第3章

C51语言基础

【学习指南】

通过本章的学习，首先要掌握C51特定的数据类型应用方法，掌握变量存储类型的应用，了解变量的存储模式；运算符和表达式是C语言基础内容，需要熟练掌握；if、for、while等语句表达式也是C语言基础内容，同样非常重要，也需要熟练掌握；了解库函数、中断、子函数等各类函数的应用。本章与C语言衔接，并引入单片机的控制方法，是C51软件的基础，一定要吃透相关知识，否则无法熟练进行后面的编程。

对于单片机应用系统，除了硬件电路外，还需要软件系统的配合。MCS-51单片机有两种开发语言，即汇编语言和C51语言。汇编语言控制精确，效率高，但可读性差，编程难度较大；C51语言是由C语言继承而来的，与C语言不同的是，C51语言运行于单片机平台，而C语言则运行于普通的桌面平台。C51语言继承了C语言的优点，可读性和可移植性强，且结合了对单片机直接控制的特点，已被开发人员广泛接受。

3.1 C51语言基础

C51语言除了有ANSI C的所有标准数据类型外，还加入了一些与MCS-51单片机密切相关的特殊数据类型。对于具备C语言基础的初学者，要重点学习C51语言特定的数据类型，这些数据类型是单片机和C语言的桥梁，是C语言控制单片机的基础。

3.1.1 数据类型

C51语言的数据类型较多，常用的包括unsigned char、unsigned int、sbit等，需要理解这些数据类型的定义和应用场合。表3-1为C51语言的数据类型。

表 3-1　C51 语言的数据类型

数据类型	说　明	长　度	值　域
unsigned char	无符号字符型	单字节	0～255
signed char	带符号字符型	单字节	－128～＋127
unsigned int	无符号整型	双字节	0～65535
signed int	带符号整型	双字节	－32768～＋32767
unsigned long	无符号长整型	四字节	0～4294967295
signed long	带符号长整型	四字节	－2147483648～＋2147483647
float	单精度型	四字节	±1.175494E－38～±3.402823E＋38
*	指针	1～3 字节	对象的地址
bit	位变量	位	0 或 1
sfr	8 位特殊功能寄存器	单字节	0～255
sfr16	16 位特殊功能寄存器	双字节	0～65535
sbit	位寻址定义	位	0 或 1

提示：表中的无符号字符型(unsigned char)和无符号整型(unsigned int)的定义和值域经常用到，要记住。

其中，bit、sfr、sfr16 和 sbit 是 C51 语言中特殊的变量类型，下面进行详细介绍。

1. bit 位变量

bit 位变量是 C51 编译器的一种扩充数据类型。利用它可定义位变量，但不能定义位指针，也不能定义位数组。它的值是一个二进制数：0 或 1，应用在单片机存储器的 0x20～0x2F 区域。

2. sfr 特殊功能寄存器

sfr 用来定义 8 位特殊功能寄存器，占用一个内存单元地址，值域为 0～255(0x80～0xFF)。sfr 是 C51 语言非常重要的关键字，通过 sfr 可直接访问 MCS-51 单片机内部的所有特殊功能寄存器。

其用法：

```
sfr 特殊功能寄存器名 = 特殊功能寄存器地址常数;
```

如：

```
sfr P1 = 0x90;      /* 定义 P1 口,其地址 90H */
P1 = 0xFF;          /* 把 FFH 送入 P1 中(对 P1 口的所有引脚置高电平) */
```

3. sfr16 为 16 位特殊功能寄存器

sfr16 用来定义 16 位特殊功能寄存器，占用两字节。

其用法：

```
sfr16 特殊功能寄存器名 = 特殊功能寄存器地址常数;
```

如：8052 的 T2 定时器，可以定义为

```
sfr16 T2 = 0xCC;    /* 定义 8052 定时器 2,地址为 T2L = CCH,T2H = CDH */
```

用 sfr16 定义 16 位特殊功能寄存器时，等号后面是它的低位地址，高位地址一定要位

于物理低位地址之上。

4. sbit 位寻址定义

sbit 是一种非常重要且常用的特殊数据类型。sbit 定义位寻址对象，访问特殊功能寄存器的某位。

sbit 的用法有三种：

(1) sbit 位变量名＝位地址，例如：

```
sbit P1_1 = 0x91;
```

(2) sbit 位变量名＝特殊功能寄存器名 ^位序号，例如：

```
sfr P1 = 0x90;
sbit P1_1 = P1^ 1;/ * P1_1 为 P1 口的 P1.1 引脚 * /
```

(3) sbit 位变量名＝字节地址 ^位序号，例如：

```
sbit P1_1 = 0x90 ^ 1
```

3.1.2 存储类型

说明了一个变量数据类型后，还可选择说明该变量的存储器类型，指定该变量在 C51 硬件系统中所使用的存储区域，并在编译时准确地定位。

表 3-2 列出的存储器类型和存储器的应用息息相关，特别是单片机的内存相对较小，需要合理分配存储空间、提高运行速度时，必须透彻理解存储器的应用场合。下面详细介绍各种存储器的特点。

表 3-2 C51 编译器存储器类型

存储器类型	说　明	地　址
data	内部数据存储器(128 字节)，访问速度最快	0x00～0x7F
bdata	位/字节寻址内部数据存储器(16 字节)	0x20～0x2F
idata	内部数据存储器(256 字节)	0x00～0xFF
pdata	外部数据存储器(256 字节)	0x00～0xFF
xdata	外部数据存储器(64KB)	0x0000～0xFFFF
code	程序存储器(64KB)	0x000～0xFFFFH

