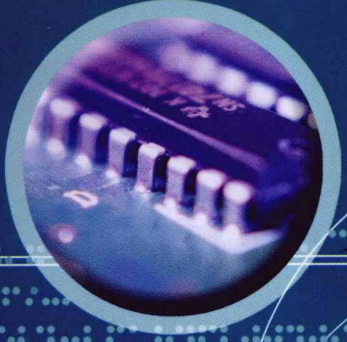
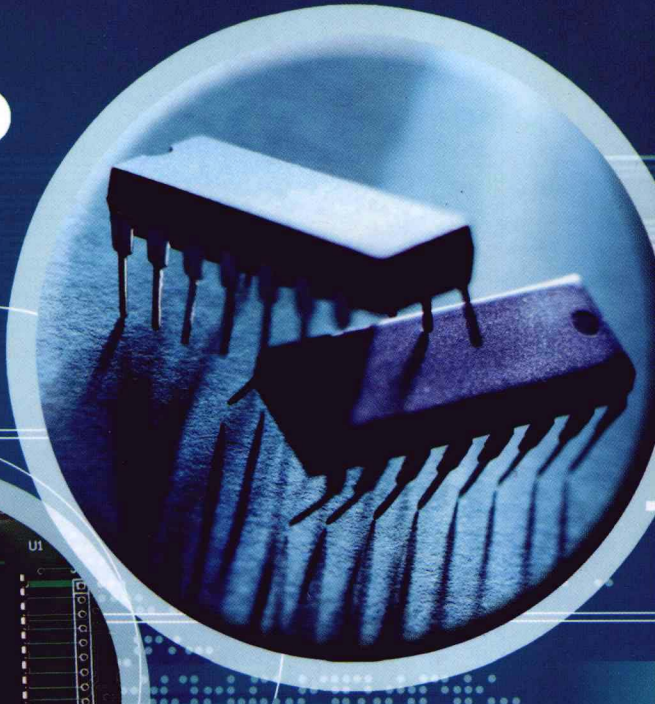
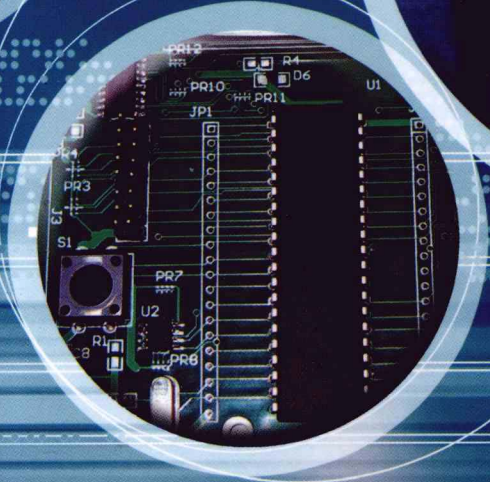
案例学单片机
C语言开发

