第3章 C51系列单片机程序设计

CHAPTER 3

应用于51系列单片机开发的C语言通常简称C51语言，C51语言与标准ANSI C语言相比，C51语言针对51单片机做了一定的扩展。本章首先介绍了C语言的特点，并与汇编语言、ANSI-C语言做了比较，然后重点介绍了C语言程序的格式和特点，数据类型、变量、运算符和表达式，指针和绝对地址的访问，C51函数的使用方法，最后通过单片机控制流水灯的C51程序设计实例来加深理解。

视频讲解

3.1 C51语言概述

从单片机引入中国时开始，汇编语言一直都是比较流行的开发工具。习惯汇编语言编程的人也许会认为，高级语言的控制性不好，不像汇编语言一样简单且功能强大。汇编语言有执行效率高、控制性强等优点，但它也有一些缺点：首先它的可读性不强，特别是当程序没有很好注解的时候；其次就是可移植性差，代码的可重用性比较差，这些使其在维护和功能升级方面有极大的困难。C语言可以克服这些缺点，在单片机开发所使用的高级语言中，最常见的就是C语言。

1. C语言与汇编语言的比较

使用C语言进行单片机系统的开发，有着汇编语言编程所不具备的优势，主要体现在以下几个方面。

(1) 不需要了解单片机指令集，也不需要了解其存储器结构。

(2) 寄存器分配和寻址方式由编译器进行管理，程序员可以忽略这些问题。

(3) 程序有规范的结构，可分为不同的系数，使程序结构化。

(4) 与使用汇编语言编程相比，程序的开发和调试时间大大缩短。

(5) C语言中的库文件提供了许多标准函数，如数学运算。开发者可以直接调用，而不必使用烦琐的汇编语言来实现。

(6) C语言可移植性好且非常普及，C语言编译器几乎适用于所有的目标系统。

(7) C语言在模块化开发、可移植性、代码管理上有明显的优势。

2. C51与ANSI-C的主要区别

目前最常见的编译器是Keil公司针对51系列单片机开发提供的C51编译器。

ANSI-C语言是一门应用非常普遍的高级程序设计语言，C51和标准的ANSI-C有一定的区别，或者说C51是对标准C语言的扩展。C51语言的特色主要体现在以下几个方面：

(1) C51 继承了标准 C 语言的绝大部分的特性,其基本语法相同,但 C51 本身又在特定的硬件结构上有所扩展,如定义了关键字 sbit、xdata、idate、code 等。

(2) 编译生成的 m51 文件,包含了硬件资源使用的情况。进行 C51 编程时可以通过该文件了解系统资源。

(3) C51 头文件体现了 51 单片机芯片的不同功能。只需要将相应的功能寄存器的头文件加载在程序内就可实现它们所指定的不同功能。

(4) C51 与标准 ANSI-C 在库函数方面来说有很大的不同。部分标准 C 的库函数,如字符屏幕和图形函数等,没有包含在 C51 内。有一些库函数虽然可以使用,但这些库函数的构成及用法都有很大不同,如 printf 和 scanf 这两个函数在 ANSI-C 中通常用于屏幕打印和接收字符;而在 C51 中,它们则主要用于串行数据的收发。


视频讲解

3.2　C51 程序的基本结构

总体而言,C 语言的程序均是由一个或多个函数(或子程序,function)构成,其程序入口处是以 main()开始的函数,其余函数都是直接或间接被 main()函数调用。这些函数就是组成 C 程序的模块。C51 程序同标准 C 程序一样,尽量在一个函数内完成较少的功能,而不同函数之间设置较少的接口参数,即高内聚、低耦合。C51 程序的基本结构如图 3-1 所示。

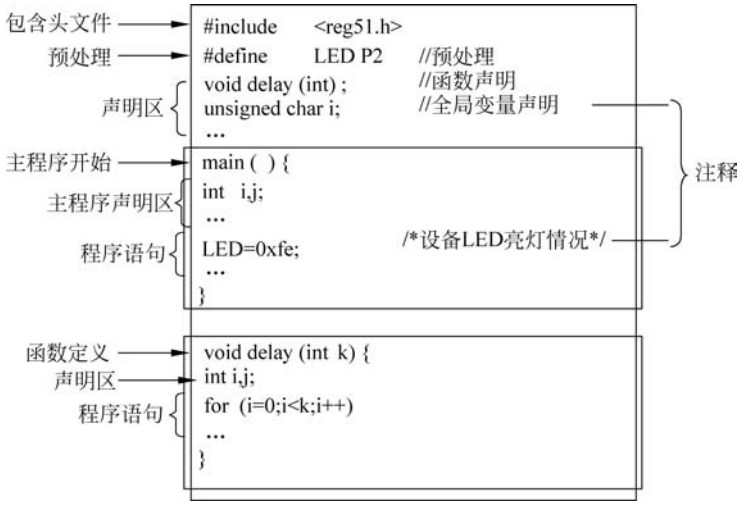

图 3-1　C51 程序的基本结构

3.3　数据类型

3.3.1　C51 数据类型

在标准 C 语言中基本的数据类型为 char、int、short、long、float 和 double,而在 C51 编译器中 int 和 short 相同;除此之外,C51 编译器还扩充了其特有的数据类型 bit、sbit、sfr 和

sfr16，如表 3-1 所示。

表 3-1　C51 的数据类型

数 据 类 型	长　　度	值　　域
unsigned char	单字节	0～255
signed char	单字节	－128～＋127
unsigned int	双字节	0～65 535
signed int	双字节	－32 768～＋32 767
unsigned long	四字节	0～4 294 967 295
signed long	四字节	－2 147 483 648～＋2 147 483 647
float	四字节	±1.175494E－38～±3.402823E＋38
double	八字节	双精度实型变量
bit	1 个二进制位	0 或 1
sbit	1 个二进制位	0 或 1
sfr	单字节	0～255
sfr16	双字节	0～65 535

1. 位数据类型 bit

bit 数据类型使用一个二进制位来存储数据，其值只有 0 和 1 两种。所有的位变量存储在 51 单片机内部 RAM 中的位寻址区，即 RAM 区的 0x20～0x2F 的地址，共计 128 个这样的地址。因此，程序中最多只能定义 128 个位变量。

