第 3 章

51 单片机 C51 语言编程基础

视频

早期单片机程序设计采用汇编语言，程序设计难度大。20 世纪末，单片机的 C 语言编译器逐渐兴起，与汇编语言程序相比，C 语言程序在功能上、结构上、可读性、可移植性、可维护性等方面有明显优势，且易学易用，现已成主流的单片机编程语言。Keil C51 是美国 Keil Software 公司专为 51 系列兼容单片机设计的高级语言 C 编译器，是使用最广泛的 51 单片机 C 语言编译器，C51 保留了 ANSI C 的所有规范，并针对 8051 单片机的特点作了一些特殊的拓展。

3.1 C51 程序与编程规范

3.1.1 C51 的程序结构

C51 源程序文件扩展名为.c，其结构与 ANSI C 语言相同，包括预处理命令、全局声明和函数定义 3 部分构成，各部分程序均有注释，程序一般结构如下：

```
/*程序说明、功能、设计者、设计时间、修改时间、版本描述等*/
预处理命令                          //用于包含头文件等
全局变量声明;                       //全局变量可被本程序所有函数引用

函数 1(形参说明表)声明;
…;
函数 n(形参说明表)声明;

/*主函数*/
main()
{     局部变量声明;
      执行语句;
      函数 i 调用(实参表);
}

/*其他函数定义*/
函数 1(形参说明表)                  //函数 1 定义
```

```
{       局部变量声明;                    //局部变量只能在函数内部引用
        执行语句;
        函数 j 调用(实参表);
}
…
函数 n(形参说明表)                       //函数 n 定义
{           …;
}
```

1. 预处理命令

预处理命令包括文件包含(#include)、宏定义及撤销(#define,#undef)、条件编译(#if、#else、#endif)。

2. 函数定义

函数是C51程序的基本单位,程序由一个或多个函数构成,其中有且仅有一个名为main()的主函数,程序总是从 main()函数开始执行,主函数所有语句执行完毕,则程序结束。主函数中可以调用其他功能函数,调用结束后又返回主函数,功能函数不能调用主函数,但功能函数之间可相互调用和嵌套。功能函数可以用户自定义,也可以是C51编译器提供的库函数。

3. 全局声明

全局声明包括变量声明和函数声明,在函数之外定义的变量为全局变量,在函数之内定义的变量则为局部变量。当一个函数调用另一个函数时,被调用的函数必须是已定义的,还未定义的必须先声明,被调用函数往往和全局变量一起声明。

4. 程序注释

注释语句只起对程序代码功能进行描述的作用,增加程序的可读性,不参与程序的编译链接,不产生可执行的机器码。与 ANSI C 的规则相同,C51 源程序的注释有单行注释和块式注释两种。

1) 单行注释

以"//"符号开始,其后面至行末的所有符号构成单行注释。一般放在执行语句的后面,用于对其所在行的语句功能进行注释。

2) 块式注释

以"/*"符号开始、以"*/"符号结束,其间的所有符号构成块式注释。一般放在某程序模块的前面,用于对其后的程序模块功能进行注释。

3.1.2　C51 的标志符与关键字

1. 标志符

在编程语言中,用来对变量、符号常量、函数、数组、结构体等对象命名的有效字符串称为标志符(或标识符),C51 语言支持自定义的标志符。与 ANSI C 标志符的命名规则完全相同,C51 的标志符可以由字母、下画线"_"及数字 0～9 组成,首符号必须是字母或下画线,

最多 32 个字符，区分大小写。

2. 关键字

C51 语言继承了 ANSI C 的 32 个关键字，并根据 51 系列单片机的硬件特点，增加了一些关键字，C51 语言的关键字如表 3.1 所示。

表 3.1 C51 语言的关键字

类别	关键字	类 型	作 用
ANSI C 标准关键字	void	基本数据类型声明	声明无(或空)类型数据
	char	基本数据类型声明	声明单字节整型数据或字符型数据
	int	基本数据类型声明	声明整型数据
	float	基本数据类型声明	声明单精度浮点型数据
	double	基本数据类型声明	声明双精度浮点型数据
	short	数据类型修饰说明	修饰整型数据，短整型数据
	long	数据类型修饰说明	修饰整型数据，长整型数据
	signed	数据类型修饰说明	修饰整型数据，有符号整数(二进制补码表示)
	unsigned	数据类型修饰说明	修饰整型数据，无符号整数
	enum	复杂数据类型声明	声明枚举型数据
	struct	复杂数据类型声明	声明结构类型数据
	union	复杂数据类型声明	声明联合(共用)类型数据
	typedef	复杂数据类型声明	声明重新定义数据类型
	sizeof	数据类型大小运算符	计算特定类型的表达式或变量的大小(即字节数)
	auto	数据存储类别说明	指定变量为自动变量，其空间在动态存储区分配(默认)
	static	数据存储类别说明	指定变量为静态变量，其空间在静态存储区分配
	register	数据存储类别说明	指定局部变量为寄存器变量，其空间优先在 CPU 内部寄存器分配
	extern	数据存储类别说明	指定变量为外部变量，由其他程序文件声明的全局变量
	const	数据存储类别说明	指定变量的值在程序执行过程中不可变更
	volatile	数据存储类别说明	指定变量的值在程序执行过程中可被隐含地变更
	return	流程控制(跳转)	用于函数体中，返回特定值
	continue	流程控制(跳转)	中止当前循环，开始下一轮循环
	break	流程控制(跳转)	退出当前循环或 switch 结构
	goto	流程控制(跳转)	无条件转移语句
	if	流程控制(分支)	用于构成 if…else 条件语句，条件及其肯定分支
	else	流程控制(分支)	用于构成 if…else 条件语句，否定分支
	switch	流程控制(分支)	用于构成 switch…case 开关语句，分支条件
	case	流程控制(分支)	用于构成 switch…case 开关语句中的第 n 个分支
	default	流程控制(分支)	用于构成 switch…case 开关语句中的其他分支
	for	流程控制(循环)	用于构成 for 循环结构语句
	do	流程控制(循环)	用于构成 do…while 循环结构
	while	流程控制(循环)	用于构成 do…while 或 while 循环结构

续表

类别	关键字	类　型	作　用
C51扩展关键字	bit	位变量声明	声明一个位变量或位类型的函数
	sbit	位变量声明	声明一个可位寻址变量(指定位地址)
	sfr	特殊功能寄存器声明	声明一个8位特殊功能寄存器(指定字节地址)
	sfr16	特殊功能寄存器声明	声明一个16位的特殊功能寄存器(指定低位字节地址)
	data	存储器类型说明	指定直接寻址的单片机片内数据存储器
	bdata	存储器类型说明	指定可位寻址的单片机片内数据存储器
	idata	存储器类型说明	指定间接寻址的单片机片内数据存储器
	pdata	存储器类型说明	指定分页寻址的单片机扩展数据存储器
	xdata	存储器类型说明	指定单片机扩展数据存储器
	code	存储器类型说明	指定单片机程序存储器
	interrupt	中断函数说明	说明一个中断服务函数
	reentrant	再入函数说明	说明一个再入函数
	using	寄存器组说明	指定单片机的工作寄存器组
	_at_	存储绝对地址说明	声明变量时指定其所在地址
	small	编译模式说明	—
	compact	编译模式说明	—
	large	编译模式说明	—

3.1.3　C51编程规范

养成良好的编程习惯很重要,按一定规范编写程序,有助于设计者理清思路,方便程序纠错、修改、整理、阅读,是程序可维护和可移植的前提。进行C51程序设计时,应注意以下几方面的编程规范。

(1) 一个好的源程序应该添加必要的注释,以增加程序的可读性。

(2) 标志符应能直观反映其含义,避免过于冗长,方便源程序的编写、阅读与理解。

(3) 编译系统专用标志符以下画线开头,建议自定义标志符不使用下画线为首字符。

(4) 自定义的标志符避免与关键字或C51库函数同名。

(5) 虽然C51语言没有限制主函数main()放置的位置,但为阅读方便,最好将其放在所有自定义函数的前面,程序语句的书写顺序依次为:头文件声明、全局变量和自定义函数声明、主函数main()、自定义函数。