吴戈　李玉峰　编著



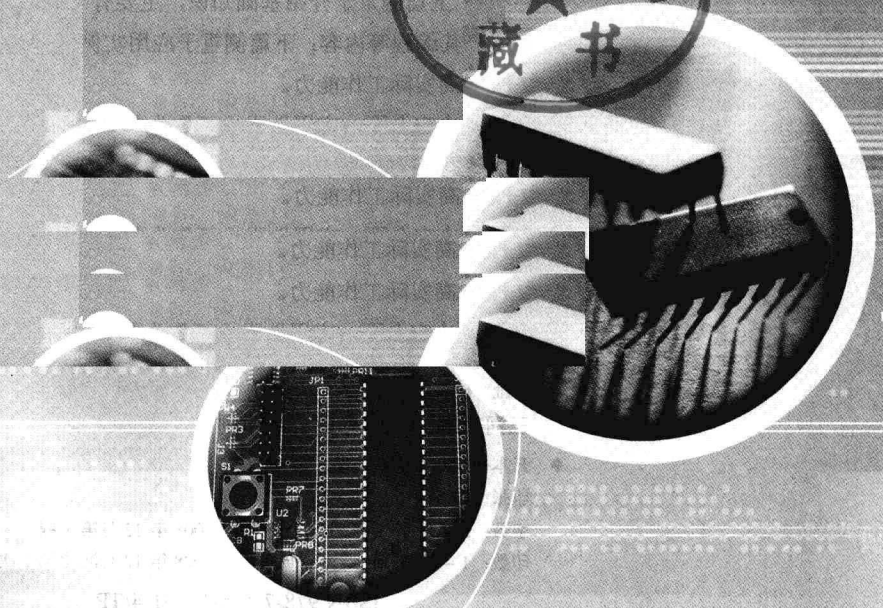




人民邮电出版社
POSTS & TELECOM PRESS

案例学单片机C语言开发

吴戈　李玉峰　编著



人民邮电出版社
北京

图书在版编目（CIP）数据

案例学单片机C语言开发 / 吴戈编著. —北京：人民邮电出版社，2008.12
ISBN 978-7-115-19031-4

Ⅰ. 案… Ⅱ. 吴… Ⅲ. 单片微型计算机－C语言－程序设计 Ⅳ. TP368.1 TP312

中国版本图书馆CIP数据核字（2008）第163222号

内 容 提 要

本书利用 Keil 公司最新版的μVision3 集成开发环境，从应用的角度，全面地介绍了用 C51 开发调试单片机程序的方法、过程和应注意的事项。书中列举了大量应用实例，使读者尽快、尽可能容易地掌握开发单片机的方法。

全书分为上、下两篇。上篇侧重于介绍基础知识，主要有 C51 语言和 C 语言以及汇编语言的对比，单片机内部资源、扩展资源及其编程等内容；下篇侧重于应用实例，通过实例，读者既可以在工作中进行类比编程，又可以开阔思路，提高实际工作能力。

本书特点是实例新颖、内容全面、实用性强，可作为单片机爱好者以及单片机开发人员的实用参考书。

案例学单片机 C 语言开发

◆ 编　　著　吴　戈　李玉峰
责任编辑　刘　浩

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市潮河印业有限公司印刷

◆ 开本：787×1092　1/16
印张：21.5
字数：527 千字　　2008 年 12 月第 1 版
印数：1－4 000 册　　2008 年 12 月河北第 1 次印刷

ISBN 978-7-115-19031-4/TP

定价：36.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前　言

8051 是 Intel 公司开发的一款相当成功的单片机，现在已普遍应用于工业生产中。目前有很多半导体芯片公司制造出了与 8051 兼容的单片机，它们构成了通常所说的 51 系列单片机。

C 语言是具有结构化、模块化编译的通用计算机语言，是国际上应用最广、最多的计算语言之一。C51 是在通用 C 语言的基础上开发出的专门用于 51 系列单片机编程的 C 语言。与汇编语言相比，C51 在功能上、结构上以及可读性、可移植性、可维护性等方面都有非常明显的优势。目前最先进、功能最强大、国内用户最多的 C51 编译器是 Keil Software 公司推出的 Keil C51。

本书没有把太多的篇幅放在介绍 51 系列单片机的结构原理以及汇编指令上，因为介绍这方面知识的书籍和资料很多，而是把主要篇幅放在基于 C51 语言的单片机应用上。本书分上篇和下篇，上篇侧重于基础知识，下篇侧重于实际应用。