1）位变量的定义

```
bit  变量名
```

例如，

```
bit flag = 0;              //定义一个位变量 flag
```

2）使用限制

bit 数据类型不能作为数组，例如，

```
bit a[10];                 //错误定义
```

bit 数据类型不能作为指针，例如，

```
bit *ptr                   //错误定义
```

使用禁止中断（#program disable）和明确指定使用工作寄存器组切换（using n）的函数不能返回 bit 类型的数据。

bit 型变量除了用于变量的定义，还可用于函数的参数传递和函数的返回值中。

2. SFR 型数据 sfr

为定义存取 SFR，C51 增加了 SFR 型数据，相应地增加了 sfr、sfr16 和 sbit 这 3 个关键字。sfr 是为了能够直接访问 51 单片机中的 SFR 所提供的一个新的关键词，其定义说明如下。

```
sfr  变量名 = 地址值;
```

例如，

```
sfr P1 = 0x90; sfr P2 = 0xA0; sfr PCON = 0x87; sfr TH0 = 0x8C;
```

3. SFR 型数据 sfr16

sfr16 是用来定义 16 位特殊功能寄存器的。对于标准的 8051 单片机，只有一个 16 位特殊功能寄存器，及 DPTR。其定义说明如下。

```
sfr16 DPTR = 0x82;
```

DPTR 是两个地址连续的 8 位寄存器 DPH 和 DPL 的组合。可以分开定义这两个 8 位寄存器，也可用 sfr16 定义 16 位寄存器。

4. SFR 型数据 sbit

在 C 语言中，如果直接写 P1.0 编译器不能直接识别，而且 P1.0 也不是一个合法的 C 语言标识符，所以必须给它起一个名，为它建立联系，可由 Keil C 增加的关键字 sbit 来定义。

sbit 的定义有以下 3 种。

(1) sbit 位变量名＝地址值。

(2) sbit 位变量名＝SFR 名称^变量位地址值。

(3) sbit 位变量名＝SFR 地址值^变量位地址值。

定义 PSW 中的 OV 标志位可以用以下 3 种方法。

```
(1) sbit   OV=0xD2;          //0xD2 是 OV 的位地址值
(2) sbit   OV=PSW^2;         //PSW 必须先用 SFR 定义
(3) sbit   OV=0xD0^2;        //0xD0 是 PSW 的地址值
```

以上是对 SFR 的位的定义。如果不是 SFR，则必须先使用 bdata 关键字定义这个变量后才能在该变量的基础上使用 sbit。

```
int bdata ibase;                    //位寻址区的 int 型变量
sbit mebit0 = ibase^0;              //ibase 的第 0 位
```

sbit 数据类型的地址是确定的且不用编译器分配。它可以是 SFR 中确定的可进行位寻址的位，也可以是内部 RAM 的 20H～2FH 单位中确定的位。例如，我们先前定义了 sfr P1＝0x90，即表示寄存器 P1 的地址是 0x90 地址，又因为寄存器 P1 是可位寻址的，所以：

```
sbit LED = P1^1;                    //声明 LED 为 P1 口的 P1.1 引脚
```

同样，可以用 P1.1 的地址去写，如：

```
sbit LED = 0x91;                    //同样声明 LED 为 P1 口的 P1.1 引脚
```

这样在后续的程序语句中就可以用 LED 来对 P1.1 引脚进行读写操作。

3.3.2　REG51.H 头文件

REG51.H 头文件是 51 单片机 C 语言编程时经常包含的头文件，在该文件中预先定义好了很多基本的数据。REG51.H 头文件的内容如下。

```
/* ------------------------------------------------------------
REG51.H
Header file for generic 80C51 and 80C31 microcontroller.
Copyright (c)1988 - 2002 Keil Elekronik GmbH and Keil Software,Inc.
All rights reserved.
------------------------------------------------------------ */

#ifndef_REG51_H_
#define_REG51_H_

/* BYTE Register */

sfr P0 = 0X80;                  //P0
sfr P1 = 0x90;                  //P1
sfr P2 = 0xA0;                  //P2
sfr P3 = 0xB0;                  //P3
sfr PSW = 0xD0;                 //程序状态寄存器
sfr ACC = 0xE0;                 //累加器 ACC
sfr B = 0xF0;                   //寄存器 B
sfr SP = 0x81;                  //堆栈指针寄存器
sfr DPL = 0x82;                 //16 位数据指针寄存器的低 8 位
sfr DPH = 0x83;                 //16 位数据指针寄存器的高 8 位
sfr PCON = 0x87;                //寄存器 PCON
sfr TCON = 0x88;                //寄存器 TCON
sfr TMOD = 0x89;                //寄存器 TMOD
sfr TL0 = 0x8A;                 //Timer0 计数器的低 8 位
sfr TL1 = 0x8B;                 //Timer1 计数器的低 8 位
sfr TH0 = 0x8C;                 //Timer0 计数器的高 8 位
sfr TH1 = 0x8D;                 //Timer1 计数器的高 8 位
sfr IE = 0xA8;                  //寄存器 IE
sfr IP = 0xB8;                  //寄存器 IP
sfr SCON = 0x98;                //寄存器 SCON
sfr SBUF = 0x99;                //寄存器 SBUF

/*     BIT Register               */
/*     PSW                        */
sbit CY = 0xD7;                 //进位位
sbit AC = 0xD6;                 //辅助进位位
sbit F0 = 0xD5;                 //用户进位位
sbit RS1 = 0xD4;                //寄存器组选择位 1
sbit RS0 = 0xD3;                //寄存器组选择位 0
sbit OV = 0xD2;                 //溢出位
sbit P = 0xD0;                  //校验位

/*     TCON                       */
sbit TF1 = 0x8F;                //Timer1 的溢出位
sbit TR1 = 0x8E;                //Timer1 的运行位
sbit TF0 = 0x8D;                //Timer0 的溢出位
sbit TR0 = 0x8C;                //Timer0 的运行位
sbit IE1 = 0x8B;                //INT1 的中断标志
```

```
sbit IT1 = 0x8A;                    //INT1 的触发信号种类
sbit IE0 = 0x89;                    //INT0 的中断标志
sbit IT0 = 0x88;                    //INT0 的触发信号种类

/*     IE                             */
sbit EA = 0xAF;                     //中断总开关
sbit ES = 0xAC;                     //串行的中断开关
sbit ET1 = 0xAB;                    //Timer1 的中断开关
sbit EX1 = 0xAA;                    //INT1 的中断开关
sbit ET0 = 0xA9;                    //Timer0 的中断开关
sbit EX0 = 0xA8;                    //INT0 的中断开关

/*     IP                             */
sbit PS = 0xBC;                     //串行中断高优先级设置位
sbit PT1 = 0xBB;                    //Timer1 中断高优先级设置位
sbit PX1 = 0xBA;                    //INT1 中断高优先级设置位
sbit PT0 = 0xB9;                    //Timer0 中断高优先级设置位
sbit PX0 = 0xB8;                    //INT0 中断高优先级设置位

/*     P3                             */
sbit RD = 0xB7;                     //RD 引脚
sbit WR = 0xB6;                     //WR 引脚
sbit T1 = 0xB5;                     //T1 引脚
sbit T0 = 0xB4;                     //T0 引脚
sbit  INT1 = 0xB3;                  //INT1 引脚
sbit  INT0 = 0xB2;                  //INT0 引脚
sbit TXD = 0xB1;                    //TXD 引脚
sbit RXD = 0xB0;                    //RXD 引脚

/*     SCON                           */
sbit SM0 = 0x9F;                    //串口方式设置位 0
sbit SM1 = 0x9E;                    //串口方式设置位 1
sbit SM2 = 0x9E;                    //串口方式设置位 2
sbit REN = 0x9C;                    //接收使能控制位
sbit TB8 = 0x9B;                    //发送的第 8 位
sbit RB8 = 0x9A;                    //接收的第 8 位
sbit TI = 0x99;                     //发送中断标志位
sbit RI = 0x98;                     //接收中断标志位
#endif
```