1. data 区

data 区的寻址是最快的，应该把使用频率高的变量放在 data 区，由于空间有限(128B)，注意节约使用。

data 区的声明如下：

```
unsigned char data ar1;
unsigned int data bar[2];
```

2. bdata 区

位寻址的数据存储区位于 0x20～0x2F，可将要求位寻址的数据定义为 bdata。如：

```
unsigned char bdata ibr;    / * 在位寻址区定义 unsigned char 类型的变量 ibr * /
```

```
int bdata ab[2];          /* 在位寻址区定义数组 ab[2],这些也称为可寻址位对象 */
```

如：

```
bit ibr7 = ibr^7;         /* 访问位寻址对象其中一位 */
bit ab12 = ab[1]^12;      /* 操作符"^"后面的位置最大值取决于指定的基址类型,比如:
                          char(0-7),int(0-15),long(0-31) */
```

3. idata 区

idata 区也可以存放使用比较频繁的变量。与外部存储器寻址比较，它的指令执行周期和代码都比较短。例如：

```
unsigned char idata st = 0;
char idata su;
```

4. pdata 和 xdata 区

pdata 和 xdata 都定义在外部存储器区域，pdata 区只有 256 字节，而 xdata 可达 65536 字节。例如：

```
unsigned char pdata pd;
unsigned int xdata px;
```

pdata 区的寻址要比 xdata 区寻址快，因为 pdata 区寻址只需要装入 8 位地址，而 xdata 区需要装入 16 位地址。

5. code 区

code 定义在单片机的程序存储器区，数据不可以改变。code 区可以存放数据表、跳转向量或状态表，code 区在编译时要初始化。由于 MCS-51 单片机的数据存储器空间有限，而程序存储器空间比较充裕，可以把一些不发生变化的数据放在程序存储器。例如：

```
unsigned char code data[8] = {0x01,0x02,0x04,0x00,0x06,0x21,0x54,0x32};
```

把上述几种存储器类型的特点和地址空间进行归纳，如表 3-3 所示。

表 3-3 存储器类型特点和地址空间

存储器		地址空间	容量	C51 编译器中变(常)量存储器类型	汇编语言中的寻址方式	访问速度
内部数据区	工作寄存器区	0x00～0x1F	32	data,idata	寄存器寻址	最快
	位地址区	0x20～0x2F	16	bdata,data	位寻址、直接寻址	快
	数据缓冲区	0x30～0x7F	80	data,idata	直接寻址、寄存器间接寻址	data 快 idata 中
		0x80～0xFF	128	idata	寄存器间接寻址	中
	特殊功能寄存器区	0x80～0xFF	128	—	直接寻址	快
内(外)部程序存储区		0x0000～0xFFFF	65536	code	变址间接寻址	最慢
外部数据存储区		0x0000～0xFFFF	65536	xdata、pdata(0x00～0xFF)	寄存器间接寻址	pdata 慢 xdata 最慢

提示：变量存储类型关系到变量在存储器中的地址分配，在C51语言中占有重要位置，要好好理解，看懂并能够记住表3-3中的地址空间和存储器类型的关系。

3.1.3 存储模式

C51编译器允许采用三种存储模式：小编译模式（SMALL）、紧凑编译模式（COMPACT）、大编译模式（LARGE）。存储模式用于决定未标明存储器类型变量的默认存储器类型。存储模式如表3-4所示。

表3-4 存储模式

存储模式	说　明
SMALL	默认的存储类型是data，参数及局部变量放入可直接寻址片内RAM的用户区中（最大128字节）。另外，所有对象（包括堆栈）都必须嵌入片内RAM。栈长很关键，因为实际栈长依赖于函数嵌套调用层数
COMPACT	默认的存储类型是pdata，参数及局部变量放入分页的外部数据存储区，栈空间位于片内数据存储区中
LARGE	默认的存储类型是xdata，参数及局部变量直接放入片外数据存储区。用此数据指针进行访问效率较低，尤其对两个或多个字节的变量，这种数据类型的访问机制直接影响代码的长度

3.1.4 绝对地址访问

1. 绝对宏

C51编译器提供了一组宏定义对单片机的data区、pdata区和xdata区、code区等不同的存储区域进行绝对地址的访问。在程序中，用“#include <absacc.h>”即可使用声明的宏来访问绝对地址，包括CBYTE、DBYTE、PBYTE、XBYTE、CWORD、DWORD、PWORD和XWORD等。

CBYTE以字节方式寻址code区；

CWORD以字方式寻址code区；

DBYTE以字节方式寻址data区；

DWORD以字方式寻址data区；

PBYTE以字节方式寻址pdata区；

PWORD以字方式寻址pdata区；

XBYTE以字节方式寻址xdata区；

XWORD以字方式寻址xdata区。

如：包含头文件#include <absacc.h>后，通过DBYTE、XBYTE、CBYTE等可访问绝对地址：

```
xvar = XBYTE[0x2000];//把外部数据存储器 0x2000 单元的一个字节数据送到变量 xvar 中;
XBYTE[0x1F00] = 0xf0;//向外部数据存储器 0x1F00 单元写入数据 0xf0;
```

2. _at_关键字

采用_at_关键字可以指定变量在存储空间中的绝对地址，一般格式如下：

```
数据类型 [存储器类型] 标识符 _at_ 地址常数
```

_at_关键字用法比较简单,但需注意以下几点:

(1) 不能初始化;

(2) bit 型变量不能被_at_指定;

(3) _at_定义的变量必须是全局变量,不能放在主程序或函数中,否则编译出错。如:

```
unsigned char data ur _at_ 0x20; //ur 变量的地址为内部数据存储区 0x20
```

3. 指针

用指针进行绝对地址的访问,更加灵活、简单。定义一个指针变量,把地址赋予绝对地址,就可以访问该变量了。

如:

```
unsigned char data * p;    //定义一个指针,指定在 data 区
p = 0x20;                  //赋地址给指定指针
* p = 0x38                 //把 0x38 送给内部数据存储器 0x20 单元
```

提示:以上三种绝对地址访问方式比较重要,特别是指针的访问方式,应用很广,要掌握好。

3.2　C51 预处理

预处理功能包括宏定义、文件包含和条件编译 3 个主要部分。预处理命令不同于 C 语言语句,具有以下特点:

(1) 预处理命令以“#”开头,后面不加分号;

(2) 预处理命令在编译前执行;

(3) 多数预处理命令习惯放在文件的开头。

1. 宏定义

不带参数宏定义的格式为:

```
#define  新名称  原内容
```

如:#define uchar unsigned char

该指令的作用是用#define 后面的第一个字母组合代替该字母后面的所有内容。

如:#define PI 3.14,以后在程序中用 PI 代替 3.14。

2. 包含文件

包含文件的含义是在一个程序文件中包含其他文件的内容。用文件包含命令可以实现文件包含功能,命令格式为:

```
#include <文件名>或 #include "文件名"
```

例如,在文件中第一句经常为#include < reg51.h >,在编译预处理时,对#include 命令进行文件包含处理。实际上就是将文件 reg51.h 中的全部内容复制插入#include < reg51.h >

的命令处。

3. 条件编译

提供一种在编译过程中根据所求条件的值有选择地包含不同代码的手段，实现对程序源代码的各部分有选择地进行编译，称为条件编译。

#if 语句中包含一个常量表达式，若该表达式求值的结果不等于 0，则执行其后的各行，直到遇到 #endif、#elif 或 #else 语句为止（预处理 elif 相当于 else if）。在 #if 语句中可以使用一个特殊的表达式 defined（标识符）：当标识符已经定义时，其值为 1；否则，其值为 0。

例如，为了保证 hdr.h 文件的内容只被包含一次，可用条件语句把该文件的内容包含起来：

```
#ifndef(hdr)
#define hdr
#include(hdr.h)
#endif
```

3.3 运算符与表达式

C 语言有丰富的运算符，绝大多数操作都可以通过运算符来处理。运算符就是完成某种特定运算的符号，包括算术运算符、赋值运算符、关系运算符、逻辑运算符、位运算符、条件运算符等。按照表达式中运算对象的个数又可将运算符分为单目运算符、双目运算符和三目运算符。单目运算符只需一个运算对象，双目运算符要求有两个运算对象，三目运算符则要求有三个运算对象。

表达式是由运算符和运算对象所组成的具有特定含义的式子。运算符和表达式可以组成 C 语言程序的各种语句。

1. 算术运算符

算术运算符包括以下几种：

+　　加或取正值运算符

−　　减或取负值运算符

*　　乘运算符

/　　除运算符

%　　取余运算符

++　自增运算符

−−　自减运算符

注：自增、自减运算符的作用是使变量的值加 1 或减 1。

++i　先使 i 值加 1，然后再使用；

−−i　先使 i 值减 1，然后再使用；

i++　使用完 i 的值以后，再使 i 值加 1；

i−−　使用完 i 的值以后，再使 i 值减 1。

2. 赋值运算符

“＝”就是赋值运算符，其功能是将一个数据赋给一个变量。例如：

```
a = 8;
b = 5;
c = a/b;
a = b = 6;
```

以上语句执行时，先计算出右边表达式的值，再将该值赋给左边的变量。

3. 关系运算符

关系运算符的功能是判断两个数的关系。C 语言有以下六种关系运算符：

＞　　大于

＜　　小于

＞＝　大于或等于

＜＝　小于或等于

＝＝　测试等于

！＝　测试不等于

关系运算符的优先级低于算术运算符，高于赋值运算符。

六种关系运算符中前四种具有相同的优先级，后两种具有相同的优先级，而且前四种的优先级高于后两种。

注：赋值运算符“＝”和测试等于“＝＝”关系运算符不一样。

两个表达式用关系运算符连接起来就构成了关系表达式。关系表达式的值为逻辑值，即只有真(true)和假(false)两种状态，在 C 语言中用 1 表示真，用 0 表示假。若关系表达式的条件成立，则表达式的值为真(1)，否则为假(0)。

4. 逻辑运算符

逻辑运算符的功能是通过逻辑运算求条件式的逻辑值。

C 语言有以下三种逻辑运算符：

&&　逻辑与

||　　逻辑或

！　　逻辑非

逻辑表达式的格式如下：

```
逻辑与：条件式 1 && 条件式 2
逻辑或：条件式 1 || 条件式 2
逻辑非：！条件式
```

三种逻辑运算中，逻辑非的优先级别最高，且高于算术运算符；逻辑或的优先级别最低，低于关系运算符，但高于赋值运算符。

5. 位运算符

位运算符的功能是对变量按位进行运算，但并不改变运算变量的值。C 语言有以下六种位运算符：

&　　位与

|　　位或

～　位取反
^　位异或
<<　左移
>>　右移

六种位运算符的优先级由高到低的顺序为：位取反～、左移<<、右移>>、位与 &、位异或^、位或|。

6. 复合赋值运算符

复合赋值运算符是C语言的一种特色，它简化了代码的编写。该类运算符的功能是将某个变量先与表达式进行指定的运算，再将运算结果赋予该变量。C语言有以下10种复合赋值运算符：