(6) 每条语句单独写一行,或将配合完成某一功能的几条短语句写在同一行,并注释。

(7) 源程序文件中不同结构部分之间要留有空行,来明显区分不同的结构。

(8) 对if、for、while等块结构语句中的“{”和“}”要配对对齐,以突出该块结构的起始和结束位置,使程序阅读更直观。

(9) 源程序书写时,可以通过适当的Tab键操作来实现代码对齐。

3.2 C51 语言的数据

使用数据时，与 ANSI C 基本相同，C51 也要说明其数据类型、常量或变量、变量的作用范围；与 ANSI C 略有不同，C51 还要说明其存放在存储器的什么区域。

3.2.1 数据类型

ANSI C 语言数据类型包括基本类型、构造类型、指针类型以及空类型等。基本类型有字符、整型、短整型、长整型、浮点型、双精度浮点型等；构造类型包括数组、结构体、共用体以及枚举类型等。基本数据类型中，C51 语言增加了位(bit)类型。整型和短整型相同，均为 16 位；浮点型和双精度浮点型相同，均为 32 位。因此，在 C51 中使用 short 和 double 标志符是没有实际意义的。C51 支持的基本数据类型、长度及值域如表 3.2 所示。

表 3.2 C51 支持的基本数据类型、长度及值域

数据类型	标志符	长度/bit(Byte)	值域
位型	bit	1	0,1
无符号字符型	unsigned char	8(1)	0～255
有符号字符型	signed char	8(1)	－128～127
无符号整型	unsigned int	16(2)	0～65 535
有符号整型	signed int	16(2)	－32 768～32 767
无符号长整型	unsigned long	32(4)	0～4 294 967 295
有符号长整型	signed long	32(4)	－2 147 483 648～2 147 483 647
浮点型	float	32(4)	±1.175494E－38～±3.402823E＋38
指针型	*	8～24(1～3)	对象地址 0～65 535

数据的存储有大端与小端模式之别。所谓大端模式(Big-Endian)，即高位字节存储在低地址，而小端模式(Little-Endian)则低位字节存储在低地址。C51 的数据存储采用大端模式。

3.2.2 常量与变量及其存储模式

1. 常量

常量是不接受程序修改的固定值，C51 的常量包括位常量、整型常量、实型常量、字符型常量、字符串常量等。

1) 位常量

位常量即位常数，只有两个值：0 或 1。

2) 整型常量

整型常量即整常数，可带负号。通常情况下，C51 程序设计时常采用十进制和十六进制。十进制整常数以非 0 开始的整数表示，例如，－6、965 等。十六进制整常数以 0x 开始的数表示，例如，－0x2a、0xc1f4 等。

整型常量后加字母“L”或“l”表示该数为长整型，例如 965 为整型，在内存中占 2 字节，

而 965L 为长整型，在内存中占 4 字节。

3）实型常量

实型常量又称浮点常量，是用十进制表示的实常数，可带负号，值包括整数部分、尾数部分和指数部分，指数以 10 为基数，幂为有符号整数，指数部分以 E 或 e 开头，字母 E 或 e 之前必须有数字，例如，－3.14、126E－3 表示 126×10^{-3}、1.76E6 表示 1.76×10^{6}。

C51 语言的浮点数使用 24 位精度（单精度），占 4 字节的存储单元，浮点数的二进制编码如表 3.3 所示。

表 3.3　浮点数的二进制编码

位	31～24	23～16	15～8	7～0
内容	SEEEEEEE	EMMMMMMM	MMMMMMMM	MMMMMMMM

其中 S（1 位）为符号位，0 表示“正”，1 表示“负”；E（8 位）为指数，偏移为 127，E 取值范围 1～254；24 位的数值中规定整数部分 1 位和尾数部分 23 位，整数始终为 1 不保存，仅保存 23 位的尾数 M。例如，浮点数－13.75，其二进制表示为－1101.11，即 $-1.10111\times2^{130-127}$，因此符号位 $S=1$，指数 $E=130$（即 10000010），尾数 $M=1011100\ 00000000\ 00000000$，所以－13.75 的十六进制编码为 0xc15c0000，采用不同的端模式，该浮点数在内存中的信息和占用的存储单元亦不同，如表 3.4 所示。

表 3.4　浮点数－13.75 在内存中的存储模式

存储地址（首地址 Addr）	Addr＋0	Addr＋1	Addr＋2	Addr＋3
大端模式（C51 的模式）	0xc1	0x5c	0x00	0x00
小端模式	0x00	0x00	0x5c	0xc1

4）字符型常量

字符型常量是指用一对单引号括起来的单个字符，并按其对应的 ASCII 码值来存储，如'a'、'9'、'!'等，ASCII 码参见附录 A。

5）字符串常量

字符串常量是指用一对双引号括起来的一串字符，字符串在内存中以 ASCII 码值存储，系统自动在字符串的末尾加的一个结束标志 NUL，其 ASCII 码值为 00H。因此在程序中，长度为 n 个字符的字符串常量，在存储器中占 $n+1$ 字节的存储空间。例如，程序中的常量 9、'9'、"9"在存储器中的信息和占用空间是不同的。

6）符号常量

C51 允许将程序中的常量定义为一个标志符，称为符号常量。符号常量一般使用大写字母表示，符号常量使用前必须用宏定义（#define）命令定义。例如：

```
#define PI 3.1416                    //定义实型符号常量 PI 为圆周率
```

2. 变量及其存储模式

变量是程序执行过程中其值可以改变的量。变量使用前必须先定义，用一个标志符作

为变量名。在 ANSI C 中，变量定义的格式如下：

[存储类别] 数据类型 变量名表;

存储类别(或存储类型)表示变量的存储方式，存储方式有静态和动态两大类，静态存储方式是指程序运行期间由系统分配固定的存储空间的方式；动态存储方式是指程序运行期间根据需要动态地分配存储空间的方式。存储类别共有 4 种：auto(自动的)、static(静态的)、register(寄存器的)、extern(外部的)，变量定义时未指定存储类别时，默认存储类别为 auto。局部或全局特性表征变量的作用域各异，动态或静态存储特性表征变量的生存期不同，各种类型变量的作用域和生存期如表 3.5 所示。

表 3.5 各种类型变量的作用域和生存期

变量存储类别	在函数内定义		在函数外定义	
	作用域	生存期	作用域	生存期
register	局部	动态	—	—
auto(或无修饰)	局部	动态	可全局	静态
static	局部	静态	限本文件	静态
extern	全局	静态	全局	静态

针对 51 单片机的数据存储器被分为片内 RAM、片外 RAM 和程序存储区的特点，在 C51 中，定义变量时还可以指定给变量分配的存储器类型，变量定义的格式如下：

[存储类别] 数据类型 [存储器类型] 变量名表;

存储器类型选项用于指定给变量分配的存储器类型，进行准确的地址范围定位。C51 编译器能够识别存储器类型如表 3.6 所示。

表 3.6 C51 存储器类型与数据存储空间的对应关系

存储器类型	说　明
data	片内 RAM 的低 128 字节，DATA 区(00H～7FH 地址空间)，直接寻址
bdata	片内 RAM 的位寻址区，BDATA 区(20H～2FH 地址空间)，可位寻址
idata	片内 RAM 的 256 字节，IDATA 区(00H～FFH 地址空间)，间接寻址
pdata	外部 RAM 存储器分页寻址(256 字节)，PDATA 区(00H～FFH 地址空间) 用“MOVX A,@Ri”和“MOVX @Ri,A”指令访问(详见第 8 章)
xdata	外部 RAM 存储器，XDATA 区(0000H～FFFFH 地址空间) 用“MOVX A,@DPTR”和 “MOVX @DPTR,A”指令访问(详见第 8 章)
code	程序存储器 ROM，CODE 区(0000H～FFFFH 地址空间) 用“MOVC A,@A+DPTR”和“MOVC A,@A+PC”指令读(只读，详见第 8 章)