如果使用 Keil C 作为 C51 程序的开发环境，则该文件默认安装在“C:\Keil\C51\INC”路径中。

3.4　变量和 C51 存储区域

在程序运行中，其值可以改变的量称为“变量”。每个变量都有一个标识符作变量名。在使用一个变量前，必须先对该变量进行定义，指出它的数值类型和存储模式，以便编译系

统为它分配相应的存储单元。

变量与符号常量的区别——变量的值在程序运行过程中可以发生变化；而符号常量不等同于变量，它的值在整个作用域范围内不能改变，也不能被再次赋值。

3.4.1 变量的定义

在C语言中，要求对所有的变量“先定义，后使用”。格式如下。

[存储种类] 数据类型 [存储器类型] 变量名表

其中，存储种类和存储器类型是可选项。存储种类有自动(auto)、外部(extern)、静态(static)和寄存器(register)4种，系统默认为自动(auto)类型。

3.4.2 存储器类型

51单片机的存储类型较多，有片内程序存储器、片外程序存储器、片内数据存储器和片外数据存储器。其中，片内数据存储器又分为低128B和高128B，高128B只能用间接寻址方式来使用，低128B的数据存储器中又有位寻址区、工作寄存器区，这与其他CPU、MCU等有很大区别。

为充分支持51单片机的特性，C51中引入了一些关键字，用来说明数据存储位置。表3-2为Keil C51编译器所能识别的存储器类型。

表3-2 Keil C51编译器所能识别的存储器类型

存储器类型	说　明
code	程序存储器(64KB)，用“MOVC @A+DPTR”访问
data	直接访问片内数据存储器(128B)，访问速度最快
idata	间接访问片内数据存储器(256B)，允许访问全部内部地址
bdata	可位寻址片内数据存储器(16B)，允许位与字节混合访问
pdata	分页访问片外数据存储器(256B)，用“MOVX @Ri”访问
xdata	外部数据存储器(64KB)，用“MOVX @DPTR”访问

1. 程序存储器

程序存储器只能读，不能写，汇编语言中可以用MOVC指令来读取程序存储器中的数据。程序存储器除了存放代码外，往往还用于存放固定的表格、字型码等不需要在程序中进行修改的数据。程序存储器的容量最大为64KB。

在C51中，使用关键字code来说明存储在程序存储器中的数据。

例如，“code int x=100;”变量x的值100将被存储于程序存储器中。这个值不能被改变。

2. 内部数据存储器

内部数据存储器既可以读出，也可以写入。对于51系统而言，共用128B的内部数据存储器，而对于52系统而言，共用256B的内部数据存储器。

在低地址128B存储器中，0x20～0x2F的存储器是可以位寻址的。在52系统中，从0x80～0xFF的高128B RAM只能采用间接寻址方式进行访问，以便与同一地址范围的SFR区分开(SFR只能用直接寻址的方式访问)。

C51 引入了 3 个新的关键字：data、idata 和 bdata，用来表达内部数据存储器的 3 个不同部分，data 用于存取前 128B 的内部数据存储器；idata 使用全部的 26B；bdata 定义与位操作有关的变量。

3. 外部数据存储器

51 单片机可以扩展外部数据存储器，尤其是使用总线以后，外部 I/O 口和外部数据存储器也是统一编址，采用同一指令进行读/写。

外部数据存储器既可读也可写，读/写外部数据存储器的数据要比使用内部数据存储器慢，但外部数据存储器可达 64KB。

汇编语言中使用 MOVX 指令来对外部存储器中的数据进行读/写，C51 提供了两个关键字 pdata 和 xdata，可用于对外部数据存储器进行读/写操作。

(1) pdata 用于只有一页(256B)的情况。这相当于汇编指令中的下列指令。

```
MOV R1, #0x10
MOVX A, @R1
```

C51 通过关键字 pdata 定义使用外部 pdata RAM 的变量。例如，

```
unsigned char pdata c1;                    //定义一个变量 c1,它被放在外部 pdata 中
```

(2) xdata 可用于外部数据存储器最多可达 64KB 的情况，这相当于汇编指令中的下列指令：

```
MOV DPTR, #1000H
MOVX A, @ DPTR
```

例如，

```
unsigned int xdata c2;                     //定义一个变量 c2,它被放在外部 xdata 中
```

4. 定义时省略存储类型标志符

如果在变量定义时略去了存储类型的标识符，则编译器会自动选择默认的存储类型。

设一个变量定义“char c;”，c 被存放在何处与工程设置中 Target 选项卡的 Memory Model 设置有关。如果将 Memory Model 设置为 Small 模式，则变量 c 会被定位在 data 存储区中；若设置为 Compact 模式，则 c 被定位在 pdata 存储区中；若设置为 LARGE 存储模式，则 c 被定位在外部数据存储区中。

