单片机C语言程序设计实训100例

——基于8051+Proteus仿真

彭　伟　编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

单片机C语言程序设计实训100例

——基于8051+Proteus仿真

彭　伟　编著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书基于 Keil μ Vision 程序设计平台和 Proteus 硬件仿真平台，精心编写了 100 余个 8051 单片机 C 语言程序设计案例。

全书基础设计类案例贯穿 8051 单片机最基本的端口编程、定时/计数器应用、中断和串口通信；硬件应用类案例涵盖常用外部存储器扩展、接口扩展、译码、编码、驱动、光机、机电、A/D 与 D/A 转换等内容；综合设计类案例涉及大量消费类电子产品、仪器仪表及智能控制设备的相关技术。

本书可作为大专院校学生学习实践单片机 C 语言程序设计的教材或参考书，也可作为电子工程技术人员或单片机技术爱好者的参考资料。

图书在版编目（CIP）数据

单片机 C 语言程序设计实训 100 例：基于 8051+Proteus 仿真 / 彭伟编著. —北京：电子工业出版社，2009.6
ISBN 978-7-121-08757-8

I. 单… II. 彭… 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（2009）第 067795 号

责任编辑：万子芬（wzf@phei.com.cn）
印　　刷：北京市海淀区四季青印刷厂
装　　订：涿州市桃园装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×1092　1/16　印张：23.25　字数：595 千字
印　　次：2009 年 6 月第 1 次印刷
定　　价：45.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。
服务热线：（010）88258888。

前　言

目前，各高校电类专业都将C语言作为专业基础课程纳入教学计划。由于C语言功能强大，便于模块化开发，所带库函数非常丰富，编写的程序易于移植，诸多优点使之成为单片机应用系统开发最快速高效的程序设计语言，仅具有C语言基础知识但不熟悉8051单片机指令系统的读者也能很快掌握单片机的C语言程序设计技术，C语言在单片机应用系统设计上的效率已经远远高于汇编、PL/M、BASIC等开发语言。

单片机C语言程序设计不同于通用计算机应用程序设计，它必须针对具体的微控制器及外围电路来完成，为了便于学习单片机应用程序设计和系统开发，很多公司推出了单片机实验箱、仿真器和开发板等，这些硬件设备可用于验证单片机程序，开发和调试单片机应用系统，但由于这些设备价格不菲，普通读者不是人人都可以配备的。幸运的是，英国Labcenter公司推出了具有单片机系统仿真功能的Proteus软件，使读者仅用一台PC在纯软件环境中完成系统设计与调试成为可能。目前Proteus支持8051、AVR、PIC等多种单片机，系统库中包含有大量的模拟、数字、光电和机电类元器件，系统还提供了多种虚拟仪器，用Keil C开发的程序可以在用Proteus设计的仿真电路中调试和交互运行，这无疑为读者学习和提高单片机C语言程序设计技术，为单片机应用系统高水平工程师的成长提供了理想平台。

为帮助读者快速提高单片机C语言程序设计水平，本书基于德国Keil公司的μVision集成开发环境和英国Labcenter公司的Proteus仿真环境，精心编写了100余个8051单片机C语言程序设计案例，各案例同时给出了难易适中的实训目标。

本书前2章分别对C51程序设计和Proteus操作基础进行概述；第3章基础程序部分给出的案例贯穿8051单片机端口编程、定时/计数器应用、中断程序设计和串口通信程序设计，各案例分别对相关知识和技术要点进行阐述与分析，源程序中还给出了丰富的注释信息；第4章硬件应用部分则针对8051单片机常用的外部存储器外展、接口扩展、译码、编码、驱动、光机、机电、传感器、I^2C等器件给出了数十个案例，对案例中涉及的硬件技术资料亦进行了有针对性的分析，使读者可以快速理解相关代码的编写原理；第5章综合单片机内部资源和外部扩展硬件，给出了数十个综合设计案例，通过对这些案例的分析、调试运行及独立实训，读者用C语言设计开发8051单片机应用系统的能力会得到大幅提升。