变量定义实例如下：

```
unsigned char data var;          //在 DATA 区定义无符号字符型变量 var
char bdata flags;                //在 BDATA 区定义字符型变量 flags
extern float idata x,y,z;        //在 IDATA 区定义浮点型外部变量 x,y,z
float pdata a,b;                 //在 PDATA 区定义浮点型变量 a,b
char xdata text[] = "OK!";       //在 XDATA 区定义字符串数组变量 text[]
unsigned int code V_to_T[200]; //在 CODE 区定义无符号整型数组变量 V_to_T[]
```

变量定义时如果省略存储器类型，Keil C51 编译器按选择的编译模式 Small、Compact 或 Large 确定默认的存储器类型，各编译模式对应的默认存储器类型如表 3.7 所示。

表 3.7　编译模式对应的默认存储器类型

编 译 模 式	默认的存储器类型
Small(小模式)	data(访问速度最快)
Compact(紧凑模式)	pdata(访问速度较快)
Large(大模式)	xdata(访问速度最慢)

3.3　用 C51 语言描述单片机资源

C51 语言对 51 单片机资源的描述主要有特殊功能寄存器和位变量的定义，以及以绝对地址方式访问片内 RAM、片外 RAM 和 I/O 端口。

3.3.1　特殊功能寄存器定义

51 单片机通过特殊功能寄存器 SFR 实现对 CPU 及其片内各种 I/O 功能模块的管理和控制，表 2.2 列出了 STC15W4K32S4 系列单片机特殊功能寄存器名称及地址映像。特殊功能寄存器的地址范围为 80H～FFH，只能用直接寻址方式访问，其中地址能被 8 整除的特殊功能寄存器还是可以位寻址的。

为了能用直接寻址方式访问特殊功能寄存器，C51 语言引入扩展的关键字 sfr，专用于特殊功能寄存器定义，其语法如下：

```
sfr 特殊功能寄存器名字 = 特殊功能寄存器地址;
```

例如：

```
sfr PSW = 0xd0;                  //程序状态字寄存器 PSW 地址 D0H(见表 2.2)
sfr DPL = 0x82;                  //16 位数据指针 DPTR 低 8 位地址 82H
sfr DPH = 0x83;                  //16 位数据指针 DPTR 高 8 位地址 83H
```

特殊功能寄存器名称(标志符)通常采用大写字母，用 sfr 定义特殊功能寄存器时，“=”后面的地址必须是常数，其值为 0x80～0xff，不允许使用带运算符的表达式。

在51单片机中,有部分特殊功能寄存器可组合成16位的寄存器,其特征是16位寄存器的高位字节地址直接位于低位字节地址之后,C51可用关键字sfr16将其定义为16位特殊功能寄存器,可直接访问其16位值。sfr16的语法与sfr类似,用低位字节的地址作为16位特殊功能寄存器的地址。

例如,51单片机有16位数据指针DPTR,如表2.2所示,DPTR由高位字节DPH和低位字节DPL两个8位寄存器组合而成,DPH的地址(83H)紧随DPL的地址(82H)之后,因此DPTR可如下定义:

```
sfr16 DPTR = 0x82;     //定义16位特殊功能寄存器,数据指针DPTR低位字节DPL地址82H
```

51单片机还有一些16位的特殊功能寄存器,如表2.2所示的定时/计数器T0(详见第6章),它也由高位字节TH0和低位字节TL0组合而成,但前者地址不紧随后者,这类寄存器不能使用sfr16关键字将其定义为16位特殊功能寄存器,其16位值不能直接访问。

3.3.2 位变量定义

51单片机片内RAM有可位寻址区BDATA,其字节地址为20H～2FH,位地址为00H～7FH,此外地址可被8整除的特殊功能寄存器SFR也是可位寻址的,位地址为80H～FFH。单片机的CPU都可以处理位数据,对此C51增加了bit(位)数据类型,位变量的定义有两种方式,一是一般位变量定义,位地址由编译系统在BDATA区自动分配;二是可位寻址整型变量和SFR的位变量定义,位地址是确定的。

1. 一般位变量定义

一般位变量定义与其他基本数据类型变量的定义相似,如:

```
bit k0;                          //在BDATA区定义位变量k0
```

2. 可位寻址位变量定义

可位寻址位变量定义使用sbit关键字,其语法格式如下:

```
sbit 位变量名 = 可寻址位的位地址;
```

可位寻址整型变量的位变量定义需先在BDATA区定义整型变量,可寻址位的位地址以“整型变量名^位序”的形式定义。如:

```
unsigned char bdata flag;        //在BDATA区定义可位寻址8位字符型变量flag
sbit bx = flag^3;                //8位整型flag的第3位定义为bx
unsigned int bdata key;          //在BDATA区定义可位寻址16位整型变量key
sbit k0 = key^0;                 //16位整型key的高位字节第0位为k0
sbit k7 = key^7;                 //16位整型key的高位字节第7位为k7
sbit k8 = key^8;                 //16位整型key的低位字节第0位定义为k8
sbit k15 = key^15;               //16位整型key的低位字节第7位定义为k15
```

对可位寻址的特殊功能寄存器,可寻址位的位地址有3种表示方法,其一先用sfr定义

SFR 名称，然后用“SFR 名称^位序”；其二用“SFR 地址^位序”；其三用绝对位地址。例如，程序状态字 PSW 的地址是 D0H（见表 2.2），该地址能被 8 整除，所以 PSW 是可位寻址的 SFR，其各位含义详见 PSW 寄存器描述，相应位变量可如下定义：

```
sfr PSW = 0xd0;                //定义特殊功能寄存器 PSW
sbit CY = PSW^7;               //用"SFR 名称^位序"表示位地址
sbit AC = 0xd0^6;              //用"SFR 地址^位序"表示位地址
sbit OV = 0xd2;                //用绝对位地址表示位地址
```

又例如，51 单片机 8 位并行 I/O 口 P1 的地址（90H）可被 8 整除，可位寻址，该并口各位可如下定义：

```
sfr P1 = 0x90;                 //定义特殊功能寄存器 P1
sbit P10 = P1^0;               //定义 P1.0 端口
sbit P11 = P1^1;               //定义 P1.1 端口
sbit P17 = P1^7;               //定义 P1.7 端口
```

3. 通过头文件访问特殊功能寄存器及其可位寻址位

Keil C51 编译器把标准 51 系列单片机的所有特殊功能寄存器及其可位寻址位进行了定义，存放在头文件 Keil\C51\INC\REG51.H 和 REG52.H 中，分别对应 80C51 和 80C52 单片机。宏晶科技有限公司也为各系列 STC 单片机的特殊功能寄存器及其可位寻址位作了定义，并存放在头文件中，STC15W4K32S4 系列单片机的头文件为 Keil\C51\INC\STC\STC15.H。开发用户程序时，只要在程序开始时用预处理命令 #include 将这个头文件包含到程序中，就可直接使用特殊功能寄存器及其可位寻址位的名称。

【例 3.1】 头文件引用举例。

```
#include <stc15.h>             //使用 STC15 系列单片机，引用片内资源定义头文件
void main(void)
{
    ACC = 0x0f;                //给累加器 ACC 赋值 0x0f
    P1 = 0x5a;                 //从 8 位并口 P1 输出 0x5a
    …
}
```

3.3.3　绝对地址访问

通常单片机系统的数据存储器和 I/O 接口是统一编址，绝对地址访问可以涵盖片内 RAM、片外 RAM、程序 ROM（或 Flash）及 I/O 接口，通过变量定义可以对单片机的硬件资源进行描述。如前所述，通常情况下 C51 在定义变量时只需说明其数据类型、存储类别（可缺省，默认为 auto）、存储器类型（可缺省，small 编译模式下默认为 data），变量的地址由编译器自动分配，但描述一些特定硬件资源时，必须指定其绝对地址。例如，单片机系统扩展了某个 I/O 模块，该模块的绝对地址由其与单片机的硬件连接决定，对该模块的访问须指定访问单元的绝对地址。又例如，单片机系统的配置参数一般保存在非易失存储器的指定

单元中，要读取这些重要参数须指定访问单元的绝对地址。C51 访问绝对地址单元常使用“绝对宏”和“_at_关键字”，由于指定了变量的绝对地址，所定义的变量只能是全局变量，无须说明其存储类别。