3.4.3 存储器模式

定义变量时如果省略存储器类型，Keil C51 编译系统会按编译模式 Small、Compact 或 Large 所规定的默认存储器类型去指定变量的存储区域。

1. Small 存储模式

该模式把所有函数变量和局部数据段存放在 51 单片机系统的内部数据存储区，因此对这种变量的访问数据最快。Small 存储模式的地址空间受限，在编写小型应用程序时，变量和数据放在 data 内部数据存储器中是很好的；但在较大的应用程序中，data 区最好只存放小的变量、数据或常用变量(如循环计数、数据索引)，大的数据放置在其他区域。

2. Compact 存储模式

该模式把变量定义在外部数据存储器中，所有默认变量均位于外部 RAM 区的一页（与显式使用关键字 pdata 进行定义，效果相同），外部数据存储器可最多有 256B（一页）。其优点是空间较 Small 宽裕。速度较 Small 慢，比 Large 快，是一种中间状态。如果在这种编译模式下要使用多于 256B 的变量，那么变量的高 8 位地址（也就是具体哪一页）由 P2 口确定，须适当改变启动程序 STARTUP. A51 中的参数 PDATA START 和 PDATA LEN，用 L51 进行连接时不仅能采用连接控制命令 PDATA 来对 P2 口地址进行定位，也可用 pdata 指定。

3. Large 存储模式

该模式所有函数和过程的变量以及局部数据段都被定位在外部数据存储器中，外部数据存储器最多达 64KB，需要用 DPTR 数据指针来间接访问数据。这种访问方式效率不高，尤其是对于两个或多个字节的变量，用这种方式访问数据，程序的代码可能很多。

4. 注意设定数组的存储空间

设定一个数组时，C 编译器就会在存储空间开辟一个区域用于存放该数组的内容。字符数组的每个元素占用 1B 的内存空间，整型数组的每个元素占用 2B 的内存空间，而长整型（long）和浮点型（float）数组的每个元素则需要占用 4B 的存储空间。

嵌入式控制器的存储空间有限，要特别注意不要随意定义大容量的数组。在 Target 选项卡中将 Memory Model 设定为 Small 时编译不能通过。例如，若定义了一个浮点型的 10×10 的二维数组，则它需要占用 400B 的 RAM，采用 Small 模式时 8051 单片机的内部 RAM 只有 128B。

3.4.4 变量的分类

1. 全局变量和局部变量

（1）全局变量是在任何函数之外说明的、可被任意模块使用的、在整个程序执行期间都有效的变量。

（2）局部变量是在函数内部进行说明，只在本函数或功能块内有效，在该函数或功能块以外则不能使用。

局部变量可以与全局变量取相同的名字，此时，局部变量的优先级高于全局变量，即同名的全局变量在局部变量使用的函数内部将被暂时屏蔽。

2. 静态存储变量和动态存储变量

从变量的生存时间来区分，变量分为两种：静态存储变量和动态存储变量。

（1）静态存储变量是指该变量在程序运行期间其存储的空间固定不变。

（2）动态存储变量则指该变量的存储空间不是固定的，而是在程序运行期间根据需要动态地为其分配存储空间。

通常，全局变量为静态存储变量，局部变量为动态存储变量。当程序退出时，局部变量占用的空间释放。

使用 Keil C 编写程序时，无论是 char 型还是 int 型，要尽可能采用 unsigned 型的数据。因为在处理有符号数据时，程序要对符号进行判断和处理，运算速度会减慢；而且对单片机而言，速度比不上 PC，又工作于实时状态，所以任何提高效率的方法都要考虑。

3.5 C51 绝对地址的访问

在一些情况下，可能希望把一些变量定位在 51 单片机的某个固定的地址空间上。C51 为这些变量专门提供了一个关键字 at。at 的使用格式如下。

```
[memory_space] type variable_name _at_ constant;
```

格式中各参数的含义如下：

- memory_space——变量的存储空间。如果没有这一项，那么会使用默认的存储空间。
- type——变量类型。
- variable_name——变量名。
- _at_——关键字。
- constant——常量。该常量的值为变量定位的地址值。这个值必须在设置的物理地址范围之内，否则 C51 编译器会报错。

```
char xdata text[256] _at_ 0xE000;  //数组在 xdata 型存储区的 0xE000 地址处
```

例如，

```
void main(void){
i = 0x1234;
text[0] = 'a'; }
```

注意：绝对地址的变量是不可以被初始化的。函数或者类型为 bit 的变量是不可以被定义为绝对地址的。

关键字_at_的另一个功能是：能通过给 I/O 元器件指定变量名，从而为输入/输出元器件指定变量名。例如，在 xdata 段的地址 0x4500 处有一个输入寄存器，那么可以通过下面的代码段为它指定变量名。

```
unsigned char xdata inputreg _at_ 0x4500;
```

以后再读该寄存器时只要使用变量名 inputreg 即可。

3.6 指针

指针是 C 语言中的一个重要概念，也是 C 语言的一个重要角色。正确灵活地运用指针，可以有效地表示复杂数据结构，方便地使用字符串，有效地使用数组，调用函数时得到多个返回值，还能直接与内存打交道，这对于嵌入式编程尤其重要。掌握指针的应用，可以使程序简洁、紧凑、高效。

指针的概念比较抽象，使用也比较灵活。本节主要针对嵌入式 51 单片机编程介绍指针的一些基本用法，不涉及 PC 编程中用到的多层指针等更为抽象的概念。

3.6.1 指针的概念、定义和引用

1. 指针的概念

在使用汇编语言进行编程时，必须自行定义每一个变量的存放位置。例如：

```
Tmp EQU 5FH                    //将 5FH 这个地址分配给变量 Tmp
```

C 语言编程中，定义为“unsigned char Tmp；”，但不能看出 Tmp 存放的位置。Tmp 存放的位置是由 C 编译程序决定的。它不是一个定值，即便是同一个程序，一旦进行修改，增加或减少若干个变量，重新编译后 Tmp 的存放位置也会随之变化。