本书由彭伟编写，在编写的中后期，笔者正在德国考察学习，为使本书早日与读者见面，笔者仍坚持挤出时间，每晚在住所笔耕不已。在本书的编写过程中，王魏、张力和魏来参与了第3章、第4章部分案例的设计调试，刘凯为本书提供了重要技术资料，在此对他们表示感谢！在本书选题、撰稿到出版的全过程中，学院领导、学院科研处及教师教育研究所始终给予了大力支持，并提供了项目资助，教务部和信息技术系也一直关注本书的编写与进展情况，在此一并对学院和部门领导的关心与支持表示由衷的感谢！

由于编者水平有限，加之时间仓促，书中错漏之处在所难免，在此真诚欢迎读者多提宝贵意见，作者邮箱是pw95aaa@foxmail.com。

另外，本书的AVR与PIC版也正在编写之中，笔者将努力争取使后续版本早日出炉，以飨读者。

彭伟

2009年5月于武昌

注：本书的案例压缩包在电子工业出版社网站（http://www.phei.com.cn）的“资源下载”栏目中提供，欢迎读者下载。

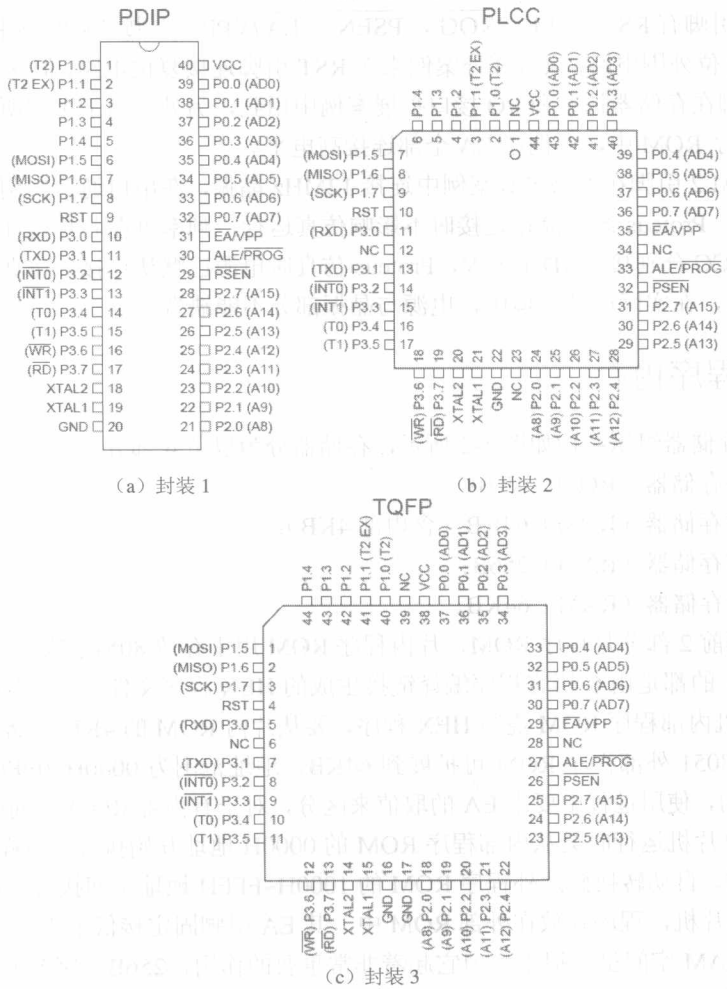
目　录

第 1 章　8051 单片机 C 语言程序设计概述

开发 8051 单片机系统时，使用 C 语言会使开发周期大为缩短，开发效率大幅提高，程序可读性好且易于移植，所以使用 C 语言开发单片机系统已经成为必然趋势。本书给出了 100 余个在 Keil μVision3 下编写的单片机 C 语言程序案例，这些案例同时给出了 Proteus 下的仿真电路。读者阅读本书需要已经学习了 8051 单片机 C 语言程序设计技术，本章仅介绍使用 C 语言设计单片机系统必须参考和重点掌握的内容，这些内容会给读者阅读、调试、研究本书案例及进行设计实训提供帮助。