上篇包括第 1～5 章。

第 1 章和第 2 章介绍 C51 语言程序设计方面的知识。

第 3 章以举例的形式介绍μVision3 集成开发环境（IDE）。

第 4 章介绍单片机内部资源及其 C 语言编程。

第 5 章介绍单片机资源扩展及其 C 语言编程，介绍的扩展芯片是目前应用比较广泛的。

下篇包括第 6～14 章。

第 6 章介绍液晶显示 LCD 应用，分别介绍了两种液晶显示控制芯片。

第 7 章介绍虚拟 I^2C 接口技术。

第 8 章介绍红外通信接口应用。

第 9 章介绍语音芯片 ISD4004 及其应用。

第 10 章介绍时钟芯片应用，也分别介绍了两种时钟控制芯片。

第 11 章介绍数据采集，详细介绍数据采集的关键芯片 AD 转换器以及 DA 转换器的应用。

第 12～13 章分别介绍单片机之间的通信、单片机与 PC 通信。

第 14 章详尽地介绍了单片机在模糊控制、医疗以及语音数据采集方面的综合应用。

读者在学习 C51 程序的过程中，应弄懂 C51 程序结构特性，领会其设计思想，而不应仅仅是盲目地照搬照抄，这样才能在原来的基础上更好地开发，取得更好的效果，设计出更好的程序和产品。

由于以 51 为内核的单片机应用很广泛，生产的厂家很多，品种型号也很多，性能和价格也有很大的差别；而且 8051 单片机家族还一直在扩展壮大，新器件、新功能不断涌现，因此，读者在设计单片机应用系统时，要结合产品器件手册，注意区分不同厂家、不同型号之间的差别，选用适合自己系统的品种型号，采用最低廉的价格、最简洁的电路，达到最优良的性能。

各章的程序代码可以在 http://www.ptpress.com.cn 中的“资源下载”处下载。

本书内容丰富、通俗、实用，所举实例的实用性都很强，稍作改动即可应用到实际当中。

本书由吴戈、李玉峰编写，参加资料整理的还有何伟、张兵、刘兆宏、季建华、刘福刚、赵文革、黄弦、邓玉春、曾庆华、石昀、朱元斌、钱文杰、陈功杰、汪洪、刘超、钟晓媛等，在此深表感谢！

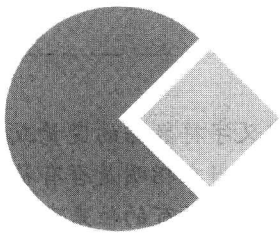
由于水平有限，书中难免有错误和不妥之处，恳请读者批评指正（可发信至book_better@sina.com）。

编者

2008 年 12 月

目　录

上　篇



第 1 章　Keil C51 语法

本章内容

C 语言在功能上、结构性、可读性和可维护性上比汇编有明显的优势，因而易学易用。Keil C51 软件提供丰富的库函数和功能强大的集成开发调试工具，全 Windows 界面。另外重要的一点，就是 Keil C51 生成的目标代码效率非常高，多数语句生成的汇编代码很紧凑，容易理解。在开发大型软件时更能体现高级语言的优势。

C51 继承了标准 C 语言的绝大部分的特性，而且基本语法相同，但其本身又在特定的硬件结构上有所扩展，本章对 C51 语法进行全面介绍。

1.1　数据类型

C51 能处理的数据类型如表 1-1 所示。

表 1-1　　Keil C51 的数据类型

类　型	长度（位数）	数 学 表 达
signed char	8	有符号字符变量，取值范围：-128～127
unsigned char	8	无符号字符变量，取值范围：0～255
signed int	16	有符号整型数，取值范围：-32 768～32 767
unsigned int	16	无符号整型数，取值范围：0～65 535
signed long	32	有符号长整型数，取值范围：-2 147 483 648～+2 147 483 647
unsigned long	32	无符号长整型数，取值范围：0～4 294 967 295
float	32	浮点数，取值范围：±1.175 494E-38～±3.402 823E+38
*	8~24	对象的地址
bit	1	布尔型位变量，0 或 1 两种取值