1. 绝对宏

在 Keil C51 编译器的安装目录下有一个头文件 Keil\C51\INC\ABSACC.H，该文件中定义了一组访问绝对地址的宏，可对 code、data、pdata、xdata 4 种类型的存储器进行绝对寻址，相关定义如下：

```
#define CBYTE ((unsigned char volatile code * ) 0)
#define DBYTE ((unsigned char volatile data * ) 0)
#define PBYTE ((unsigned char volatile pdata * ) 0)
#define XBYTE ((unsigned char volatile xdata * ) 0)

#define CWORD ((unsigned int volatile code * ) 0)
#define DWORD ((unsigned int volatile data * ) 0)
#define PWORD ((unsigned int volatile pdata * ) 0)
#define XWORD ((unsigned int volatile xdata * ) 0)
```

CBYTE、DBYTE、PBYTE、XBYTE 是以字节(8 位)形式分别对 code 区、data 区、pdata 区、xdata 区绝对寻址的宏，CWORD、DWORD、PWORD、XWORD 是以字(16 位)形式分别对 code 区、data 区、pdata 区、xdata 区绝对寻址的宏。可以用绝对宏对 RAM、ROM 或 I/O 口进行定义和操作，例如：

```
#include <absacc.h>
#define PORT XBYTE[0xffc0]          //PORT 定义为外部扩展 I/O 口，地址 0xffc0，8 位
void main(void)
{unsigned char data x;              //在片内 data 区定义局部变量 x
     PORT = 0xa5;                  //将数据 0xa5 写入 PORT 口
     x = CBYTE[0x1000];            //将程序 ROM 区 0x1000 地址单元内容传给 x
}
```

2. _at_关键字

使用_at_关键字指定变量在存储空间的绝对地址，一般格式如下：

```
数据类型 [存储器类型] 变量名 _at_ 地址常数;
```

所定义的变量为未初始化变量(不能既定义又初始化)。如果使用_at_关键字声明访问 xdata 外设的变量，则需要 volatile 关键字，以确保 C 编译器不会优化必要的内存访问。例如：

```
#define uchar unsigned char          //定义宏名 uchar，表示 8 位无符号整型
uchar code p1 _at_ 0x1000;           //在 CODE 区定义 8 位整型变量，地址 0x1000
uchar volatile xdata PA _at_ 0x8000; //在 XDATA 区定义 8 位 I/O 端口，地址 0x8000
void main(void)
{uchar data y;                       //在片内 data 区定义局部变量 y
```

```
    y = p1;                          //将 CODE 区 0x1000 地址单元内容传给 x
    PA = (y&0xf0)|0x05;              //P1 高 4 位保留、低 4 位改为 5,之后从 PA 口输出
}
```

3.4　C51 语言的基本语句

3.4.1　基本运算

C51 语言的基本运算与 ANSI C 语言完全相同，有赋值运算、算术运算、自加减运算、位运算、复合赋值运算、关系运算、逻辑运算、逗号运算等。

1. 赋值运算

赋值运算符“＝”将一个数据赋给一个变量，例如：

```
P1 = 0x5a;                           //将十六进制数据 0x5a 从并行口 P1 输出
```

2. 基本算术运算

基本算术运算符都是双目运算符，共有“＋”(加)、“－”(减)、“＊”(乘)、“/”(除)和“％”(模)5 个运算符。

3. 自加与自减运算

“＋＋”(自加)与“－－”(自减)运算符是单目运算符，是变量自加或自减 1 的操作。有“先自加自减”还是“后自加自减”之分，例如：

```
i = (j++) + k;                       //先将 j + k 的值赋给 i,然后 j 再自加 1
i = (++j) + k;                       //j 先自加 1,然后将 j + k 的值赋给 i
```

采用自加自减运算编程，可提高代码效率和运算速度。

4. 位运算

对 char 和 int 型数据，可以按二进制位进行操作运算，位运算有“～”(按位求反)、“&”(按位求与)、“|”(按位求或)、“^”(按位求异或)、“<<”(指定次数的按位左移)和“>>”(指定次数的按位右移)。执行按位左移(或右移)操作时，从低位(或高位)补 0。在 C51 语言中，最后移出的位进入了 PSW 中的进位标志位 CY。

为适应控制领域的应用，单片机重要特点之一是可进行位处理。在 C51 语言控制类程序设计中，普遍使用位运算，并占相当比例的代码量。

5. 复合赋值运算

赋值运算可以和算术运算、位运算结合构成复合的赋值运算，有 10 种复合赋值运算：＋＝、－＝、＊＝、/＝、％＝、&＝、|＝、^＝、<<＝、>>＝。例如：

```
unsigned char x = 0xe5;
x <<= 2;                             //对 x 的值(11100101)左移 2 位,结果 x = 0x94,CY = 1
```

“与”运算的特点是：和 0 与，结果为 0；和 1 与，结果不变。C51 语言中常用这个特点

将 char 或 int 型变量的某位或某几位清 0,例如:

```
unsigned char x;
x &= 0xf0;                              //x 高半字节(或高 4 位)保持,低半字节清 0
P1 &= (1<<7)|(1<<6)|(1<<3);             //P1 口第 7、6 和 3 位保持不变,其他位清 0
P1 &= ~((1<<7)|(1<<6)|(1<<3));          //P1 口第 7、6 和 3 位清 0,其他位保持不变
```

“或”运算的特点是:和 0 或,结果不变;和 1 或,结果为 1。C51 语言中常用这个特点将 char 或 int 型变量的某位或某几位置 1,例如:

```
P2 |= (1<<6)|(1<<4)|(1<<2);             //P2 口第 6、4 和 2 位置 1,其他位不变
```

“异或”运算的特点是:和 0 异或,结果不变;和 1 异或,结果为求反。C51 语言中常用这个特点将 char 或 int 型变量的某位或某几位置求反,例如:

```
P3 ^= (1<<5)|(1<<2)|(1<<0);             //P3 口第 5、2 和 0 位求反,其他位不变
```

6. 关系运算

关系运算即比较运算,用于比较操作数的大小关系。C51 语言有 6 种关系运算符:<(小于)、<=(小于或等于)、>(大于)、>=(大于或等于)、==(等于)、!=(不等于),其中前 4 种运算符<、<=、>、>=优先级相同,为高优先级;后 2 种运算符==、!=优先级相同,为低优先级。关系运算符的优先级低于算术运算符的优先级,高于赋值运算符的优先级。关系表达式的值为逻辑值,只有真(逻辑 1 或非 0 值)和假(逻辑 0)两种取值。

7. 逻辑运算

逻辑运算是对逻辑变量进行逻辑与、逻辑或、逻辑非 3 种运算,其 C51 语言运算符为 &&(逻辑与)、||(逻辑或)、! (逻辑非),其中逻辑非运算的优先级最高,高于算术运算符;或逻辑运算的优先级最低,低于关系运算符,但高于赋值运算符。逻辑表达式的值也是逻辑量,只有真和假两种取值。

3.4.2 分支判断语句

C51 语言的分支判断语句与 ANSI C 的分支判断语句完全相同,有条件语句和开关语句。

1. 条件语句

条件语句使用关键字 if,C51 提供了 3 种形式的条件语句,如图 3.1 所示。

第一种形式如图 3.1(a)所示,若逻辑表达式结果为真(逻辑 1 或非 0 值),则执行后面的语句;若逻辑表达式为假(逻辑 0),则不执行后面的语句,其语法格式为:

```
if (逻辑表达式){语句;}
```

第二种形式如图 3.1(b)所示,若逻辑表达式结果为真(逻辑 1 或非 0 值),则执行语句 1;若逻辑表达式为假(逻辑 0),则执行语句 2,其语法格式为:

```
if (逻辑表达式){语句 1;}
else {语句 2;}
```

第三种形式如图 3.1(c)所示，用于实现多条件分支，其语法格式为：

```
if (逻辑表达式 1){语句 1;}
else if (逻辑表达式 2){语句 2;}
⋮
else if (逻辑表达式 n){语句 n;}
else {语句 m;}
```