1.1　8051 单片机引脚

图 1-1 给出了 8051 单片机的几种不同封装形式及引脚分布，本节将对单片机 4 个双向 I/O 端口引脚、控制引脚、晶振及电源引脚进行简要介绍。



（a）封装 1　　（b）封装 2

（c）封装 3

图 1-1　8051 单片机不同封装形式及引脚图

8051 的 4 个双向 I/O 端口功能如下所述。

（1）P0 端口具有双重功能，它可用做 I/O 端口，连接输入/输出设备，但需要外接上拉电阻，P0 端口还能用做数据总线（D0~D7）和低 8 位地址总线（A0~A7）复用端口，低 8 位地址与数据分时使用 P0 端口，低 8 位地址由 ALE（Address Latch Enable，地址锁存允许）信号的下降沿锁存到外部地址锁存器中，然后传送 8 位数据。本书有关 2764、6264 和 8255 的案例中使用了 P0 端口的第二重功能。

（2）P1 端口是通用 I/O 端口，每一位都能作为 I/O 接口线使用，本书有关独立按键或键盘矩阵按键的案例中，按键多数连接在 P1 端口。

（3）P2 端口具有双重功能，它可以作为 I/O 端口，连接输入/输出设备，在扩展外部存储器或扩展 I/O 接口时，P2 端口作为系统扩展时 16 位地址总线的高 8 位（A8~A15）使用，低 8 位地址由 P0 端口提供。本书有关存储器扩展和接口扩展的案例使用了 P2 端口的第二重功能。

（4）P3 端口具有双重功能，功能一与 P1 端口相同，作为功能二使用时，P3.0~P3.7 分别对应于 RXD、TXD、INT0、INT1、T0、T1、WR 及 RD，第 3 章基础程序设计中有关串口、外部中断及定时/计数器的案例分别涉及前 6 位，第 4 章硬件应用中关于存储器扩展和接口扩展的案例连接了最后 2 位 WR 和 RD，这 2 位实际上用于控制总线。

8051 的控制引脚有 RST，ALE/$\overline{PROG}$，$\overline{PSEN}$，$\overline{EA}$/VPP，本书多数案例中的 RST（Reset）引脚连接了系统复位外围电路，只有部分案例未在 RST 引脚连接复位电路，但这并不影响 Proteus 的仿真，ALE 引脚在存储器扩展和 I/O 接口扩展案例中用到，另外，由于本书所有案例程序都绑定在 8051 内部程序 ROM 中，案例中 EA 全部连接高电平。

XTAL1，XTAL2 引脚在本书多数案例中连接 12MHz 晶振，在串口通信案例中本书选择的是 11.0592MHz 晶振。Proteus 缺省晶振连接时不影响仿真运行，频率可在芯片属性中直接设置。主电源引脚 VSS，VCC 分别接 GND 和+5V，Proteus 仿真时电源已默认连接，仿真电路原理图中它们全部缺省。当然，在实际硬件环境中，电源与晶振部分不能缺省。

1.2 数据与程序内存

8051 单片机存储器组织结构如图 1-2 所示，存储器分为以下 4 部分：

（1）片内程序存储器（ROM）4KB；

（2）片外程序存储器（ROM）64KB（含内部 4KB）；

（3）片内数据存储器（RAM）256B；

（4）片外数据存储器（RAM）64KB。

这 4 部分中的前 2 部分是程序 ROM，片内程序 ROM 用于存放 8051 控制程序，本书各案例中 8051 单片机绑定的都是由 C 语言程序编译链接生成的 HEX 程序文件，这相当在硬件环境下向 4KB 的 8051 单片机内部程序 ROM 烧写 HEX 程序，要从片内 ROM 的 4KB 存储器取指令时，需要将 EA 接高平。8051 外部程序 ROM 可扩展到 64KB，地址范围为 0000H~0FFFH 的 4KB 内外 ROM 地址是重叠的，使用过程中要用 EA 的取值来区分，对于有内部 ROM 空间的 8051 单片机，EA 可接高电平，单片机运行时会从内部程序 ROM 的 0000H 地址开始执行，当程序计数器（PC）的值超过 0FFFH 时，自动转换到片外程序 ROM 的 1000H~FFFH 地址空间执行。对于没有内部程序 ROM 的 8051 单片机，程序存放在外部 ROM 中，其 EA 引脚固定接低电平。

8051 的数据 RAM 空间虽然很小，但它起着非常重要的作用，256B（字节）被分为两个区，00H-7FH 的 128B 空间是真正的 RAM 区，可读/写各种数据；80H~FFH 的 128B 空间大部分专门