续表

类　型	长度（位数）	数 学 表 达
sfr	8	取值范围：0～255
sfr16	16	取值范围：0～65 535
sbit	1	取值范围：0 或 1

下面解释 C51 的基本数据类型。

➢ char（字符型）。char 类型的长度是一个字节，通常用于定义字符数据的变量或常量。char 型又分无符号字符型 unsigned char 和有符号字符型 signed char，当未指明是否有符号时，Keil C51 默认 char 字符型值为 signed char 型。unsigned char 型用字节中所有的位来表示数值，所以可以表达的数值范围是 0～255。unsigned char 常用于处理 ASCII 字符或用于处理小于或等于 255 的整型数。signed char 型用字节中最高位字节表示数据的符号，“0”表示正数，“1”表示负数，负数用补码（即该数的绝对值按位取反后再加 1）表示。因此，signed char 型数据所能表示的数值范围是-128～+127。

➢ int（整型）。int 整型数据的长度为两个字节，用于存放一个双字节数据。同样它分有符号 int 整型数 signed int 和无符号整型数 unsigned int，默认值为 signed int 类型。signed int 表示的数值范围是-32768～+32767，字节中最高位表示数据的符号，“0”表示正数，“1”表示负数，负数也是用补码表示。unsigned int 表示的数值范围是 0～65535。

➢ long（长整型）。long 长整型数据的长度为 4 个字节，用于存放一个四字节数据。与 char 和 int 数据一样，long 数据也分有符号长整型 signed long 和无符号长整型 unsigned long，默认值为 signed long 型。signed long 表示的数值范围是-2 147 483 648～+2 147 483 647，字节中最高位表示数据的符号，“0”表示正数，“1”表示负数，负数也是用补码表示。unsigned long 表示的数值范围是 0～4 294 967 295。

➢ float 浮点型。float 浮点型在十进制中具有 7 位有效数字，是符合 IEEE－754 标准的单精度浮点型数据，占用 4 个字节。

➢ *指针型。指针型本身就是一个变量，在这个变量中存放的指向另一个数据的地址。这个指针变量要占据一定的内存单元，对不同的处理器长度也不尽相同，在 C51 中它的长度一般为 1～3 个字节。

➢ bit 位标量。bit 位标量是 C51 编译器的一种扩充数据类型，利用它可定义一个位标量，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位，不是 0 就是 1，类似一些高级语言中的 Boolean 类型中的 True 和 False。

➢ sfr 特殊功能寄存器。sfr 也是一种扩充数据类型，占用一个内存单元，值域为 0～255。利用它可以访问 MCS-51 单片机内部的所有特殊功能寄存器。如用 sfr P1 = 0x90 这一句定义 P1 为 P1 端口在片内的寄存器，在后面的语句中我们用以用 P1 = 255（对 P1 端口的所有引脚置高电平）之类的语句来操作特殊功能寄存器。

➢ sfr16 16 位特殊功能寄存器。sfr16 占用两个内存单元，值域为 0～65535。sfr16 和 sfr 一样用于操作特殊功能寄存器，所不同的是它用于操作占两个字节的寄存器。

➢ sbit 可寻址位。sbit 可寻址位是 C51 中的一种扩充数据类型，利用它可以访问芯片内部 RAM 中的可寻址位或特殊功能寄存器中的可寻址位。例如我们定义 zwe 为位于位寻址区

bdata 的无符号字符型变量，如下所示：

```
unsigned char bdata zwe
```

再用 sbit 定义 zwe 中的每一位，如下所示：

```
sbit zwe_bit0=zwe^0
sbit zwe_bit1=zwe^1
sbit zwe_bit2=zwe^2
sbit zwe_bit3=zwe^3
sbit zwe_bit4=zwe^4
sbit zwe_bit5=zwe^5
sbit zwe_bit6=zwe^6
sbit zwe_bit7=zwe^7
```

于是我们在程序中就可以分别用 zwe_bit0、zwe_bit1……zwe_bit7 来代表 zwe 变量的第 0 位、第 1 位直至第 7 位。

在 C 语言中，用 signed 表示一个变量（或常数）是有符号类型，unsigned 表示无符号类型。它们表示数值范围不一样。必须注意的是有符号运算比无符号运算耗资源，因此应尽可能使用无符号数。