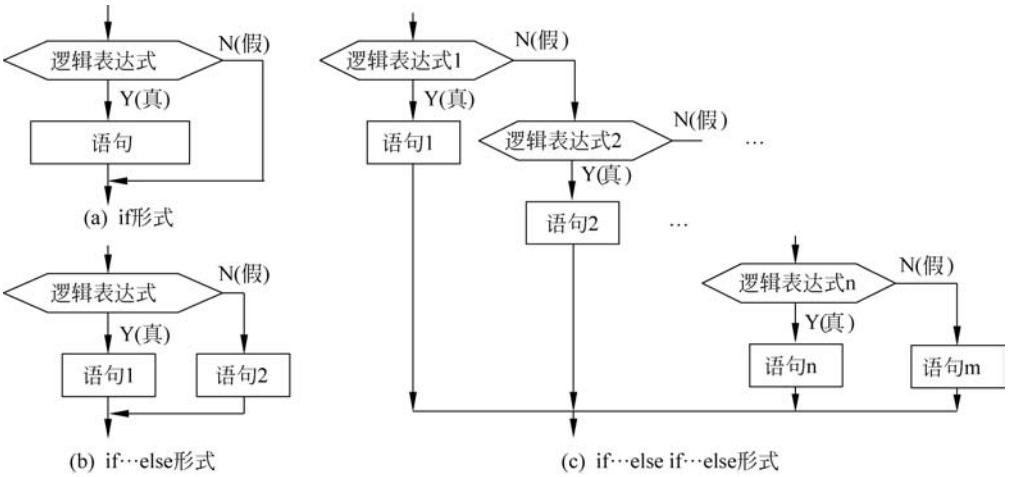

图 3.1　if 语句的 3 种形式

此处所说的语句可以是组合语句。例如：

```
unsigned char x,y,max,min;
if (x>=y){max=x;min=y;}
else {max=y;min=x;}
```

2. 开关语句

开关语句使用关键字 switch，实现多分支选择，其语法格式为：

```
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 m};
}
```

其中，switch()内的表达式可以是整型或字符型表达式，也可以是枚举型数据，每个 case 和 default 后面的语句的花括号{}可以省略。开关语句将 switch()中的表达式的值与 case 后面的各个常量表达式的值逐一进行比较，若与某个 case 后面的常量表达式的值匹配，则执行其后的语句，遇到 break 语句时，中止执行，跳出 switch 语句；如果该 case 中没有 break

语句，那么将会顺序执行下一个 case 后的语句；若无匹配情况，则执行 default 后的语句。

3.4.3 循环控制语句

C51 语言的循环控制语句与 ANSI C 的循环控制语句完全相同，有 while 语句、do…while 语句和 for 语句。

1. while 语句

如图 3.2(a)所示，while 语句用来实现“当型”循环，其语法格式为：

```
while (逻辑表达式){语句};
```

其中 while()后的语句称为循环体语句，当 while()中的逻辑表达式的结果为真(逻辑 1 或非 0 值)时，则执行循环体语句；反之则终止 while 循环，向后执行其余程序。如果逻辑表达式的结果一开始就为假，那么循环体语句一次也不会执行。

2. do…while 语句

如图 3.2(b)所示，do…while 语句用来实现“直到型”循环，其语法格式为：

```
do{语句;}while (逻辑表达式);
```

其中 do 之后的语句称为循环体语句。do…while 结构先执行循环体语句，然后检查逻辑表达式的结果，为真(逻辑 1 或非 0 值)则重复执行循环体语句，直到逻辑表达式的结果变为假(逻辑 0)时为止，循环体语句至少执行一次。

3. for 语句

如图 3.2(c)所示，for 语句是最为灵活、复杂的循环控制语句，其语法格式为：

```
for (表达式 1; 逻辑表达式 2; 表达式 3){语句;}
```

其中 for()之后的语句称为循环体语句。for 语句执行过程如下：先求解表达式 1，初始化循环；然后检查逻辑表达式 2 的结果，为真(逻辑 1 或非 0 值)则执行循环体语句并求解表达式 3，为假则退出循环。

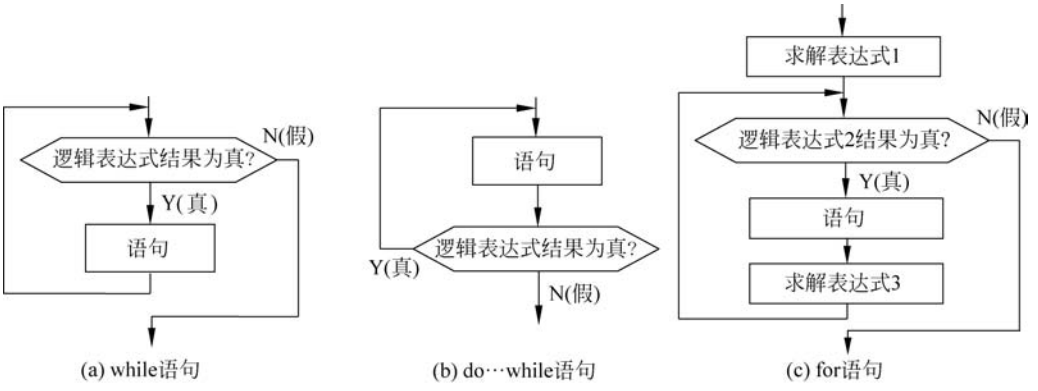

图 3.2 循环控制语句结构

单片机用户程序开发时，经常使用循环语句构成查询等待、延时、无限循环等程序模块，分别举例说明。

【例 3.2】 如图 3.3(a)所示，51 单片机的 I/O 引脚 P3.2 外接按键开关 K1，试编程一程序段实现等待键抬起，如图 3.3(b)所示。

由图 3.3(a)可知，键 K1 按下时，I/O 引脚 P3.2 为低电平，抬起时为高电平，因此等待键抬起的程序段如下：

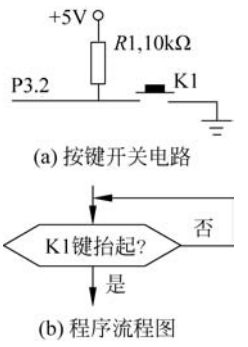

图 3.3　例 3.2 图

```
#include<reg51.h>                    //引用单片机资源定义头文件
sbit K1 = P3^2;                      //定义 K1 键位变量
…
void main(void)
{    …;
     while(!K1);                     //等待 K1 抬起
     …;
}
```

【例 3.3】 延时程序设计。

单片机用户程序常用的延时函数可以使用 while 语句，也可使用 for 语句。

(1) 使用 while 语句的延时函数。

```
void delay(unsigned int td)          //td 为控制延时时长的形参
{    while(td--);                    //判断 td,不为 0 则自减 1 直到为 0
}
```

(2) 使用 for 语句的延时函数。

```
void delay(unsigned int td)          //td 为控制延时时长的形参
{    unsigned char i,j;
     for (i = td;i > 0;i--)          //判断 td,不为 0 则自减 1 直到为 0
     {    for (j = 0xff;j > 0;j--);
     }
}
```

【例 3.4】 无限循环程序设计。

单片机用户程序的整个结构一般是个无限循环程序，使用 while 和 for 均可实现无限循环。

(1) 使用 while 语句的无限循环。

```
while(1)                             //逻辑表达式的值恒为"非 0"
{    …;                              //循环体
}
```

(2) 使用 for 语句的无限循环。

```
for ( ; ; )                          //逻辑表达式 2
{    …;                              //循环体
}
```

3.4.4 goto 等语句

1. goto 语句

goto 语句是无条件转移语句，其一般形式为：

```
goto 标号;
…
标号:语句;
```

其中标号是一个标志符，程序中存在以该标号带冒号“:”开头的某语句，执行 goto 语句时，程序将无条件转移到给定的标号处，执行其后的语句。C51 程序常用 goto 语句跳出多重循环，且只能从内层循环跳到外层循环，不允许从外层循环跳到内层循环。在结构化的程序设计中使用 goto 语句容易导致程序混乱，在 C51 语言程序设计中应尽量避免使用该语句。

2. break 语句

break 语句也是一个无条件转移语句，只能用于开关语句和循环语句之中，其一般形式为：

```
break;
```

执行 break 语句程序将退出开关语句或循环语句，执行其后的语句。对于多重循环，break 语句只能跳出其所在的那一层循环，而 goto 语句可以直接从最内层循环跳出来。