用于特殊功能寄存器（SFR：Special Function Register），8051 在这个空间安排了 21 个特殊功能寄存器，不论是用汇编语言还是用 C 语言编写单片机程序，这些特殊功能寄存器都要重点掌握。

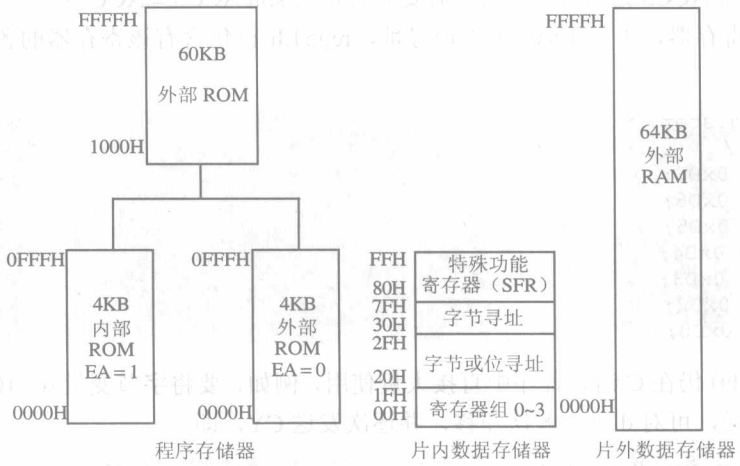


图 1-2　8051 单片机存储器结构

1.3　特殊功能寄存器

由图 1-2 可知，片内数据 RAM 中 80H~FFH 空间大部用于特殊功能寄存器，表 1-1 列出了 8051 单片机的所有特殊功能寄存器。

表 1-1　8051 单片机的特殊功能寄存器

符号	地址	说　明	符号	地址	说　明
P0*	0x80	P0 端口	TCON*	0x88	定时器控制
P1*	0x90	P1 端口	TMOD	0x89	定时器方式选择
P2*	0xA0	P2 端口	TL0	0x8A	定时器 0 低 8 位
P3*	0xB0	P3 端口	TL1	0x8B	定时器 1 低 8 位
PSW*	0xD0	程序状态字	TH0	0x8C	定时器 0 高 8 位
ACC*	0xE0	累加器	TH1	0x8D	定时器 1 高 8 位
B*	0xF0	乘法寄存器	IE*	0xA8	中断允许寄存器
SP	0x81	堆栈指针	IP*	0xB8	中断优先级寄存器
DPL	0x82	数据存储器指针 DPTR 低 8 位	SCON*	0x98	串行口控制器
DPH	0x83	数据存储器指针 DPTR 高 8 位	SBUF	0x99	串行数据缓冲
PCON	0x87	电源控制及波特率选择			

上述特殊功能寄存器分别用于以下功能单元。

CPU：ACC、B、PSW、SP、DPTR。

并行口：P0、P1、P2、P3。

中断系统：IE、IP。

定时器/计数器：TMOD、TCON、T0（TH0，TL0）、T1（TH1，TL1）。

串行口：SCON、SBUF、PCON。

本书用 C 语言开发单片机系统时，由于 ACC 可以位寻址，它常被用于判断字节中各位的状态，例如，要判断某字节第 3 位是否为 1，可将 ACC 看成一个字节变量，让 ACC 获取该字节的值，然后直接判断 ACC3 是否为 1，当然需要先有定义 sbit ACC3 = ACC^3。

又如 PSW 寄存器，由于 PSW 可以位寻址，reg51.h 已包含有该寄存器的各位定义，定义片断如下：

```
/*  BIT Register  */
/*  PSW   */
sbit CY  = 0xD7;
sbit AC  = 0xD6;
sbit F0  = 0xD5;
sbit RS1 = 0xD4;
sbit RS0 = 0xD3;
sbit OV  = 0xD2;
sbit P   = 0xD0;
```

其中 CY 和 F0 仍在 C 语言程序中直接大量使用，例如，要将字节变量 d（10101101）由高位开始逐位串行发送，可对 d 进行 8 次左移，并逐次发送 CY，即

```
for (i = 0 ; i < 8 ; i++)
{
  d <<= 1;
  DQ = CY;
}
```