另外需要注意的是，C51 支持的多字节数据都是按照高字节在前，低字节在后的原则安排的，即一个多字节数，比如 int 型，在内存单元中存储顺序为高位字节存储在地址低的存储单元中，低位字节存储在地址高的存储单元中。

1.2 存储种类及存储区

C51 语言里把在程序执行过程中数值不改变的量叫常量。常量的数据类型有整型、浮点型、字符型等。

1.2.1 整型常量

整型常量就是整型常数，可按一般数字的写法直接写成十进制，如 12345、-12345、0，也可以写成十六进制数，写成十六进制数时必须以 0x 开头，如 0x64、0x123、0xFF。另外还可以在整型常数后面加一个字母“L”，构成长整型数，如 10L、123L 和 0x4FL。

浮点型常数可以写成十进制定点表示形式，即由数字和小数点组成，如 3.14159、-4.7、130.4 等，也可以写成指数形式，即：

```
[±] 数字 [.数字] e [±] 数字
```

其中，[ ]中的内容为可选项，可根据具体情况可有可无，但其余部分必须有。例如-456e-3、123e5、-123e4，等等。

1.2.2 字符型常量

字符型常量是单引号内的字符，如'a'、'b'等，一个字符占用一个字节。对于不可以显示的控制字符，可以在该字符前面加一个反斜杠“\”组成专用转义字符。利用转义字符可以完成一些特殊功能和输出时的格式控制。常用转义字符表如表 1-2 所示。

表 1-2 常用转义字符表

转义字符	含义	ASCII 码（16/10 进制）
\o	空字符（NULL）	00H/0
\n	换行符（LF）	0AH/10
\r	回车符（CR）	0DH/13
\t	水平制表符（HT）	09H/9
\b	退格符（BS）	08H/8
\f	换页符（FF）	0CH/12
\'	单引号	27H/39
\"	双引号	22H/34
\\	反斜杠	5CH/92

1.2.3 字符串常量

字符串型常量由双引号内的字符组成，如"ABCDE"、"OK"等。当引号内没有字符时，称为空字符串。请注意字符串常量前面和后面的双引号是界限符，当需要表示双引号字符串时，可以使用转义符号（\）表示，如"\""。

在 C51 语言中，字符串常量是作为字符类型数组来处理的，所以在存储字符串时，系统会在字符串尾部加上\o 转义字符以作为该字符串的结束符，正因如此，字符串常量"A"和字符常量'A'是不同的，前者在存储时多占用一个字节的空间。

1.2.4 位标量

位标量是 C51 编译器扩充的一种数据类型，它用关键字 bit 来定义，其值是一个二进制位。

在函数中可以包含 bit 类型的参数，函数的返回值也可以为 bit 型。例如：

```
#static bit a1                    //定义一个静态位标量 a1
extern bit b1                     //定义一个外部位 b1
bit func (bit c1,c2)     //定义一个返回位型值的函数 func，包含 2 个位型参数 c1 和 c2
{
……
return(c2) ;                      //返回一个位型值 c2
}
```

常量可用在不需要改变其值的场合，如固定的数据表、字库等。常量的定义方式有几种，

里，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位，不是 0 就是 1，类似

下面来加以说明。

```
#difine False 0x0;          //用预定义语句可以定义常量
#difine True 0x1;           //这里定义 False 为 0,True 为 1
                                //在程序中用到 False 编译时自动用 0 替换，同理 True 替换为 1
unsigned int code a=200;    //这一句用 code 把 a 定义在程序存储器中并赋值
const unsigned int c=110;   //用 const 定义 c 为无符号 int 常量并赋值
```

以上两句它们的值都保存在程序存储器中，而程序存储器在运行中是不允许被修改的，所以如果在这两句后面用了类似 a=100，a++这样的赋值语句，编译时将会出错。

1.3 变量

1.3.1 变量的定义

在程序执行过程中其值会发生变化的量叫变量。每个变量都必须有一个标识符作为它的变量名。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。