3. continue 语句

continue 语句也是一个无条件转移语句，只用于循环语句之中，其一般形式为：

```
continue;
```

执行 continue 语句程序将中止本轮循环，程序从下一轮循环开始处继续执行，直到循环条件不满足(逻辑表达式的值为假)为止，结束循环。

4. return 语句

return 语句用于终止函数的执行，控制程序返回到调用该函数处，其一般形式为：

```
return [表达式];
```

如果 return 语句后边有表达式，则返回时要计算表达式的值，并将其作为函数的返回值；如果不带表达式，则函数返回时，函数值不确定。一个函数内部可以有多个 return 语句，但程序仅执行其中的一个 return 语句返回调用处。函数也可以不含 return 语句，程序执行到最后一个界限符“}”时，返回调用处。

3.5 C51 语言的数组、指针、函数

3.5.1 数组

数组是一组数目一定、类型相同的数据的有序集合，用一个名字标记，称为数组名，其中的数据称为数组元素，数组元素的数据类型为该数组的基本类型。例如，字符型(char)变量

的有序集合称为字符型(char)数组,整型(int)变量的有序集合称为整型(int)数组。C51 语言不能定义 bit 类型数组。

数组中各元素的顺序用下标表示,下标为 n 的元素表示为“数组名[n]”。只有一个下标的数组称为一维数组,有两个(或多个)下标的数组称为二维(或多维)数组。C51 语言中常用一维数组、二维数组,除了说明数组元素的存储器类型外,定义数组的方法与 ANSI C 相同。

1. 一维数组

定义一维数组的一般形式如下:

```
数据类型 [存储器类型] 数组名[元素个数];
```

其中数组名是用户标志符,元素个数是一个整型常量表达式。例如:

```
unsigned char data DispBuf[8];
```

该语句在 DATA 区定义名为 DispBuf 的数组,含 8 个无符号字符型元素。定义数组时可同时对数组进行整体初始化,若定义后需给数组赋值,只能逐个对元素赋值。例如:

```
char b[4] = {1,2,3,4};            //全部初始化,b[0] = 1,b[1] = 2,b[2] = 3,b[3] = 4
int a[6] = {1,2,3};               //部分初始化,a[0] = 1,a[1] = 2,a[2] = 3,未初始化的元素为 0
char s[] = {"this is string"};    //定义一维字符数组,共 15 个元素,最后一个是"\0"
```

2. 二维(或多维)数组

二维数组定义的一般形式如下:

```
数据类型 [存储器类型] 数组名[行数] [列数];
```

其中行数和列数都是整型常量表达式,例如:

```
char c[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};  //全部初始化
int d[3][4] = {{1,2,3,4},{5,6,7,8}};                //部分初始化,未初始化的元素为 0
char s[3][6] = {"length","width","height"};          //定义二维字符数组
```

3. 用数组实现查表

在基于单片机的嵌入式控制系统中,经常需要按某已知规律实现特定的变换,例如,控制 DA 转换器(数字量到模拟量的转换)生成正弦波信号,或将热敏电阻(非线性元件)的电阻值变换为温度值。由于单片机运算能力有限,所以人们通常用查表法取代复杂数学计算,以实现特定的变换,查表法代码量少而且运行速度快。

【例 3.5】 用 8 位 DA 转换器生成 32 点的正弦信号,该正弦信号可如下表示:

$$x(n)=\mathrm{INT}[128+127\sin(2\pi n/N)] \tag{3.1}$$

其中 $N=32$,$n=0,1,\cdots,31$。

按式(3.1)计算出所有 32 点的正统信号值,如表 3.8 所示。在 CODE 区定义无符号字符型数组 xsin[32]保存数字正弦信号。

表 3.8　32 点 8 位数字正弦信号表

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$x(n)$	128	152	176	198	217	233	245	252	255	252	245	233	217	198	176	152
n	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$x(n)$	128	103	79	57	38	22	10	3	1	3	10	22	38	57	79	103

```
unsigned char code xsin[32] =                          //在 CODE 区定义 xsin[32]
{   128,152,176,198,217,233,245,252,255,252,245,233,217,198,176,152,
    128,103,79,57,38,22,10,3,1,3,10,22,38,57,79,103
};
```

3.5.2　指针

指针是存放变量(或存储器)地址的变量,与 ANSI C 不同,C51 语言定义指针需要说明指针所指变量的存储器类型和指针自身的存储器类型,其声明格式如下:

```
数据类型 [存储器类型 1] * [存储器类型 2] 指针名;
```

其中数据类型和存储器类型 1 说明指针所指变量的数据类型和存储器类型,存储器类型 2 说明指针自身的存储器类型。定义指针时,如果未给出指针所指变量的存储器类型(存储器类型 1 缺省),则称为一般指针;若给出指针所指变量的存储器类型,则称为基于存储器的指针;若未给出指针自身的存储器类型(存储器类型 2 缺省),则由编译模式选择默认存储器类型(如表 3.7 所示)。C51 语言不能定义 bit 类型的指针。

1. 基于存储器的指针

基于存储器的指针定义时,指针所指对象具有明确的存储器空间。如表 3.6 所示,若指针所指对象的存储器空间为 data 型、bdata 型、idata 型或 pdata 型,存储单元地址均为 1 字节,则这种指针占 1 字节;若指针所指对象的存储器空间为 xdata 型或 code 型,存储单元地址均为 2 个字节,则这种指针占 2 字节。例如:

```
char xdata * px;          //定义指向 XDATA 区的 char 型数据指针 px,其在默认存储区占 2 字节
int data * data pd;       //定义指向 DATA 区的 int 型数据指针 pd,其在 DATA 区占 1 字节
px = 0x1000;              //px 存入 XDATA 区存储单元地址 0x1000
 * px = 0xa5;             //px 指向的单元(XDATA 区地址 0x1000)存入 0xa5(char 型数据)
pd = 0x30;                //pd 存入 DATA 区存储单元首地址 0x30
 * pd = 0x5ac3;           //pd 指向的单元(DATA 区首地址 0x30)存入 0x5ac3(int 型数据)
```

2. 一般指针

一般定义指针时,若没有说明指针所指对象的存储器空间,则这种指针占 3 字节,第 1 字节保存所指对象的存储器类型编码(如表 3.9 所示),第 2 字节和第 3 字节分别保存所指对象的高位和低位地址。例如:

```
char * spt;                  //定义指向 char 型数据的一般指针 spt,其在默认存储区占 3 字节
char * data dpt;             //定义指向 char 型数据的一般指针 dpt,其在 DATA 存储区占 3 字节
spt = (char code * )0x1000; //spt 指向 CODE 区 0x1000 单元,其 3 字节存放 0xff1000
dpt = (char data * )0x1f;    //dpt 指向 DATA 区 0x1f 单元,其 3 字节存放 0x00001f
* dpt = * spt;               //将 CODE 区 0x1000 单元内容读出并存入 DATA 区 0x1f 单元
```

表 3.9　一般指针的存储器类型编码

存储器类型	data/bdata/idata	xdata	pdata	code
编码值	0x00	0x01	0xfe	0xff

3.3 节中使用绝对宏进行存储器绝对地址访问,所用的宏即为指向 0 基址的基于存储器的常数指针,例如:

```
#define CBYTE ((unsigned char volatile code * ) 0)
```

此即指向 CODE 区基址为 0 的字符型常数指针。

一般指针长度长,系统需根据指针首字节的存储器类型编码选择不同的寻址方式访问所指对象,程序运行速度慢,使用一般指针可以访问任意对象,兼容性高,许多 C51 的库函数都采用一般指针。基于存储器的指针长度短,既节省存储器空间运行速度又快,使用该指针只能指向特定的存储空间,兼容性较低。采用基于存储器的指针作为自定义函数的参数,应在程序的开始处直接给出函数的原型声明,或用预处理命令“#include”将函数原型说明文件包含进来,否则编译系统会自动将基于存储器的指针转换为一般指针,从而导致错误。