获得 Tmp 变量所在位置的方法：把变量的地址放到另一个变量中，然后通过对这个特殊的变量进行操作。

这个用来存放其他变量地址的变量称为“指针变量”。例如定义一个变量 p，并且 p 中存储的数据就是 Tmp 所在的地址值(0x5F)，则 p 就是一个指向 Tmp 的“指针变量”。

2. 指针变量的定义

1）定义

定义指针变量的一般形式如下。

```
基本类型 *指针变量名;
```

例如，

```
char *cp1, *cp2;               /*定义两个字符型的指针变量 cp1 和 cp2*/
int *P1, *P2;                  /*定义了两个整型指针变量 P1 和 P2*/
```

char 和 int 是在定义指针变量时指定的“基本类型”，P1、P2 可以指向整型数据，但不能指向 float 或者 char 等其他类型的数据。

2）注意事项

定义指针变量时需注意以下两点。

（1）指针变量前的“*”表示该变量为指针变量。

（2）定义指针变量时必须指定基本类型。不同类型的数据在内存中占用的字节数是不一样的。对于 C51 而言，char 或 unsigned char 型变量在内存中占用 1B；int 或 unsigned int 型变量在内存中占用 2B；long 或 unsigned long 和 float 型变量，在内存中占用 4B。

在指针的操作中，常用的一种操作是指针变量自增。如 p++，其含义是将指针指向这个数据的下一个数据，如果一个数据占用 1B，那么每次指针自增时，只要将地址值增加 1 即可；而如果一个数据占用 2B，每次指针自增加时，就必须将该值增加 2，这才能指向下一个变量。

3. 指针变量的引用

C 语言提供了两个运算符，用来获得变量地址，或使用指针所指变量的值。

（1）*&：取地址运算符。

（2）*：指针运算符(或称“间接访问”运算符)。

例如，&a 为变量 a 的地址，*Point 为指针变量 Point 所指向的变量。

3.6.2 C51的指针类型

C51支持“通用”和“存储器专用”两种指针类型。

1. 通用指针

1）通用指针结构

通用指针需占用3B，其中存储器类型占1B，偏移量占2B，如表3-3所示。存储器类型决定对象所用的C51存储空间，偏移量指向实际地址。通用指针可以被用来指示51单片机存储器中的任何类型的变量，所以在C51库函数中通常使用这类指针类型。

表3-3 通用指针的构成

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1个字节表示指针的存储器类型编码，如表3-4所示。

表3-4 一般指针存储器类型的编码

存储器类型	idata	xdata	pdata	data	code
值	1	2	3	4	5

例如，一个通用指针指向地址为0×1234的xdata类型数据时，其指针值如表3-5所示。

表3-5 指向xdata型数据的一般指针的值

地址	+0	+1	+2
内容	0x02	0x12	0x34

2）通用指针的定义

通用指针的定义与一般的C语言的指针定义相同，例如，

```
char * s;                    // 指向字符型的指针 s
int * numptr;                //指向 int 型的指针 numptr
long * state;                //指向 long 型的指针 state
```

例如，将一个数值0x12写入地址为0x8000的外部数据存储器，程序代码如下：

```
#define XBYTE ((char * )0x20000L)
XBYTE[0x8000] = 0x12;
```

其中，0x20000L是一个通用指针，将其分为3字节：0x02\0x00\0x00，查表可以看到0x02表示存储器类型xdata型，而地址则是0x0000。

XBYTE被定义为(char *)0x20000L，即XBYTE为指向xdata零地址的指针。XBYTE[0x8000]则是外部数据存储器的0x8000绝对地址。

3）应用

下面的代码显示了使用通用指针的变量在51单片机中是如何实现的，请注意指针各个字节的作用。

```
char * c_ptr;                    //char 型指针
int * i_ptr;                     //int 型指针
long * l_ptr;                    //long 型指针
main()
{
    char data dj;                //data 区变量
    int data dk;
    long data dl;

    char xdata xj;               //xdata 区变量
    int xdata xk;
    long xdata xl;

    char code cj = 9;            //code 区变量
    int code ck = 357;
    long code cl = 123456789;

    c_ptr = &dj;                 //data 区指针
    i_ptr = &dk;
    l_ptr = &dl;

    c_ptr = &xj;                 //xdata 区指针
    i_ptr = &xk;
    l_ptr = &xl;

    c_ptr = &cj;                 //code 区指针
    i_ptr = &ck;
    l_ptr = &cl;
```

在上面的代码中，通用指针 c_ptr、i_ptr 和 l_ptr 都被存放在单片机的内部数据存储区中。如果有需要，则可以使用关键字对指针的存储位置进行声明，其格式如下。

```
基本类型 * 存储类型 指针变量名;
```

例如，

```
char * xdata strptr;             //存储在 xdata 的字符串指针
int * data numptr;               //存储在 data 的 int 型指针
long * idata varptr;             //存储在 idata 的 long 型指针
```

2. 存储器专用指针

存储器专用指针的定义一般包含了数据类型和存储器类型的说明，其格式如下。

```
基本类型 存储器类型 * 指针变量名;
```

例如，

```
char data * px;                  //指向 data 的字符串型指针 px
int xdata * numtab;              //指向 xdata 的 int 型指针 numtab
long code * powtab;              //指向 code 的 long 型指针 powtab
```

存储器专用指针只需要 1B（当数据类型为 idata、data、pdata 时）或者 2B（当数据类型为

code、xdata 时)。因为专用指针比通用指针的字节少,所以在程序执行时会快一点。由于专用指针的一些特性在编译时由编译器来处理,所以优化选项有时会对编译结果产生一些影响。

与通用指针相同,也可以为专用指针指定存储空间,如:

```
char data * xdata str;              // str 在 xdata 中,指向 data 的 char 类型
int xdata * data pdx;               //pdx 在 data 中,指向 xdata 的 int 类型
long code * idata powtab;           //powtab 在 idata 中,指向 code 的 long 类型
```

3. Keil 预定义指针

Keil 软件预定义了一些指针,用来对存储器指定地址进行访问,其完整定义在 absacc.h 中,读者可自行查看。部分定义如下:

```
#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)
```

借助于这些指针可以对指定的地址进行直接访问。

3.7 C51 函数

一个较大的程序一般应由若干程序模块组成,每一个模块用来实现一个特定的功能。所有的高级语言都有子程序,正是通过这些子程序实现模块的功能。在 C 语言中,子程序的作用是由函数来完成的。