C51 对变量进行定义的格式如下：

```
[ 存储种类 ] 数据类型 [ 存储器类型 ] 变量名表
```

这里的“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。如果在定义变量时省略了存储种类这个选项，那么该变量将默认为自动（auto）变量。

1.3.2 存储器类型

存储器类型就是指该变量在 C51 硬件系统中所使用的存储区域，并在编译时能够准确地定位。表 1-3 所示是 Keil C51 所能识别的存储器类型。必须注意的是在 Intel 8051 芯片中，RAM 只有低 128 位，位于 80H 到 FFH 的高 128 位则在 52 芯片中或其他公司在扩展了 Intel 8051 后才有，并和特殊寄存器地址重叠。

表 1-3 Keil C51 编译器所能识别的存储器类型

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

还有要特别指出的是，变量的存储种类与存储器类型是完全无关的。例如：

```
static unsigned char data i;  //在内部数据存储器中定义一个静态无符号字符型变量
int j;                        //定义一个自动整型变量j，它的存储器类型由编译模式确定
```

第一条语句所定义的字符型变量 i，其存储种类为静态，即俗称的静态变量，它的存储器类型为data，也就是说，这个变量位于8051芯片内部RAM中的低128位，即地址为00H~7FH之间。

第二条语句所定义的是一个整型变量，由于没有定义存储种类和存储器类型，于是系统默认存储种类为自动，存储器类型由存储器编译模式确定。

1.3.3 存储器模式

定义变量时如果省略存储器类型，Keil C51 编译系统则会按编译模式 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。无论什么存储模式都可以声明变量在任何的 8051 存储区范围，但是把最常用的命令如循环计数器和队列索引放在内部数据区可以显著地提高系统性能。

- SMALL 存储模式把所有函数变量和局部数据段放在 8051 系统的内部数据存储区，因此对这种变量的访问数据最快。但 SMALL 存储模式的地址空间受限。在写小型的应用程序时，变量和数据放在 data 内部数据存储器中是很好的，因为访问速度快，但在较大的应用程序中 data 区最好只存放小的变量、数据或常用的变量（如循环计数、数据索引），而大的数据则放置在别的存储区域，否则 data 区就很容易溢出。
- COMPACT 存储模式中把变量定位在 MCS-51 系统的外部数据存储器中。外部数据存储段可有最多 256 字节（一页），这时对变量的访问是通过寄存器间接寻址（MOVX @Ri）进行的。采用这种编译模式时，变量的高 8 位地址由 P2 口确定，因此，在采用这种模式的同时，必须适当改变启动程序 STARTUP.A51 中的参数 PDATASTART 和 PDATALEN，用 L51 进行连接时还必须采用连接控制命令 PDATA 来对 P2 口地址进行定位，这样才能确保 P2 口为所需要的高 8 位地址。
- LARGE 存储模式中，所有函数和过程的变量以及局部数据段都被定位在 MCS-51 系统的外部数据存储器中，外部数据存储器最多可有 64KB，这要求用 DPTR 数据指针来间接地访问数据，因此，这种访问方式效率并不高，尤其是对于 2 个或多个字节的变量，用这种方式访问数据程序的代码将可能会很大。

变量还分为全局变量和局部变量。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保持有效的变量；局部变量在函数内部说明，只在本函数或功能块以内有效，在该函数或功能块以外则不能使用它。局部变量可以与全局变量取同样的名字，此时，局部变量的优先级将高于全局变量，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。

从变量的生存时间来区分，变量又可以分为两种：静态存储变量和动态存储变量。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，动态存储变量则指该变量的存储空间并不是固定的，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，