3.5.3　函数

1. 函数的定义

函数是 C51 语言的重要内容,C51 语言编译器继承了 ANSI C 的函数定义方法,并在选择函数的编译模式、选择函数所用工作寄存器组、定义中断服务函数、指定再入方式等方面进行了扩展。C51 语言定义函数的一般格式为:

```
函数类型 函数名([形式参数表])[编译模式][reentrant][interrupt n][using m]
{    局部变量定义;
     函数体语句;
}
```

其中函数类型、函数名、形式参数表、局部变量定义、函数体语句 5 部分继承 ANSI C 函数定义相关语法,编译模式、reentrant、interrupt n、using m 这 4 个可选项是 C51 语言的扩展。

(1) 函数类型说明函数返回值的类型,函数无返回值时其类型为 void(空),与 ANSI C 不同,C51 语言中函数类型可以为 bit 型。

(2) 函数名是用户自定义的函数标志符。

(3) 形式参数表列出主调用函数与被调用函数之间传递数据的形式参数,形式参数必须有数据类型说明,定义无形式参数函数时圆括号不能省略。

(4) 局部变量定义是对函数内部使用的临时变量进行定义。

(5) 函数体语句为实现自定义函数特定功能而设置的各种语句。

(6) 编译模式有 Small、Compact、Large 共 3 种选择,用于说明函数内部局部变量和参数的默认存储器类型,各编译模式的默认存储器类型如表 3.7 所示。

(7) reentrant 选项用于定义可再入函数(或可重入函数)。

(8) interrupt n 选项用于定义中断服务函数,其中 n 为中断号,可取值 0～31,详见第 6 章。

(9) using m 选项用于确定函数所用的工作寄存器组,其中 m 为所选工作寄存器组编号,可取值 0～3。

调用有"using m"选项的函数时,将完成以下操作:进入函数时将当前 PSW(程序状态字)的值压入堆栈保护,根据 m 值,更改 PSW 中的工作寄存器组选择位 RS1 和 RS0,执行函数体语句,退出函数时从堆栈中弹出数据恢复 PSW。使用"using m"选项声明的函数时,函数既不能通过寄存器返回其值,也不能返回 bit 值。非中断服务函数应慎用"using m"选项,以免发生错误。

2. 可再入函数

可再入函数(或称可重入函数)可以由多个进程共享。所谓共享,即当一个进程正在执行一个可再入函数时,另一个进程可以中断该进程,然后执行同一个可再入函数,而不会影响函数的运行结果。

调用函数时,ANSI C 会将调用参数和函数所用局部变量压入堆栈保护,递归调用仅使用局部变量的函数时,ANSI C 函数总是可再入的。与 ANSI C 不同,C51 堆栈使用片内 RAM,堆栈很浅,调用函数时,使用固定的存储空间(称为局部数据区)传递参数和保护局部变量,递归调用将导致局部数据区被覆盖,因此 C51 函数一般是不可再入的。

C51 语言必须用 reentrant 关键字声明函数为可再入的,C51 编译器在默认的存储空间(由编译模式确定)中为可再入函数创建一个模拟堆栈,实现函数调用时的参数传递与局部变量入栈保护,解决数据覆盖问题。可再入函数一般占用较大内存空间,不允许传递 bit 型参数,不能使用 bit 型局部变量,运行速度也较慢。

中断服务函数和非中断服务函数共同调用的函数必须声明为可再入函数,否则将导致不可预测的结果。

3.6 C51 语言的预处理命令

与 ANSI C 类似,一个 C 文件经过编译后才能形成可执行的目标文件,编译过程有预处理、编译、汇编和连接共 4 个阶段。C51 语言也提供了预处理命令,在 C 文件中以"#"开头的命令被称为预处理命令,有宏定义 #define、包含 #include,以及条件编译 #if、#ifdef 等。对 C51 源程序进行编译时,首先进行预处理,将要包含的文件插入源程序中、将宏定义展开、根据条件编译命令选择要使用的代码,然后将预处理的结果和源代码一并进行编译,最

后生成目标代码。

3.6.1　宏定义

宏定义指令即#define命令，以指定的标志符作为宏名来替代一个字符串的预处理命令。使用宏定义指令，可以减少程序中字符串输入的工作量，而且可以提高程序的可移植性。宏定义分为简单的宏定义和带参数的宏定义，其格式分别为：

```
#define 宏名 宏替换体
#define 宏名(形参) 带形参的宏替换体
```

其中#define是宏定义指令的关键词，宏名即指定的标志符，一般使用大写字母表示，宏替换体可以是数值常量、自述表达式、字符和字符串等。使用宏定义命令时，应注意以下几点：

(1) 宏定义可以出现在程序的任何地方，通常“#define”命令写在文件开头，函数之前，作为文件的一部分，在此文件内有效，或直到用“#undef”命令终止该宏定义处有效。

(2) 在编译过程的预处理阶段，编译器将宏定义有效作用域内的所有宏名用宏替换体替换，带参数时，形参用实参替换。进行宏定义时可以引用已定义的宏名，预处理时实现逐层替换。

(3) 带参数的宏定义时，宏替换体内的形参一定要带括号，因为实参可能是任意表达式，形参若不加括号可能导致其用实参替换后所得表达式与设计者的原意不符。

(4) 宏名引用只是字符串替换，带参数的宏名展开时不分配内存单元，既不进行数值的传递，也没有“返回值”的概念，因此宏定义不存在类型问题，宏名、形参和实参都没有类型。

3.6.2　文件包含

文件包含指令即#include命令。所谓包含，是一个程序文件将另一个指定文件的全部内容包含进来。文件包含的一般格式为：

```
#include <文件名>或#include "文件名"
```

例如，“#include < stdio. h >”就是将C51编译器提供的输入/输出库函数的说明文件stdio. h包含到用户程序中。进行大规模程序设计时，往往将程序的各功能模块函数分散到多个程序文件中，各文件由团队成员分工完成，最后再由包含命令嵌入到一个总的程序文件中。使用包含命令时，应注意以下几点：

(1) “#include”命令出现在程序中的位置，被包含文件就从该位置插入。一般情况下，被包含的文件要放在程序文件的前面，否则可能会出现内容尚未定义的错误。

(2) 需包含多个文件时，要用多个包含命令，每个#include命令只指定一个文件。

(3) 采用<文件名>格式时，在头文件目录(即Keil\C51\INC目录)中查找指定文件；采用“文件名”格式时，在当前目录中查找指定文件。

3.6.3 条件编译

一般情况下，源程序中所有代码行都参与编译，但有时希望程序中某些功能模块只在满足一定条件时才参与编译，这就是条件编译。与 ANSI C 类似，C51 语言编译器提供＃if、＃ifdef 和＃ifndef 共 3 种条件编译预处理指令。

1. ＃if 型的基本格式

```
＃if 常量表达式
代码段 1;
＃else
代码段 2;
＃endif
```

如果常量表达式为非 0，则编译代码段 1，否则编译代码段 2。

2. ＃ifdef 型的基本格式

```
＃ifdef 标志符
代码段 1;
＃else
代码段 2;
＃endif
```

如果本组条件编译命令之前指定标志符已用宏定义＃define 指令定义过（不论定义为何字符串），则编译代码段 1，否则编译代码段 2。

3. ＃ifndef 型的基本格式

```
＃ifndef 标志符
代码段 1;
＃else
代码段 2;
＃endif
```

与＃ifdef 型相反，如果本组条件编译命令之前指定标志符没有用宏定义＃define 指令定义过，则编译代码段 1，否则编译代码段 2。

这里的代码段 1 或 2 既可以是 C51 语言语句（以“;”结束），也可以是预处理指令语句（不以“;”结束）。在以上 3 种类型的条件编译指令中，＃else 分支又可以再带自己的编译选项，形成多分支的条件编译；当然＃else 分支也可以没有。

条件编译在单片机用户程序设计中非常有用，单片机应用系统常有多种硬件配置，使用条件编译可使一套软件适应不同硬件配置，且不增加编译后的形成的可执行目标代码的代码量和运行速度。

3.7 C51 语言的库函数

C51 语言提供了丰富的可直接调用的库函数，使用库函数可使程序代码简单、结构清晰、易于调试和维护，体现 C51 功能强大、高效等优点。每个库函数都在相应的头文件中给