＋＝　加并赋值运算符
－＝　减并赋值运算符
＊＝　乘并赋值运算符
/＝　除并赋值运算符
%＝　取余并赋值运算符
<<＝　左移并赋值运算符
>>＝　右移并赋值运算符
&＝　位与并赋值运算符
|＝　位或并赋值运算符
^＝　位异或并赋值运算符

C语言中凡是双目运算都可以用复合赋值运算符来表示，格式如下：

```
变量　复合赋值运算符　表达式
```

如：a+=5,相当于 a=a+5;
　　a*=b-6,相当于 a=a*(b-6);
　　y/=x+9,相当于 y=y/(x+9)。

7. 条件运算符

条件运算符是三目运算符，格式如下：

```
判断结果=(判断式)?结果1:结果2
```

其含义是先求判断式的值，若为真，则判断结果＝结果1；若为假，则判断结果＝结果2。如：

```
a=8;b=10;
max=a>b?a:b;
```

结果：max=10;

8. 指针和地址运算符

```
* 指针运算符(取内容)
& 取地址运算符(取地址)
```

一般形式分别如下：

```
取内容：变量= * 指针变量
取地址：指针变量= & 目标变量
```

变量前面加“*”说明该变量为指针，所以操作时取的不是变量的值，而是将指针变量所指向的目标变量的值赋给左边的变量；取地址运算是将目标变量的地址赋给左边的变量。“*”和“&”运算符均为单目运算符。

3.4　C51 语句

C51 语句是单片机执行的操作命令，每条语句都以分号结尾。需要注意的是，变量、函数的声明部分也以分号结尾，但不是语句。

3.4.1　表达式语句

由一个表达式加上一个分号就构成了表达式语句。如：

```
i = 7;
j = a = b;
i++;
```

3.4.2　复合语句

用大括号“{ }”将多条语句括起来就组成了复合语句，也称为功能块。

复合语句中的每一条语句都必须以“;”结束，而不允许将“;”写在“}”外。复合语句不需要以“;”结束。

C 语言中将复合语句视为一条单语句，也就是说在语法上等同于一条单语句。对于一个函数而言，函数体就是一个复合语句。例如：

```
{
    i = 7;
    j = a = b;
    i++;
}
```

3.4.3　空语句

空语句是仅由一个分号“;”组成的语句。空语句什么也不做。

语句格式：;

3.4.4　函数调用语句

函数调用的一般形式加上分号就构成了函数调用语句。

语句格式：函数名(实际参数表);

执行函数调用语句就是调用函数体并把实际参数赋予函数定义中的形式参数，然后执行被调函数体中的语句。

3.4.5 控制语句

控制语句用于控制程序的流程，以实现程序的各种结构方式。C51 语言的控制语句有以下几类：

1. 选择语句 if

(1) if 分支结构

```
if(表达式)
 {
    语句序列;
 }
    其他语句
```

功能：如果表达式的值为真，则执行语句，否则不执行语句。流程图如图 3-1 所示。

(2) if-else 分支结构

```
if(表达式)
    {语句序列 1;}
else
    {语句序列 2;}
其他语句;
```

功能：如果表达式的值为真，则执行语句序列 1，否则执行语句序列 2。流程图如图 3-2 所示。

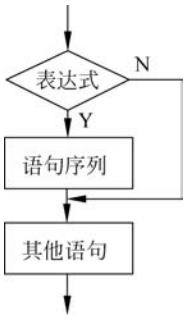

图 3-1 if 分支结构流程图

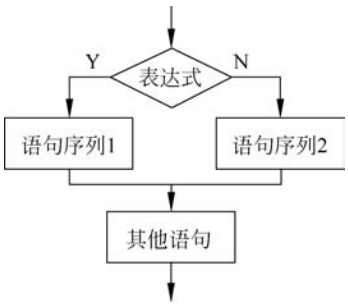

图 3-2 if-else 分支结构流程图

(3) if-else if 分支结构

```
if(表达式 1){语句序列 1;}
else if(表达式 2){语句序列 2;}
else if(表达式 3){语句序列 3;}
…
else if(表达式 n){语句序列 n;}
else {语句序列 n + 1;}
其他语句;
```

流程图如图 3-3 所示。

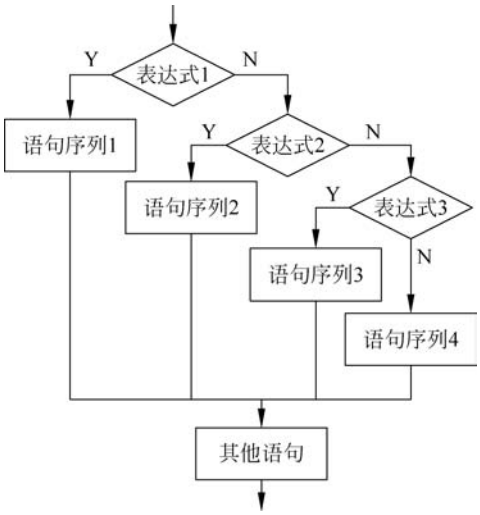

图 3-3　if-else if 分支结构流程图

2. switch 语句

switch 语句是多分支选择语句，也称开关语句。一般格式如下：

```
switch(表达式)
{
    case 常量表达式 1:语句序列 1;
    case 常量表达式 2:语句序列 2;
    …
    case 常量表达式 n:语句序列 n;
    default :语句序列 n + 1;
}
```