代码中 DQ 为某外部芯片接收串行数据的引脚，这段代码利用汇编语言程序中常用的进位标志位 CY，可以方便地获取移出的各位。

表 1-1 中带有*号的特殊功能寄存器是可位寻址的，Keil\c51 下头文件 reg51.h 对 P0~P3 以外的可位寻址的各寄存器位给出了独立定义，如果要直接引用 4 个 I/O 端口的各引脚，需要在程序中用 sbit 进行单独定义，当然也可以将头文件 reg51.h（keil\c51\inc）改成 at89x52.h（keil\c51\inc\atmel），4 个端口的各引脚在该头文件中均被单独定义，如 P0.1 被定义为 P0_1，尽管其中有些定义对 8051 单片机是无效的，但这并不影响程序正常编译运行。

1.4 外部中断、定时/计数器及串口应用

使用 C 语言开发 8051 单片机程序时，除了要控制 4 个双向 I/O 端口外，还要掌握 8051 外部中断、定时/计数器及串口的中断程序设计。当然，在应用定时/计数器及串口时，它们既可以工作于中断方式，又可以工作于非中断方式，读者在本书中可找到它们工作于不同方式的案例。

8051 的 5 个中断源分别是：

（1）外部中断请求 0，由 INT0（P3.2）输入；

（2）外部中断请求 1，由 INT1（P3.3）输入；

（3）片内定时/计数器 0 溢出中断请求；

（4）片内定时/计数器 1 溢出中断请求；

（5）片内串行口接收/发送中断请求。

这 5 个中断源的中断号分别是 0、2、1、3、4，用 C 语言编写中断程序时，中断函数后要添加 interrupt n，其中 n 为中断号。

在设计中断程序时需要用到的字节 SFR 和位 SFR（sbit）如表 1-2 所示。

表 1-2　8051 中断控制、定时/计数器及串口控制寄存器

寄存器名称	寄存器符号	寄存器位符号							
		D7	D6	D5	D4	D3	D2	D1	D0
中断允许	IE	EA		ET2	ES	ET1	EX1	ET0	EX0
中断优先级	IP				PS	PT1	PX1	PT0	PX0
T/C 控制	TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
T/C 方式控制	TMOD	GATE	C/T	M1	M0	GATE	C/T	M1	M0
串行口控制	SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
电源控制	PCON	SMOD				GF1	GF0	PD	IDL
注：对于 PCON，本书中主要使用其高位 SMOD 设置波特率是否倍增。									

由于这些 SFR 都是可以位寻址的，因此在程序设计中，既可以直接给字节 SFR 赋值，也可以对 SFR 中的相应位赋值，例如，某程序同时允许外部 0 中断和定时器 0 中断，读者可编写代码：

```
IE = 0x83;  //IE 被设为 10000011.
```

或者写成：

```
EX0 = 1;
ET0 = 1;
EA = 1;
```

和 1.3 节中的 PSW 一样，这些 SFR 中的位也定义在 reg51.h 中，下面给出了 reg51.h 中的部分定义：

```
/*  IE   */
sbit EA  = 0xAF;
sbit ES  = 0xAC;
sbit ET1 = 0xAB;
sbit EX1 = 0xAA;
sbit ET0 = 0xA9;
sbit EX0 = 0xA8;
```

这些位定义中的大多数会在本书大量案例中使用，读者需要熟练掌握它们各自的含义和用途。更完整的定义可在 at89x52.h 文件中找到，读者可根据需要选择 reg51.h 或 at89x52.h。

1.5　有符号与无符号数应用、数位分解、位操作

Keil C51 的数据类型如表 1-3 所示。

表 1-3　Keil C51 的数据类型

数据类型	本书重定义	长度（位）	值　域
bit		1	0,1
unsigned char	uchar	8	0~255
signed char（char）		8	−128~127
unsigned int	uint	16	0~65 535
signed int（int）		16	−32 768~32 767
unsigned long	ulong	32	0~4 294 967 295
signed long（long）		32	−2 147 483 648~2 147 483 647

续表

数据类型	本书重定义	长度（位）	值　域
float		32	±1.175494E-38~±3.402823E+38
sbit		1	0,1
sfr		8	0~255
sfr16		16	0~65 535