局部变量也就失去意义。静态变量占用固定的存储单元，因此这个存储单元不会被别的函数使用，其他函数可以通过指针访问或修改静态变量的值。静态变量在程序开始时只初始化一次，因此若需要在某函数内部使用一变量，而又希望其值在其他函数调用期间保持不变，为实现程序模块化，则可将其声明为静态变量。

1.3.4 特殊功能寄存器（SFR）

51单片机提供128字节的SFR寻址区，地址为80H~FFH。51单片机中，除了程序计数器PC和4组通用寄存器组之外，其他所有的寄存器均为SFR，并位于片内特殊寄存器区。这个区域可位寻址、字节寻址或字寻址，用以控制定时器、计数器、串口、I/O及其他部件。特殊功能寄存器可由以下几种关键字说明。

① sfr 声明字节寻址的特殊功能寄存器，比如 sfr P0=0x80；表示P0口地址为80H。注意："sfr"后面必须跟一个特殊寄存器名；"="后面的地址必须是常数，不允许带有运算符的表达式，这个常数值的范围必须在特殊功能寄存器地址范围内，位于0x80H到0xFFH之间。

② sfr16 许多新的8051派生系列单片机用两个连续地址的SFR来指定16位值，例如8052用地址0xCC和0xCD表示定时器/计数器2的低和高字节，如sfr16 T2=0xCC；表示T2口地址的低地址T2L=0xCC，高地址T2H=0xCD。sfr16声明和sfr声明遵循相同的原则，任何符号名都可用在sfr16的声明中。声明中名字后面不是赋值语句，而是一个SFR地址，其高字节必须位于低字节之后，这种声明适用于所有新的SFR，但不能用于定时/计数器0和计数器1。

③ sbit 声明可位寻址的特殊功能寄存器和别的可位寻址的目标。"="号后将绝对地址赋给变量名，3种变量声明形式如下。

- sfr_name^int_constant

该变量用一个已声明的**sfr** sfr_name作为sbit的基地址（SFR的地址必须能被8整除）。"^"后面的表达式指定了位的位置，必须是0～7之间的一个数字，例如：

```
sfr PSW = 0xD0;                    //声明 PSW 为特殊功能寄存器，地址为 0xD0
sfr IE = 0xA8;
sbit OV = PSW^2;
sbit CY = PSW^7;
sbit EA = IE^7;                    //指定 IE 的第 7 位为 EA，即中断允许
```

- int_constant^int_constant

该变量用一个整常数作为sbit的基地址，基地址值必须能被8整除。"^"后面的表达式指定位的位置，必须在0～7之间。例如：

```
sbit OV = 0xD0^2;
sbit CY = 0xD0^7;
sbit EA = 0xA8^7;                  //指定 0xA8 的第 7 位为 EA，即中断允许
```

- int_constant

该变量是一个sbit的绝对位地址，例如：

```
sbit OV = 0xD2;
sbit CY = 0xD7;
sbit EA = 0xAF;
```

特殊功能位代表一个独立的声明类，它不能和别的位声明或位域互换。sbit数据类型可以用来访问用bdata存储类型标识符声明的变量的位。

不是所有的SFR都是可位寻址的，只有地址可被8整除的SFR可位寻址。SFR地址的低半字节必须是0或8，例如，SFR在0xA8和0xD0是可位寻址的，0xC7和0xEB的SFR是不能位寻址的。计算一个SFR的位地址需要在SFR字节地址上加上位所在地址，因此若访问SFR0xC8的第6位，则SFR的位地址是0xCE，即0xC8+6。

1.3.5 重新定义数据类型

前面说过，Keil C51 支持的基本数据类型有 char、int 和 float 等，除了这些数据类型外，用户还可以根据自己的需要，用关键字 typedef 对数据类型重新定义，重新定义方法如下：

```
typedef  已有的数据类型  新的数据类型名;
```