每个 case 和 default 出现的顺序不影响执行结果，但每个常量表达式值必须互不相同。该语句的执行过程如下：

(1) 求 switch 后括号内的表达式的值，并将其值与各 case 后的常量表达式值进行比较；

(2) 当表达式的值与某个常量表达式值相等时，则执行该常量表达式后边的语句序列；

(3) 接着执行下一个常量表达式后边的语句序列，直到后边所有的语句序列都执行完(即执行到语句序列 n+1)；

(4) 如果表达式的值与所有 case 后的常量表达式值都不相等，则执行 default 后面的语句序列。

通常当某个常量表达式的值与 switch 后表达式的值相等时，只需要执行该 case 后的语句序列，不希望程序一直执行下去，直到语句序列 n+1。要达到这一目的，只需要在每个语句序列后加上“break”语句即可。

格式如下：

```
switch(表达式)
{
    case 常量表达式 1:语句序列 1;
```

```
    break;
    case 常量表达式 2:语句序列 2;
    break;
    …
    case 常量表达式 n:语句序列 n;
    break;
    default :语句序列 n + 1;
}
```

流程图如图 3-4 所示。

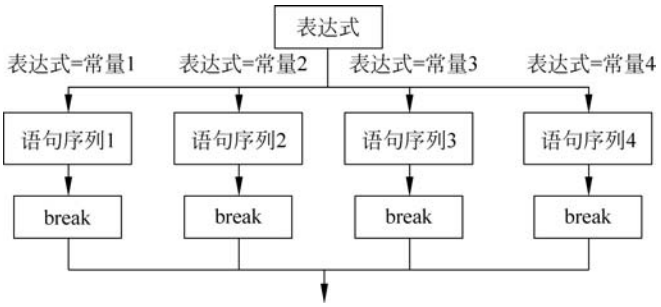

图 3-4　switch 分支结构流程图

3. for 语句

for 语句格式如下：

```
        for(表达式 1;表达式 2;表达式 3)
{语句序列;}      //循环体,可为空
```

表达式 1 通常为赋值表达式，用来确定循环结构中控制循环次数的变量的初始值，实现循环控制变量的初始化。

表达式 2 通常为关系表达式或逻辑表达式，用来判断循环是否继续进行。

表达式 3 通常为表达式语句，用来描述循环控制变量的变化，最常见的是自增或自减表达式，实现对循环控制变量的修改，当循环条件满足时就执行循环体内的语句序列。

语句序列可以是简单语句，也可以是复合语句。若只有一条语句，则可以省略{ }。

for 语句的执行过程如下：

(1) 计算表达式 1 的值，为循环控制变量赋初值。

(2) 计算表达式 2 的值，如果为“真”则执行循环体一次，否则退出循环，执行 for 循环后的语句。

(3) 如果执行了循环体语句，则执行循环体后，要计算表达式 3 的值，调整循环控制变量。然后回到第(2)步重复执行，直到表达式 2 的值为“假”时，退出循环。流程图如图 3-5 所示。

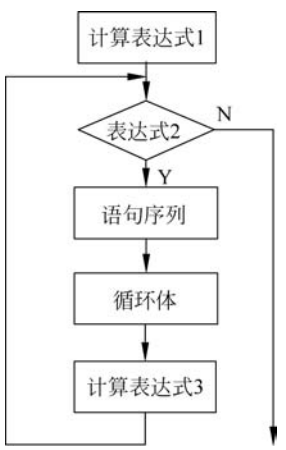

图 3-5　for 分支结构流程图

4. while 语句

(1) while 语句

while 语句用于实现“当型”循环的语句，格式为：

```
while (条件表达式)
{
    语句序列;      //循环体
}
```

流程图如图 3-6 所示。

(2) do-while 语句

do-while 语句用于实现“直到型”循环的语句，格式如下：

```
do
{
    语句序列;
}
while(表达式);
```

流程图如图 3-7 所示。

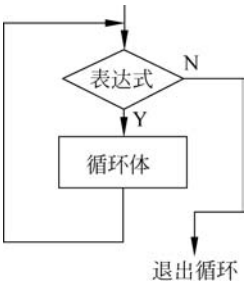

图 3-6　while 分支结构流程图

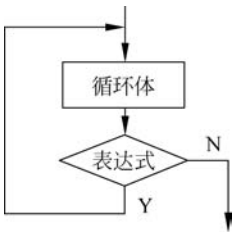

图 3-7　do-while 分支结构流程图

提示：if、for、while 等语句是程序代码的重要组成部分，难度不算大，是编程基础代码，要好好掌握。

3.5　C51 函数

3.5.1　函数的一般格式

1. 函数的定义

C 程序由一个主函数 main()和若干个其他函数组成。由主函数调用其他函数，其他函数也可以互相调用，同一个函数可以被调用多次。

函数定义的一般格式为：

```
函数类型 函数名 (形式参数列表)
  {
    局部变量声明;
    语句;
    (有返回值的要有 return 语句)
}
```

2. 函数返回值

返回语句 return 用来回送一个数值给定义的函数，从函数中退出。返回值是通过 return 语句返回的。如果函数无须返回值，可以用 void 类型指明函数无返回值。

