# else
    程序段 2
# end if
```

如果常量表达式为“真”,那么就编译该语句后的程序段。

3) 形式三

```
# ifndef 标志符
    程序段 1
# else
    程序段 2
# end if
```

该形式编译命令的格式与第一种命令格式只有第一行不同,它的作用与第一种编译命

令的作用刚好相反,即如果标志符还没有被定义,那么就编译该语句后的程序段。

【例 3-13】 用带参数的宏定义完成运算 $a * b / (a + b)$, 将结果送 P2 口显示。

本例可以用带参数的宏定义如下:

```
#define F(a,b) (a) * (b) / ((a) + (b))
```

注意: 在字符串中的形式参数外面加上一个括号,可以避免编译时的歧义;宏名 F 与带参数的括号之间不应加空格;带参数的宏和函数不同,函数是先求出实参数表达式的值,然后代入形参。而带参数的宏只是进行简单的字符替换。

```
#include <reg51.h>
#define F(a,b) (a) * (b) / ((a) + (b))
void main()
{
    int i,j,k;
    i = 34;
    j = 45;
    k = 30;
    P2 = F(i + j,k);
    while(1);
}
```

本例中定义的宏 F 其中的形参 a、b 分别被实参 i+j、k 代替。将 i、j、k 的值代入,公式的计算结果为 21.743,计算结果自动舍去小数点后的值,送到 P2 口的值为十进制的 21,展成二进制数为 0001 0101B,这个数送 P2 口,我们可以看到:送“1”的位二极管熄灭,送“0”的位二极管点亮。

【例 3-14】 使用条件编译控制 P2 口点亮灯的状态。

通过本例掌握条件编译的使用方法。要求某条件满足时,P2 口低四位灯点亮;若不足,则高四位灯点亮。

```
#include <reg51.h>
#define max 100
void main()
{
    #if max > 80          //当 max > 80 时,0xf0 送 P2 口
        P2 = 0xf0;
    #else
        P2 = 0x0f;        //当 max ≤ 80 时,0x0f 送 P2 口
    #endif
}
```

本例根据常量表达式“max > 80”的值是否为真来控制编译,因为前面已经用“# define

max 100”定义了宏名 max 来表示 100,所以常量表达式的值为真,执行“P2=0xf0;”,再用“#endif”命令结束本次条件编译。使用这种格式,需要事先给定一个条件,使程序在不同的条件下完成不同的功能。

习题

(1) 用 C51 编程实现除法“ $90 \div 8$ ”的运算,并通过 P2 口的发光二极管分时显示结果的商和余数。

(2) 编程实现:设初始状态为 P2 口 8 个灯全亮,用 if 语句控制 P2 口流水灯从高位到低位顺序熄灭。

(3) 编程实现:用多分支选择语句 switch...case 实现 P2 口流水灯从高位到低位点亮。

本章小结

本章内容是对 C51 主要知识点(包括运算符、控制语句、数组、指针、预处理)的完整总结归纳,因为涉及的内容较多,读者全面掌握有一定难度。但实际上,学习单片机 C 语言是为了灵活应用,不需要把这些知识点都一一牢记,只需对它们都有初步的印象,在需要时知道怎么用就可以了。另外,也可以把本章内容作为参考,在后面具体软件设计中遇到问题时,及时查阅本章知识,以对知识点内容融会贯通。本章采用边学边练的方法,所有程序都是针对学习板上流水灯的控制。