3.7.1 C51 函数及其定义

1. 函数及其分类

1) 函数

在程序设计中,通常将一些常用的功能模块编写成函数,并可放在函数库中以供选用,这样可以减少重复程序段的工作量。

一个完整的 C 程序可由一个主函数和若干个函数组成,由主函数调用其他函数,其他函数也可以相互调用。同一个函数可以被一个或多个函数多次调用。C 语言中的主函数为 main()。对于函数有如下说明:

(1) 一个源程序文件由一个或多个函数组成。

(2) 一个 C 程序由一个或多个源程序文件组成。对于较大的程序,通常不希望把所有源程序全部放在一个文件中,而是将函数和其他内容分别放入若干个文件中,再由这些文件组合成一个完整的 C 程序。这样可以分别编写、编译,而且一个源文件也可供多个程序使用,从而提高效率。

(3) C 程序的执行从 main()函数开始。

(4) 所有函数都是平行的,即在定义函数时是相互独立的,一个函数并不从属于另一个函数,即函数不能嵌套定义。函数间可以相互调用,但不能调用 main()函数。

2）函数的分类

（1）从形式上看，函数可以分为以下两种。

- 无参函数。即主函数不向被调用函数传递参数，这类函数只是完成一定的操作功能。无参函数可以有返回值，但大多数的无参函数通常也没有返回值。
- 有参函数。在调用函数时，主函数将一些数据传递给被调用函数，通常被调用函数会对这些数据进行处理，然后进行不同的操作，最后还可能有返回值。

（2）从用户使用的角度上看，函数可以分为以下两种。

- 标准函数，即库函数。这是由编译系统（如 Keil 软件）提供的，用户不必自己编写这些函数。如 sin 函数提供正弦函数计算功能。
- 用户函数。这是用户根据自己的需要而编写的特定功能的函数。

2. 函数的定义

1）定义

C51 函数的定义与 ANSI-C 中基本相同，唯一不同的是函数的后面可能带若干 C51 专用的关键字。

C51 函数的定义格式如下，其中方括号内是可选项。

```
[return _ type] funcname([args ])[ {small | compact | large } ] [reentrant] [interrupt n][using n]
{
  声明部分
  语句
}
```

各参数说明如下：

- return_type——返回值类型（数据类型标识符）。
- funcname——函数名。
- args——形式参数。
- {small | compact | large}——函数模式选择，在没有显示选择函数模式的情况下，使用默认的模式来编译。
- reentrant——再入函数。
- interrupt n——中断函数。
- using n——寄存器组选择，CPU 可以通过切换到一个不同的寄存器组来执行程序而不需要对若干寄存器进行保存。

声明部分：声明部分定义要使用的变量，此外还对将要调用的函数做声明。

例如，

```
int max(int x,int y)
{   int z:
    z = x > y?x: y;
    Return(z);
}
```

这是一个求 x 和 y 两者中哪个数值较大的函数。函数名 max 前面的 int 表示函数的返回值是一个整型数。括号中有两个形式参数 x 和 y，它们都是整型的。大括号里是函数体，

它包括声明部分和语句部分。在声明部分定义要使用的变量 z。return(z)的作用是将 z 的值作为返回值带回主调函数中。函数被定义为 int 型的，z 也是 int 型，两者是一致的。

再如：

```
long factorial(int n) reentrant
void time0_int(void) interrupt 1 using 1
```

2）空函数

C 语言允许有空函数，空函数的定义形式为：

```
类型标识符函数名()
{ }
```

调用空函数表示什么工作也不做。

例如，

```
void dummy()
{ }
```

在程序设计中往往根据需要确定若干个模块，分别由一些函数来实现，而在第一阶段只设计最基本的模块，即先把架子搭起来，细节留待进一步的完善。以这样的方式编写程序时，可以在将来准备扩充功能的地方定义一个空函数，表示这些函数未编写好，只是先占一个位置，以后用一个编写好的函数替代它。这样做可使程序的结构清楚，可读性好，以便以后扩充新功能，而对程序结构影响不大。

3.7.2　C51 的中断服务函数

中断是指当计算机执行正常程序时，由于系统中出现某些紧急处理的情况或特殊请求时，计算机打断当前正在运行的程序，转而对这些紧急情况进行处理。处理完毕后，再返回继续执行被打断的程序。

51 系列单片机的中断共分 2 个优先级，5 个中断源：外部中断请求 0，由 $\overline{\text{INT0}}$ 输入；外部中断请求 1，由 $\overline{\text{INT1}}$ 输入；定时器/计数器 0 溢出中断请求；定时/计数器 1 溢出中断请求；串口发送/接收中断请求。每个中断源的优先级都是可以编程的。

对于 52 系列单片机来说，除了以上 5 个中断外，还增加了一个定时/计数器 2 溢出中断请求。

1. 中断服务函数程序的定义

Keil C51 支持在 C 语言源程序中直接编写 51 单片机的中断服务程序，为此 Keil C51 对函数的定义进行了扩展，增加了一个扩展关键字 interrupt。其定义形式为：

```
类型标识符 函数名(形式参数)[interrupt m] [using n]
```

（1）函数名可以是任意合法的字母或数字组合。

（2）m：关键字 interrupt 后面的中断号取值范围是 0～4 或 0～5。Keil C51 编译器从 8m＋3 处产生中断向量，即当响应中断申请时，程序会根据中断号自动转入地址为 8m＋3 的位置，执行相对应的中断服务子程序。51 单片机的中断号、中断源和中断入口地址如表 3-6 所示。

表 3-6　51 单片机的中断号、中断源和中断入口地址

m	中　断　源	中断入口地址 8m+3
0	外部中断 0	0003H
1	定时/计数器 0 溢出	000BH
2	外部中断 1	0013H
3	定时/计数器 1 溢出	001BH
4	串口中断	0023H
5	定时/计数器 2 溢出	002BH

表 3-6 中第 5 号中断定时/计数器 2 溢出仅对 52 系统具有 3 个定时/计数器的单片机有效。

中断服务函数可以被放置在程序的任意位置。因为 Keil C51 在最后进行代码连接时会自动将服务函数定位到中断入口处，实现中断服务响应。

由于各个中断入口地址相距较近，Keil C 编译时会自动在对应的中断入口地址单元中安排一条转移类指令，以便转入到中断服务程序。此外，51 编译器在对中断函数进行编译时，也会根据需要自动将 PC 寄存器压入堆栈，在中断服务程序的最后自动安排一条 RETI 指令，以便将响应中断时所置位的优先级状态触发器清 0，然后从堆栈弹出程序计数器 PC，从原来打断处继续执行被中断的程序。