本书大多数案例使用的都是无符号数，对于 255 以内的整数，本书全部定义为 uchar 类型（相当于字节类型 byte），对于 0~65 535 范围内的整数，本书定义为 uint 类型（相当于字类型 word）。有关案例因为涉及正负数的处理，例如，温度控制程序中有零上温度与零下温度，由于温度管实际上可处理的范围为-55℃~125℃，为了使程序对温度值进行正确比较，程序中将温度类型定义为 char 类型（即 signed char 类型，其取值范围为-128~127）。

另外，本书大量用到的延时程序中既有 uchar 类型，也有 uint 类型，如果读者在编写代码时，给某变量赋值为 2000，但该变量类型为 uchar，编译程序并不会报错，但变量实际上并没得到所期望的值。

本书大量案例要将整数或浮点数显示在数码管上，这需要对待显示数据各数位进行分解，例如：

```
uchar d = 124;
uchar c[3];
c[0] = d / 100;
c[1] = d / 10 % 10;
c[2] = d % 10;
```

又如：

```
float x = 123.45;
```

如果要得到 x 的各个数位，可以先将 x 乘以 100，然后再分解各数位：

```
uint y = x*100;
uchar c[5];
uchar i;
for (i = 4 ; i!=0xFF; i--)
{
  c[i] = y % 10;
  y /= 10;
}
```

上面 for 循环中的循环条件本来要写成 i>=0 的，但由于当 i = 0 时，如果将 i 再减 1，i 变为 0xFF，这个无符号数仍被认为≥0，这样就不能保证 5 次循环了，因此要改写成 i!=0xFF，如果将 i 定义成 char 类型而不是 uchar 类型，使用 i>=0 时才能得到正确结果。

上述分解常用于数码管数字显示，因为显示时需要根据各数位提取数码管段码。

在本书案例中会大量出现位操作，例如，在有关 LED 流水灯、数码管位控制、串行收发、键盘扫描等大量案例中，位的各操作符和有关函数都会频繁使用，例如，字节位循环左移函数（_crol_）、循环右移函数（_cror_），位左移（<<）、右移（>>）、与（&）、或（|）、取反（~）、异或（^）、非（!）等，读者都要熟练掌握。

下面是有关位操作的几个简单应用。

例如，要 P1 端口 P1.7~P1.0 逐位循环轮流置 1，可先设字节变量 c = 0x01，然后在循环语句中执行 c = _cror_(c,1)，并使 P1 = c，这样即可使 P1 端口出现 10000000，01000000，00100000，…，

00000001，如此重复。如果要 P1.7 至 P1.0 逐位循环轮流置 0，可先设 c = 0xFE，然后使用同样的循环移位操作即可。这样的位操作在 LED 流水灯或集成式数码管位码控制时常会用到。

又如，已知 P2.3 连接外部 LED 或蜂鸣器，如果需要 LED 闪烁或蜂鸣器发声，可先定义 sbit LED = P2^3 或 sbit BEEP = P2^3，然后在循环中执行语句 LED = ~LED 或 BEEP = ~BEEP 即可，注意，对于独立的位定义，执行取反（~）与非（!）操作效果是一样的，但对于字节变量则是不同的，这一点要注意。

另外，在本书 4×4 键盘矩阵扫描程序中，假设 P1 高 4 位连接矩阵行，低 4 位连接矩阵列，为判断整个键盘中是否有键按下，通常会先在行上发送 4 位扫描码 0000，即 P1 = 0x0F，然后检查列上是否出现 0，这时使用的位操作语句是：

```
if ( (P1 & 0x0F) != 0x0F) {//有键按下}
```

因为!=的优先级高于&，该语句中的与操作表达式要加上括号。

本书多个案例使用了字符液晶，向连接在 P0 端口的液晶屏发送显示数据时，需要先判断液晶是否忙，其忙标志在读取字节的最高位，因此又有类似语句：

```
if ( (P0 & 0x80) == 0x80) {//液晶忙}
```

1.6 变量、存储类型与存储模式

全局变量定义在函数外面，生命期从所定义的地方开始，其后面的所有函数都可以读/写该全局变量，本书案例在使用定时器时，为得到更大的延时值，程序中定义了全局变量 tCount，在定时器中断内对该变量累加，从而得到更大的延时间隔，如果程序中其他地方不需要使用 tCount，该变量可以不必放在程序最前面，而是放在定时器中断函数的上一行。