3. 形式参数与实际参数

与使用变量一样，在调用一个函数之前，必须对该函数进行声明。函数声明的一般格式为：

```
函数类型  函数名(形式参数列表)
```

函数定义时参数列表中的参数称为形式参数，简称形参。

函数调用时所使用的替换参数，是实际参数，简称实参。定义的形参与函数调用的实参类型应该一致，书写顺序应该相同。

4. 调用函数的方式

被调用的函数必须是已经存在的函数。

(1) 函数作为语句。把函数调用作为一个语句，不使用函数返回值，只是完成函数所定义的操作。例如：

```
DelayMS(150);
```

(2) 函数作为表达式。函数调用出现在一个表达式中，使用函数的返回值。例如：

```
int k;
k = sum(a,b);
```

(3) 函数作为一个参数。函数调用作为另一个函数的实参。例如：

```
int k;
k = sum(sum(a,b),c);
```

3.5.2 中断函数

C51 语言中断函数的结构与其他函数的结构类似，但中断函数不带任何参数，而且使用中断函数之前不需要声明。

定义中断函数的格式如下：

```
void  函数名( ) interrupt 中断号 n (using 工作寄存器组 m)
```

在中断函数中为了避免数据的冲突，可指定一个寄存器组；若不需要指定，则该项可以省略。MCS-51 单片机中断源和入口地址如表 3-5 所示。

表 3-5 中断源与入口地址

中断号 n	中断源	入口地址 8n+3
0	外部中断 0	0x03
1	定时器/计数器 0	0x0B
2	外部中断 1	0x13
3	定时器/计数器 1	0x1B
4	串行口	0x23

3.5.3　C51 的库函数

C51 语言具有丰富的可供直接调用的库函数，使用库函数可使程序代码简单，结构清晰，易于调试和维护。

每个函数都在相应的头文件(.h)中有原型声明。如果使用库函数，必须在源程序中用预处理命令“#include”将与该函数相关的头文件(即包含了该函数的原型声明文件)包含进来，否则将不能保证函数的正确执行。

3.5.4　本征库函数和非本征库函数

本征库函数是指编译时直接将固定的代码插入当前行，大大提高了函数访问的效率。

C51 有以下 9 个本征库函数：

crol，_cror_：将无符号字符型变量循环向左(右)移动指定位数后返回；

irol，_iror_：将无符号整型变量循环向左(右)移动指定位数后返回；

lrol，lror_：将无符号长整型变量循环向左(右)移动指定位数后返回；

nop：空操作；

testbit：测试该位变量并跳转，同时将该位变量清除；

chkfloat：检查浮点数的类型。

3.5.5　几类重要的库函数

1. 内部函数 intrins.h

如：

```
#incluclе<intrins.h>
void main()
{
    unsigned int x;
    x=0x00ff;
    x=_irol_(x,4);
}
```

运行后，x＝0x0ff0。

2. 绝对地址访问函数 absacc.h

通过包含头文件 absacc.h 进行绝对地址访问。

方法：把通用指针指向各存储空间的首地址，并按存取对象类型实施指针强制，再用定义宏说明为数组名。对于绝对地址对象的存取，用指定下标的抽象数组来实现。

char 类型：CBYTE[i]　DBYTE[i]　PBYTE[i]　XBYTE[i]

int 类型：CWORD[i]　DWORD[i]　PWORD[i]　XWORD[i]

3. 缓冲区处理函数 string.h

(1) 计算字符串 s 的长度 strlen。

原型：`extern int strlen(char *s);`

说明：返回 s 的长度，不包括结束符 NULL。

(2) 由 src 所指内存区域复制 count 个字节到 dest 所指内存区域。

memcpy 原型：extern void * memcpy(void * dest, void * src, unsigned int count);

说明：src 和 dest 所指内存区域不能重叠，函数返回指向 dest 的指针。

(3) 由 src 所指内存区域复制 count 个字节到 dest 所指内存区域。

memmove 原型：extern void * memmove(void * dest, const void * src, unsigned int count);

说明：与 memcpy 工作方式相同，但 src 和 dest 所指内存区域可以重叠，复制后 src 内容会被更改。函数返回指向 dest 的指针。

(4) 比较内存区域 buf1 和 buf2 的前 count 个字节。

memcmp 原型：extern int memcmp(void * buf1, void * buf2, unsigned int count);

说明：当 buf1 < buf2 时，返回值< 0；当 buf1 = buf2 时，返回值 = 0；当 buf1 > buf2 时，返回值> 0。

(5) 把 buffer 所指内存区域的前 count 个字节设置成字符 c。

memset 原型：extern void * memset(void * buffer, int c, int count);

说明：返回指向 buffer 的指针。

(6) 从 buf 所指内存区域的前 count 个字节查找字符 ch。

memchr 原型：extern void * memchr(void * buf, char ch, unsigned count);

说明：当第一次遇到字符 ch 时停止查找。如果成功，返回指向字符 ch 的指针；否则返回 NULL。

提高篇

Keil 编译器支持多文件系统，但文件再多，也只能包含一个入口函数 main()；建立 ex1.c，ex2.c 和 ex3.c 三个文件，其中 main()函数放在 ex1.c 文件中。下面比较绝对地址的几种应用方法。

ex1.c 文件

```
#include "reg51.h"
//DBYTE、CBYTE、XBYTE 必须先加 absacc.h 头文件
#include "absacc.h"
#define uchar unsigned char
uchar xx;
void main()
{

  DBYTE[0x20] = 0x88;      //把数据 0x88 送到内部 RAM0x20 单元
  DBYTE[0x21] = 0x99;      //把数据 0x99 送到内部 RAM0x21 单元
  xx = 0x21;               //把数据 0x21 送到变量 xx
  func1();
  func2();
}
```