在这里“已有的数据类型”指的是前面介绍的 C 语言中所有的数据类型，包括结构、指针和数组等。“新的数据类型名”可以按用户的习惯或根据任务的需要来决定。关键字 typedef 的作用只是将 C 语言中已有的数据类型作了置换，因此可用置换后的新数据类型名来进行变量的定义。举例如下：

```
typedef int counter;          /*将 counter 定义为整型*/
counter i,j;                  /*将 i,j 定义为整型*/
```

在这个例子里，第一条语句是用 typedef 将 counter 定义为新的整数型数据类型，实际上表示的是用 counter 置换了 int，这样，下面就可以直接用 counter 对变量 *i*，*j* 进行定义，此时 counter 相当于 int，所以 *i*，*j* 被定义成整型变量。

再举几个例子如下：

```
typedef int x [ 10 ]          //定义 x 为整型数组类型
x   n;                                  //将 n 定义为整型数组变量
```

```
typedef char * POINTER;       //将 POINTER 定义为字符指针类型
POINTER point;                          //将 point 定义为字符指针变量
```

用 typedef 还可以定义结构类型，见下面的例子：

```
typedef struct;               //定义结构体
{
    int month;
    int day;
    int years;
} DATE;
```

这里 DATE 为一个新的数据类型（结构类型）名，可以直接用它来定义变量：

```
DATE birthday;                          //定义 birthday 为结构类型变量
DATE * point;                       //定义指向这个结构类型数据的指针
```

注意用 typedef 可以定义各种新的数据类型名，但不能直接用来定义变量。typedef 只是对已有的数据类型作了一个名字上的置换，并不是创造出一个新的数据类型。例如第一个例子中的 counter，它只是 int 类型的一个新名字而已，并不是一种新的数据类型。

采用 typedef 来重新定义数据类型可以方便程序的移植，同时可以简化较长的数据类型定义（如结构数据类型等）。在采用多模块程序设计时，如果不同的模块程序源文件中用到同一类型的数据时（尤其是像数组、指针、结构、联合等复杂数据类型），经常用 typedef 将这些数据重新定义并放到一个单独的文件中，需要时再用预处理命令#include 将它们包含进来，可以方便程序的编写设计。

1.4 数组

数组是一组有序数据的组合。数组所表示的对象可以是字符型、整数型、浮点数型等，甚至指针型，以及结构联合等类型。数组中的每一个数据都属于同一种数据类型。

n 维数组的定义形式如下：

```
数据类型  数组名 [常量表达式 1] [常量表达式 2].....[常量表达式 n];
```

例如定义一个具有 10 个元素的一维整型数组 x 和一个具有 2×6 个元素的字符型二维数组 y，代码如下：

```
int x [10];
char y [6][6];
```

数组元素从 0 开始，而不是从 1 开始。例如 int x [10]是定义了 x [0]～x [9] 10 个元素，不是 x [1]～x [10]，这一点要注意。

如果希望在定义数组时同时给数组中各元素赋予初值，可以采用如下方法：

```
数据类型 [存储器类型]  数组名 [常量表达式]={常量表达式};
```

必须注意，初值的个数必须小于或等于数组中元素的个数。如果初值的个数小于数组中元素的个数，则不足者后面的元素都用 0 来添补。另外，在赋初值时，也可以不指定数组的长度，编译器会根据初值的个数自动计算出该数组的长度。下面是几个数组赋值的例子：

```
unsigned char x [ ] = {1,2,3,4,5};
unsigned char y [5] = {1,2,3,4,5};
int   ABC [4] [3] = {{0，0，0}，{1，2，3}，{3，5，7}，{4，8，3}};
```

数组名也可以用作函数的参数。用数组名作为函数的参数时，在进行函数调用的过程中，参数传递方式采用的是地址传递，而一个数组名表示的就是该数组的首地址，所以可以用数组名来传递。C 语言规定在引用数值数组时，只能逐个引用数组中的各个元素，而不能一次引用整个数组，但如果是字符数组则可以一次引用整个数组。

用数组作为函数的参数，必须在主调函数和被调函数中分别进行数组定义，而且在两个