局部变量定义在函数内部，对于循环及其他的临时计算，应尽可能使用局部变量，作为优化处理的一部分，编译程序会试图将局部变量维持在寄存器中，寄存器访问是最快的内存访问类型，特别是 unsigned char 和 unsigned int 类型的变量。

对于刚才讨论的 tCount 变量，在 C 语言程序设计时，还可以定义在中断函数内部实现计时累加，但要注意在定义前面加上 static，因静态变量仅在函数首次调用时被初始化一次，此后不再被初始化。

8051 系列单片机中，程序存储器与数据存储器严格分开，特殊功能寄存器与片内数据存储器统一编址，这与一般微型计算机的存储结构是不同的。

Keil C51 编译器完全支持 8051 单片机硬件结构，可完全访问 8051 硬件系统的所有部分，编译器通过将变量、常量定义成不同的存储类型（data、bdata、idata、pdata、xdata、code），从而将它们定位在不同的存储区中。表 1-4 列出了 Keil C51 存储类型与 8051 存储空间的对应关系。

表 1-4 Keil C51 存储类型与 8051 存储空间的对应关系

存储类型	与存储空间的对应关系	地址范围
data	直接寻址片内数据存储区（前 128B）	00H~7FH
bdata	可位寻址片内数据储存区，允许位与字节混合访问（16B）	20H~2FH
idata	间接寻址片内数据储存区，可访问片内全部 RAM 空间	00H~FFH
pdata	分页寻址片外数据存储区（256B），编译后由 MOVX @Rn 访问，地址高 8 位保存在 P2 端口	00H~FFH
xdata	片外数据储存区（64KB），编译后由 MOVX @DPTR 访问	0000H~FFFFH
code	代码存储区（64KB），编译后由 MOVC @A+DPTR 访问	0000H~FFFFH

读者在阅读表 1-4 时可参考图 1-1 所示的 8051 单片机存储器结构。

上述存储类型指示符 data、bdata、idata 将数据定位在内部数据存储器中，访问内部数据速度很快，它们只需要 8 位地址。

Keil C51 编译器提供了两种外部存储类型：xdata 和 pdata，指定为 xdata 存储器类型的数据保存在最大空间为 64KB 的外部数据存储器空间中，通过 P0 与 P2 端口给出的 16 位地址可访问外部空间中的任意位置，但是，并非 64KB 地址空间总是用于存储器，单片机的外围设备也可以映射到存储器空间，在这种情况下，程序将会以与访问外部数据内存同样的方法编程控制外围设备，这种技术被称为内存映射的 I/O 技术，本书有关 8255 和部分液晶显示案例使用的就是这种技术。第二种存储类型 pdata 可访问外部数据内存一页中的 256B，页地址则由 P2 提供。

本书有关案例在扩展外部存储器或外部设备时，程序中包含了绝对内存访问（Absolute Memory Access）头文件 absacc.h，该文件中有宏定义：

```
#define XBYTE ((unsigned char volatile xdata *) 0)
```

由其中的 XBYTE 定义可知，表达式 XBYTE[16 位字地址]可用来读/写外部 RAM 空间的字节数据，本书 6264 扩展内存案例就使用了 XBYTE 访问外部数据存储器，它相当于汇编语言中的 MOVX @DPTR 语句。

在 8255 接口扩展实验中，C 语言程序内有：

```
#define PA   XBYTE[0x0000]
#define PB   XBYTE[0x0001]
#define PC   XBYTE[0x0002]
#define COM  XBYTE[0x0003]
```

这里 PA、PB、PC、COM 分别用于访问 8255 的 4 个不同端口。

上述定义还可以写成：

```
#define PA   *(XBYTE + 0x0000)
#define PB   *(XBYTE + 0x0001)
#define PC   *(XBYTE + 0x0002)
#define COM  *(XBYTE + 0x0003)
```

本书涉及数码管显示和图像与文字显示的案例中，由于数码管段码固定，待显示的图像或文字点阵数据也是固定的，将这些数据全部保存在 RAM 中会占用太多宝贵的空间，而且数据 RAM 空间本来就很有限，因此，在用 C 语言开发单片机程序时，可以将这些运行过程中不会发生变化的数据定义为 code 存储类型，将这些数据保存在程序内存而不是数据内存。