（3）n：51 系列单片机可以在内部 RAM 中使用 4 个不同的工作寄存器组，称为 0～3 组。每个寄存器组都包含 8 个工作寄存器（R0～R7）。可以通过关键字 using 来选择不同的工作寄存器组。using 后面的 n 取值为 0～3 的整数，分别代表 4 个不同的工作寄存器组。

注意：m 和 n 必须是整数，不能是表达式。

在单片机响应中断进入中断服务函数时，特殊功能寄存器 ACC、B、DPH、DPL、PSW 都将被压入堆栈。如果不使用寄存器组切换，则中断函数中所用到的全部工作寄存器也都会入栈。函数返回前，所有的寄存器内容再依次出栈。但如果在中断函数定义时用 using 指定了工作寄存器组，那么发生中断时，平时默认的工作寄存器组就不会被压栈，也就是说，系统直接切换寄存器组而不必进行大量的 PUSH 和 POP 操作，这将节省 32 个处理周期，因为每个寄存器入栈和出栈都需要两个处理周期。由此可以节省 RAM 空间，加速 MCU 执行时间。但这样也有缺点，就是所有调用中断的过程都必须使用指定的同一个寄存器组，否则参数传递会发生错误。因为对于 using 的使用应根据情况灵活取舍。

2. 规定

编制中断函数时应遵循以下规定：

（1）中断函数不能进行参数传递。

（2）中断函数没有返回值。

（3）中断服务函数不能被其他函数调用，只能由硬件产生中断后自动调用。

如果在中断函数中调用其他函数，则必须保证被调用函数所使用的寄存器组与中断函数一样，否则会产生不正确的结果。另外，由于中断的产生不可预测，中断函数对其他函数的调用有可能会形成递归调用，为了避免产生递归调用，尽量不要在中断服务函数内使用函数调用。如果确实需要调用其他函数，应保证此函数为中断服务独自专用，或者用扩展关键字 reentrant 将被中断函数调用的其他函数定义为再入函数。

（4）如果中断函数中用到浮点运算，必须保存浮点寄存器的状态，当没有其他程序执行浮点运算时可以不保存。在 Keil C 编译器的数学函数库 math.h 中，专门提供了保存浮点

寄存器状态的库函数 fpsave 和恢复浮点寄存器状态的库函数 fprestore，可供用到浮点运算的中断函数使用。

(5) 在中断函数程序执行过程中，对其他可能在此产生的中断并不响应，因而为了系统能够及时地响应各种中断，提高实时性能，中断函数的执行时间不宜过长，因此中断函数应尽量简洁。

3.7.3　C51 库函数

库函数并不是 C 语言的一部分，它是由编译软件开发公司根据需要编制并提供给用户使用的。本节只介绍了 C51 提供的库函数的一小部分，其余库函数请查阅相应的手册。

1. C51 库函数的测试方法

不同类型的函数运行时要采用不同的方法观察其测试结果。

(1) 如果在测试函数中用到了 print()函数，那么首先要用 #include < stdio.h >将头文件 stdio.h 包含到源程序中，其次要在 main()函数中设置串口，利用 Keil 软件的串行窗口进行输出，以便于观察。而要设置串口，又必须用 #include < reg51.h >或 #include < reg52.h > 将头文件 reg51.h 或 reg52.h 加入源程序中，否则无法通过编译。

(2) 使用 get()、getchar()之类的输入函数时，采用与上述相同的方法处理，可以在串行窗口中输入所需要的字符，这些字符可以被有关函数接受。

(3) 如果测试函数有 printf()之类的输出函数，那么可以直接观察输出以确定结果，也可以观察变量窗口以确定函数的工作是否正常。

(4) 部分函数测试时定义了大容量的数组，因此在设置工程时，必须将 Memory Model 由默认的 Small 模式改为 Large 模式，否则无法通过编译和链接。

2. 绝对地址访问 absacc.h

使用这一类函数时，应该把 absacc.h 头文件包含到源程序文件中。

1) CBYTE、DBYTE、PBYTE、XBYTE 函数

原型：

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

描述：上述宏定义用来对 8051 系列单片机的存储器空间进行绝对地址访问，可以作为字节寻址。CBYTE 寻址 CODE，DBYTE 寻址 DATA 区，PBYTE 寻址分页 PDATA 区，XBYTE 寻址 XDATA 区。

例如，若访问外部数据存储器区域的 0x1000 处的内容，则可以使用如下指令：

```
val = XBYTE[0x1000];
```

2) CWORD、DWORD、PWORD、XWORD 函数

原型：

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

描述：这个宏与前面的一些宏类似，只不过数据类型为 unsigned int 型。

3.8 C51 程序设计实例——实现单片机控制流水灯

1. 硬件电路设计

硬件电路如图 3-2 所示。P1 口的 8 个引脚经限流电阻分别接了 8 个发光二极管的阴极，这 8 个发光二极管的阳极共同接＋5V 电源。规定这 8 个发光二极管的点亮顺序是：先点亮 P1.0 引脚接的发光二极管，随后依次点亮 P1.1～P1.7 引脚所接的发光二极管，然后