出了函数原型声明，调用库函数之前，必须在源程序开始处使用预处理命令 # include 将相应的头文件包含进来，C51 语言主要函数库及其相应的头文件如表 3.10 所示。

表 3.10　C51 主要函数库及其相应的头函数

函数库名称	函数原型声明头文件	函数库名称	函数原型声明头文件
本征函数库	intrins. h	数学函数库	math. h
输入/输出函数库	stdio. h	绝对地址访问函数库	absacc. h
字符函数库	ctype. h	变量参数表函数库	stdarg. h
字符串函数库	string. h	全跳转函数库	setjmp. h
标准函数库	stdlib. h	偏移量函数库	stddef. h

C51 程序设计时经常使用本征函数库、输入/输出函数库和数学函数库，分别介绍如下。

3.7.1　本征函数库

本征函数是指编译时直接将固定的代码插入到当前行，而不是采用子程序调用方式实现函数功能，避免了堆栈操作，大大提高了函数的访问效率。本征函数库提供了循环移位和延时操作等函数，该函数库中的函数如表 3.11 所示。

表 3.11　本征函数库的函数及其功能(intrins. h)

循环左移函数	原型	unsigned char _crol_(unsigned char val, unsigned char n);
crol		unsigned int _irol_(unsigned int val, unsigned char n);
irol		unsigned long _lrol_(unsigned long val,uchar n);
lrol	功能	将字符型、整型、长整型数据 val 循环左移 n 位，相当于 RL 指令
循环右移函数	原型	unsigned char _cror_(unsigned char val, unsigned char n);
cror		unsigned int _iror_(unsigned int val, unsigned char n);
iror		unsigned long _lror_(unsigned long val, unsigned char n);
lror	功能	将字符型、整型、长整型数据 val 循环右移 n 位，相当于 RR 指令
位测试函数	原型	bit _testbit_(bit x);
testbit	功能	相当于 JBC bit 指令
浮点数检查函数	原型	uchar _chkfloat_(float ual);
chkfloat	功能	测试并返回浮点数状态
延时函数	原型	void _nop_(void);
nop	功能	使单片机产生延时，相当于插入 NOP 指令

3.7.2　输入/输出函数库

输入/输出函数主要用于通过串口的数据传输操作，由于这些函数使用 51 单片机的标准串口，因此调用之前应对串口初始化，确保串口通信正常。串口初始化需设置串口模式、波特率和中断允许，8051 单片机串口初始化具体代码详见第 7 章。该函数库提供的库函数

及其功能如表 3.12 所示。

表 3.12 输入/输出函数库的函数及其功能(stdio.h)

字符读入输出函数 getchar	原型	char getchar(void);
	功能	从串口读入一个字符,并输出该字符
字符读入函数 _getkey	原型	char _getkey(void);
	功能	从串口读入一个字符(改变输入串口唯一需修改的函数)
字符串读入函数 gets	原型	char *gets(char *s,int n);
	功能	从串口读入一个长度为 n 的字符串并存入由 s 指向的数组
格式化输出函数 printf	原型	int printf(const char *fmstr[,…]);
	功能	按一定格式从串口输出数据或字符串
字符输出函数 putchar	原型	char putchar(char c);
	功能	从串口输出一个字符(改变输出串口唯一需修改的函数)
字符串输出函数 puts	原型	int puts(const char *s);
	功能	将字符串和换行符写入串口
格式化输入函数 scanf	原型	int scanf(const char *fmstr[,…]);
	功能	将字符串和数据按照一定格式从串口读入
格式化内存缓冲区输出函数尾 sprintf	原型	int sprintf(char *s, const char *fmstr[,…]);
	功能	按一定格式将数据或字符串输出到内存缓冲区
格式化内存缓冲区输入函数 sscanf	原型	int sscanf(char *s,const char *fmstr[,…]);
	功能	将格式化的字符串和数据送入数据缓冲区
字符回送函数 ungetchar	原型	char ungetchar(char c);
	功能	将读入的字符回送到输入缓冲区
字符串内存输出函数 vprintf	原型	void vprintf(const char *fmstr, char *argptr);
	功能	将格式化字符串输出到内存数据缓冲区
指向缓冲区的输出函数 vsprintf	原型	void vsprintf(char *s, const char *fmstr, char *argptr);
	功能	将格式化字符串和数字输出到内存数据缓冲区

3.7.3 数学函数库

数学计算库函数有绝对值、指数、对数、平方根、三角、反三角、双曲函数等常用函数的,数学函数库提供的库函数及其功能如表 3.13 所示。

表 3.13 数学函数库的函数及其功能(math.h)

绝对值函数 abs、cabs	原型	int abs(int val);	float fabs(float val);
		char cabs(char val);	long labs(long val);
fabs、labs	功能	用于计算并返回整型、字符型、浮点型、长整型数据的绝对值	
指数以及对数函数 exp、log	原型	float exp(float x);	float log10(float x);
		float log(float x);	float sqrt(float x);
log10、sqrt	功能	用于计算并返回指数、对数、以 10 为底的对数、平方根	

续表

三角函数 cos、sin、tan acos、asin、atan atan2 cosh、sinh、tanh	原型	float cos(float x)；	float atan(float x)；
		float sin(float x)；	float atan2(float y, float x)；
		float tan(float x)；	float cosh(float x)；
		float acos(float x)；	float sinh(float x)；
		float asin(float x)；	float tanh(float x)；
	功能	用于计算并返回余弦、正弦、正切、反余弦、反正弦、反正切、(y/x)反正切、双曲余弦、双曲正弦、双曲正切	
取整函数 ceil、floor	原型	float ceil(float x)；	float floor(float x)；
	功能	计算并返回浮点数的整数部分。ceil 不小于最小；floor 不大于最大	
浮点型分离函数 modf	原型	float modf(float x, float * ip)；	
	功能	将浮点数 x 分成整数和小数两部分，整数部分放入 * ip，返回小数部分	
幂函数 pow	原型	float pow(float x, float y)；	
	功能	计算并返回 xy	

3.7.4　其他函数库

除了前述的本征函数库 intrins.h、输入/输出函数库 stdio.h、数学函数库 math.h 外，C51 还有以下函数库：字符函数库 ctype.h、字符串函数库 string.h、标准函数库 stdlib.h、绝对地址访问函数库 absacc.h、变量参数表函数库 stdarg.h、全跳转函数库 setjmp.h、偏移量函数库 stddef.h，这些库函数请参见附录 C 和 Keil Software 公司的相关资料。

本章小结

单片机的 C 语言编程与单片机的硬件紧密联系。以第 2 章 51 单片机硬件基本知识为基础，本章主要介绍 51 单片机 C 语言编程涉及 C51 基本知识，包括 C51 语言的数据、对单片机主要资源的描述、基本运算、程序流程控制、数组、指针、函数、预处理等。读者必须掌握这些单片机 C 语言编程的基本知识，为编写简单的单片机 C 语言程序打下基础。

本章的叙述是以读者已初通 ANSI C 语言为前提的，如读者此前没有学习过 ANSI C 语言，请另补充阅读相关内容。

习题

3.1　C51 语言支持哪些变量类型？其中什么类型是 ANSI C 没有的？

3.2　C51 语言中整型 int、浮点型 float 数据各是几字节、几位？

3.3　简述 C51 语言的数据存储器类型。

3.4　简述 C51 语言对 51 单片机特殊功能寄存器的定义方法。

3.5　简述 C51 语言对 51 单片机片内 I/O 口和外部扩展 I/O 口的定义方法。

3.6 简述 C51 语言对 51 单片机位变量的定义方法。

3.7 简述 C51 语言定义指针的方法,指出其与 ANSI C 的不同点。基于存储器的指针和一般指针有什么不同?

3.8 简述 C51 语言的函数定义的方法,指出其与 ANSI C 的不同点。

3.9 按照给定的数据类型和存储器类型,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 code 区定义无符号字符型数组 val3[4]。

(4) 在 data 区定义一个指向 xdata 区无符号整型的指针 pi。

(5) 在 bdata 区定义无符号字符型变量 key,定义其第 0 位为 k0。

(6) 定义特殊功能寄存器变量 P3 口,定义其第 2 位端口 P3.2。