在定义变量时，如果省略存储类型，编译程序会自动选择默认存储类型，默认存储类型有小模式（Small）、紧缩模式（Compact）和巨模式（Large）限制，存储模式决定了变量的默认存储类型、参数传递区和未指明存储类型变量的存储类型。表 1-5 列出了这 3 种存储模式及相关说明。

表 1-5　3 种存储模式及相关说明

存储模式	参数及局部变量分配	默认存储类型/空间大小
Small	放入可直接寻址的片内数据存储区	data/128B
Compact	放入片外分页存储区	pdata/256B
Large	放入片外数据存储区	xdata/64KB

在固定的存储器地址上进行变量传递是 Keil C51 的特征之一，在 Small 模式下，参数传递在片内数据存储区完成，Compact 和 Large 模式允许参数在外部存储区中传递。模式选择可在 Keil C51 项目选项窗口中的 Target 选项卡下完成，默认选择的是 Small 模式。

1.7 数组、字符串与指针

本书大量案例使用了数组定义，例如，下面的数组 DSY_CODE 定义了 0~9 的七段数码管段码表：

```
uchar code DSY_CODE[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
```

由于在程序运行过程中 DSY_CODE 的数据保持不变，因此，这里将存储类型设为 code，如果将 code 改为 data 也不会影响程序的运行，只是在程序运行时数组会被分配到数据 RAM，而不是仅占用程序 ROM 空间，在 Small 模式下，省略 code 相当于将程序存储类型设为默认的 data 类型。

读者在实际设计时，如果要将变化的数组定义在数据 RAM 中，由于 data 类型仅允许使用 128B，如果编译时提示数据 RAM 空间不够，可尝试将 data 改为 idata，例如：

```
uchar idata Sort_Result[200];
```

另外，存储类型 code、data 和 idata 还可以放到数据类型前面。

字符串类型也在本书大量案例中使用，例如，下面的字节串定义：

```
char s[20] = "Current Voltage:";
char s[20] = {"Current Voltage:"};
char s[20] = {'R','e','s','u','l','t',':'};
```

这 3 种定义是完全相同的，它们都占用 20B 空间，实际串长为 16 个字符，最后未明确赋值的 4B 全部为 0x00(即'\0')，在液晶屏上显示这类字符串时，可用以下方法：

```
(1) for(i = 0; i < 16; i++){//显示字符 s[i]};
(2) for(i = 0; i < strlen(s1); i++){//显示字符 s[i]};
(3) i = 0; while (s[i++]!= '\0') {//显示字符 s[i]};
```

要注意的是，如果字符串长为 16，而字符数据空间也只固定给出了 16B，那么上述方法中的后两种就不可靠了，因为最后一个字符后面不一定是字符串结束标志'\0'。

字符串还可以这样定义：

```
char s[] = "Current Voltage:";
char *s = "Current Voltage:";
```

这两种定义也是相同的，其串长均为 16 个字符，所占用的字节空间均为 17B，因为字符串末尾被自动附加了结束标志字节 0x00(即'\0')。

在已知串长时，上述三种字符串显示方法均可使用，在字符串长未知时可使用上述方法中的后两种，另外，上述显示方法还可以改写成：

```
(1) for(i = 0; i < 16; i++){//显示字符*(s+i)};
(2) for(i = 0; i < strlen(s1); i++){//显示字符*(s+i)};
(3) i = 0; while (*(s+i)!= '\0') {//显示字符*(s+i);i++;};
```

在编写 C 语言程序时，除使用字符数组（字符串）外，还会使用字符串数组，例如：

```
char s[][20] = {"Current Voltage:","Counter:      ","TH:     TL:     "};
```

如果要在液晶上显示“Counter:”这个字符串，可用以下语句实现：

```
for(i = 0; i < strlen(s[1]);i++){//显示字符 s[1][i]};
```

在英文字符液晶上显示数值时需要将待显示数据转换为字符串，这时可用此前提到的数据位分解方法，先分解出各位数字，然后加上 0x30（即'0'）得到对应数字的 ASCII 编码。另一种更为