Keil main 程序运行如图 3-8 所示。

ex2.c 文件

```
#define uchar unsigned char
```

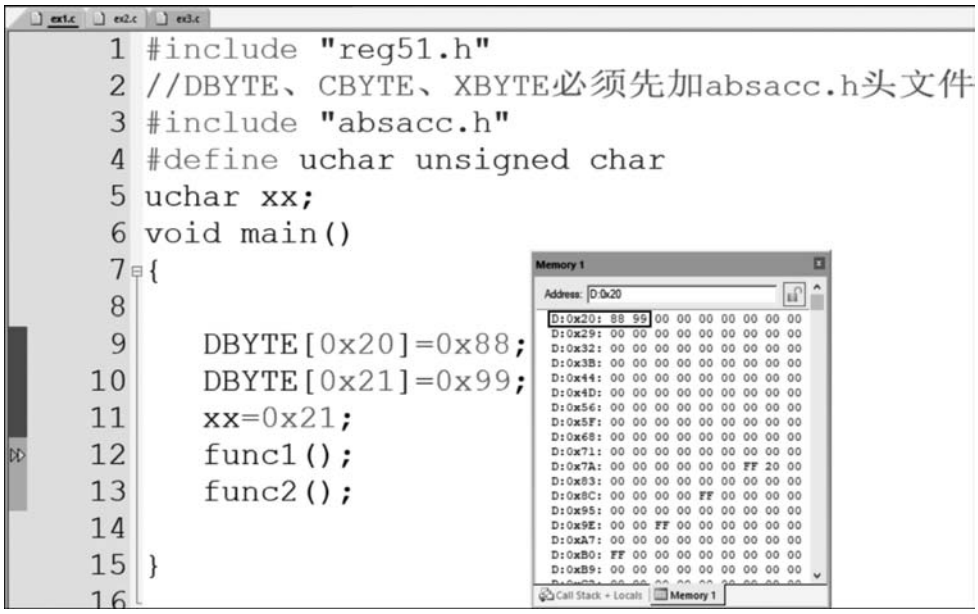

图 3-8　Keil main 程序运行图

```
data uchar * p;
void func1()
{
    p = 0x20;           //指针地址指向 0x20
    * p = 0x44;         //把数据 0x44 送到内部 RAM0x20 单元
}
```

Keil func1 程序运行如图 3-9 所示。

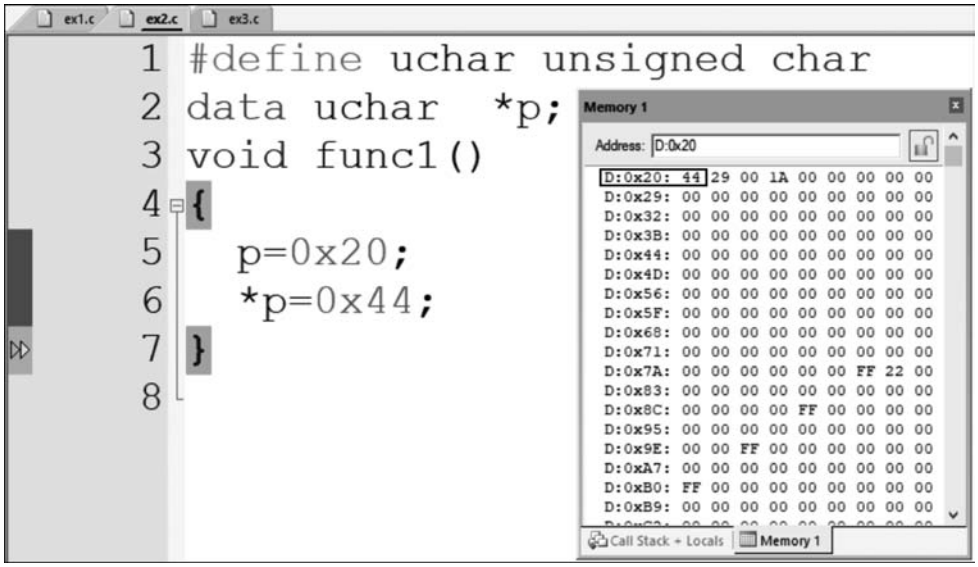

图 3-9　Keil func1 程序运行图

```
ex3.c 文件
#define uchar unsigned char
/* 注意:此_at_变量定义不能放在函数体内,否则编译出错 */
unsigned char a_20h _at_ 0x20;
void func2()
{
  a_20h = 0x22;          //把数据 0x22 送到内部 RAM0x20 单元
}
```