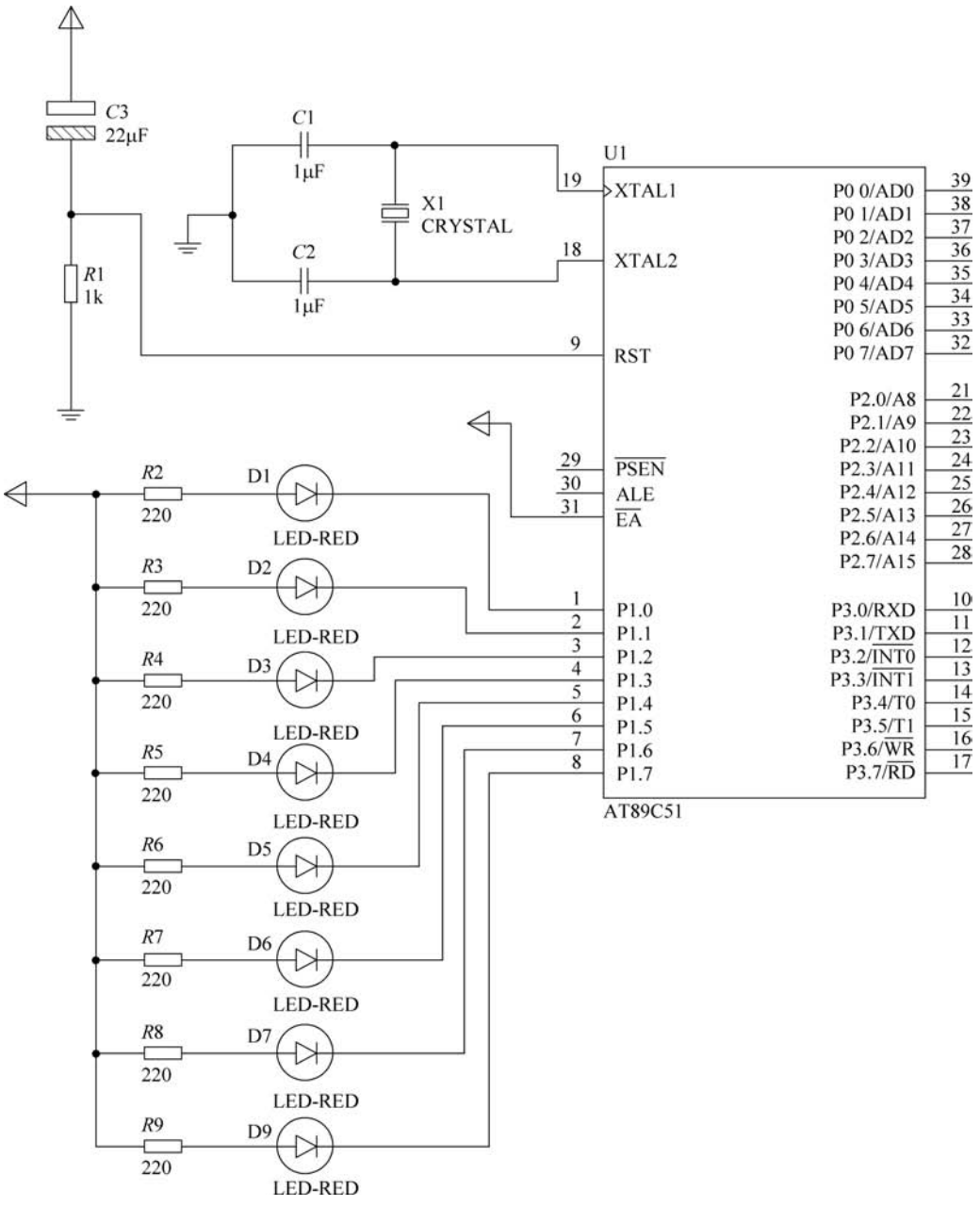

图 3-2　硬件电路

倒序，先从 P1.7 所接的发光二极管，依次过渡到 P1.0 所接的发光二极管进行点亮，然后依次循环。由于 8 个发光二极管共阳，所以点亮哪个发光二极管只需要所对应阴极向对应的 P1 口引脚输送低电平即可。

2. 程序设计

实现该功能的 C51 程序清单如下：

```
#include <reg51.H>
unsigned char i;
unsigned char temp;

void delay(void)
{
        unsigned char m,m,s;
        for(m = 20; m > 0; m -- )
           for(n = 20; n > 0; n -- )
                for(s = 248; s > 0; s -- );
}
void main(void)
     {
       while(1)
       {
          temp = 0xfe;
          P1 = temp;
          delay ();
        }
        For(i = 1; i < 8; i++)
        {
             a = temp >> i;
             P1 = a;
             delay();
```

以上 C51 源代码是不能够直接在单片机上执行的，单片机系统能够运行的为可执行程序，也就是经编译器译成的二进制文件。要实现从源代码编译成可执行文件，需要 C51 编译器及对应的集成开发环境。后面两章将介绍相应的编译器及集成开发环境的使用方法。

本章小结

C51 是面向 51 系列单片机所使用的程序设计语言，使 MCS-51 单片机的软件具有良好的可读性和可移植性。具有操作直接、简洁和程序紧凑的优点，为大多数 51 单片机实际应用最为广泛的语言。

C51 系列单片机在物理上有 3 个存储空间，即程序存储器、片内数据存储器、片外数据存储器。C51 在定义变量和常量时，需说明它们的存储类型，将它们定位在不同的存储区中。单片机常用的存储类型有 data、bdata、idata、pdata、xdata 和 coda 6 个具体类别。默认类型由编译模式指定。

C51 编译器已经把 MCS-51 系列单片机的特殊功能寄存器、特殊位和 4 个 I/O 口(P0～P3)进行了声明，放在 reg51.h 或 reg52.h 头文件中。用户在使用之前用一条预处理命令

“#include <reg51.h>”把这个头文件包含到程序中，就可以使用特殊功能寄存器名和特殊位名称了，而对于未定义的位，使用之前必须先定义。

C51 提供了一组宏定义，包括 CBYTE、DBYTE、XBYTE、PBYTE、CWORD、DWORD、XWORD 和 PWORD 来对单片机进行绝对寻址，同时也可以使用_at_关键字对指定的存储器空间的绝对地址进行访问。

C51 支持基于存储器的指针和一般指针两种指针类型。基于存储器的指针可以高效访问对象，且只需 1～2B。而一般指针需占用 3B，其中 1B 为存储器类型，2B 为偏移量，具有兼容性。

C51 语言中断函数的定义中使用了关键字 interrupt、using、中断号、寄存器组号等；并且 C51 也提供了一些常用的库函数，如 I/O 函数库、标准库函数、内部函数库、数学函数库、绝对地址访问函数库等。

思考题与习题

3-1　C51 语言的变量定义包含哪些关键因素？

3-2　C51 与汇编语言的特点各有哪些？怎样实现两者的优势互补？

3-3　定义变量为有符号字符型变量数据类型为_____，无符号整型变量数据类型为_____。

3-4　定时器 T0 中断号为_____。

3-5　关键字 bit 和 sbit 有何区别？