Keil func2 程序运行如图 3-10 所示。

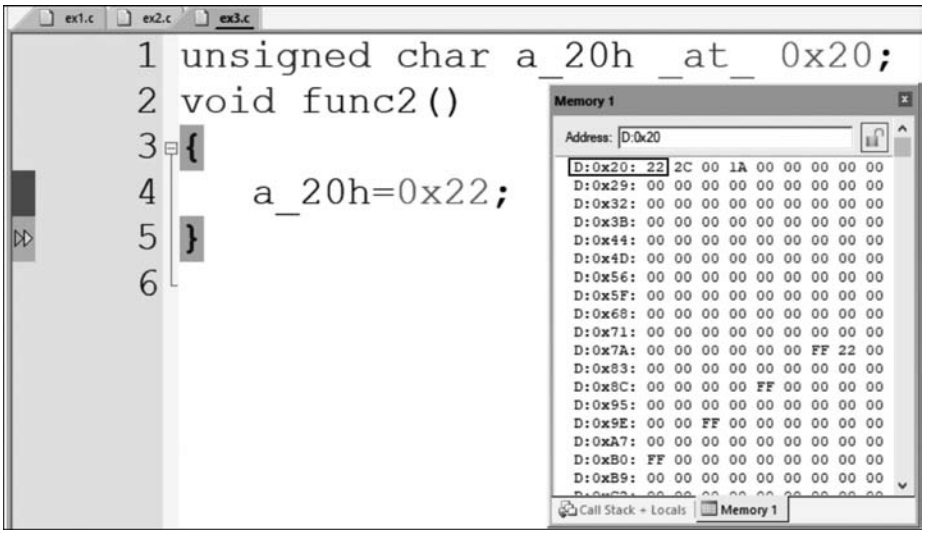

图 3-10 Keil func2 程序运行图

注：查看内部存储器指令 D:0x20(X-外部 RAM,D-内部 RAM(0x00-0x7F),I-内部 RAM(0x00-00xFF),C-程序存储器)。

习题与思考

3-1 请归纳 C51 语言有哪些关键字?

3-2 判断下列 bit 型变量定义的正误：

```
bit  data  a1;
bit  bdata  a2;
bit  pdata  a3;
bit  xdata  a4;
```

3-3 在 C51 程序里,一般函数和中断函数有什么不同?

3-4 按给定存储器类型和数据类型,写出下列变量的说明形式：

(1) 在 data 区定义字符变量 val1；

(2) 在 idata 区定义整型变量 val2；

(3) 在 xdata 区定义无符号字符数组 val3[9]；

(4) 定义位寻址变量 flag;

(5) 定义特殊功能寄存器 P3;

(6) 定义特殊功能寄存器 TCON;

(7) 定义 16 位特殊功能寄存器 T0。

3-5　用三种不同的循环结构实现 1～100 的求和。

3-6　用指令实现下列功能:

(1) 用绝对宏实现:读出外部数据存储器 0x40 内容,送到内部存储器 0x30 单元。

(2) 用指针实现:读出外部数据存储器 0x20 内容,送到内部存储器 0x20 单元。

(3) 用_at_关键字实现:读出外部数据存储器 0x20 内容,送到内部存储器 0x20 单元。

(4) 用_at_关键字实现:读出程序存储器 0x20 内容,送到外部数据存储器 0x20 单元。

3-7　如图 3-11 所示,当开关 K 闭合时 4 个发光二极管亮,K 断开时 4 个发光二极管灭。试编写程序。

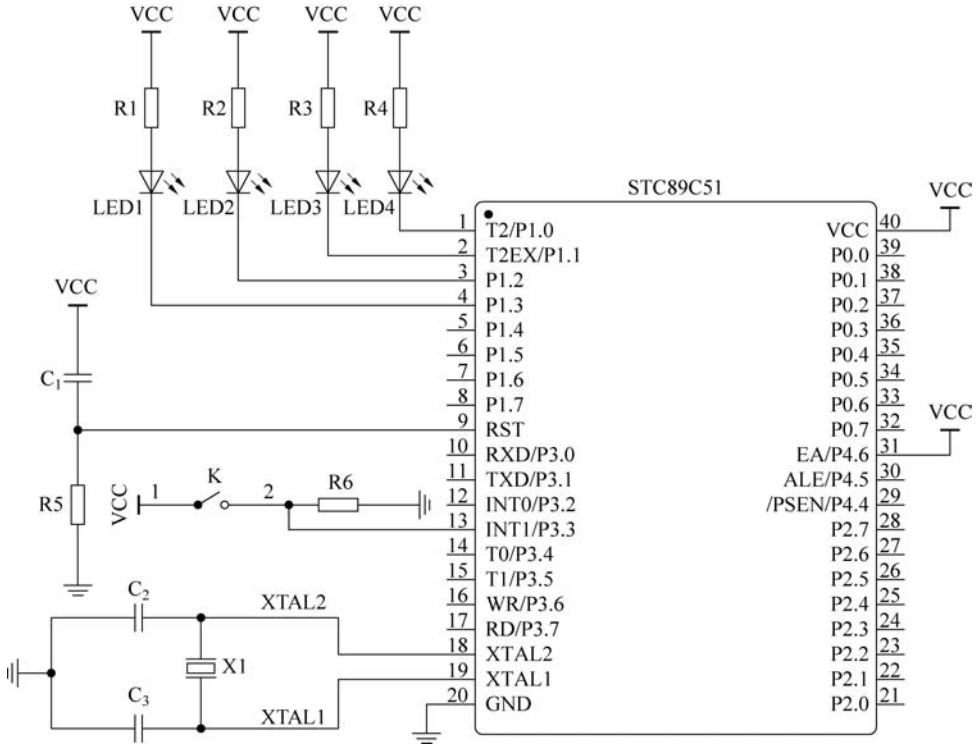

图 3-11　按键控制指示灯电路图

3-8　写出下列代码的 while 和 do-while 的算法:

```
unsigned char i;
unsigned int s = 0;
void main( )
{
for(i = 0;i < 250;i++){s = s + i;}
}
```