从零开始学电子技术丛书

从零开始学

单片机C语言

刘建清　主编
刘建清　寻立波　陈培军　编著

随书附光盘一张

国防工业出版社
National Defense Industry Press

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

责任编辑：杨星豪　xhyang@ndip.cn
文字编辑：李　燕
责任校对：钱辉玲
封面设计：王晓军　xjwang@ndip.cn

从零开始学电子技术丛书

从零开始学电路仿真Multisim与电路设计Protel技术
从零开始学电气控制与PLC技术
从零开始学电子测量技术
从零开始学CPLD和Verilog HDL编程技术
从零开始学单片机C语言
从零开始学单片机技术
从零开始学电路基础
从零开始学元器件识别与检测
从零开始学电动机控制与维修
从零开始学模拟电子技术
从零开始学数字电子技术

◎ 上架建议：电子技术 ◎
http://www.ndip.cn
ISBN 7-118-04643-4
9 787118 046434 >

ISBN 7-118-04643-4/TP · 1066
定价：34.00 元（含光盘）

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

从零开始学电子技术丛书

从零开始学单片机C语言

刘建清　主编
刘建清　寻立波　陈培军　编著

国防工业出版社
·北京·

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

内 容 简 介

本书定位于让初学者从零起步，轻松学会单片机高级编程 C 语言以及实战技术。书中首先简要介绍了 8051 单片机的主要构成，然后重点介绍了单片机 Cx51 高级编程语言开发与仿真环境的使用方法，透彻地分析了 Cx51 的语法结构，并给出了大量实例。书中大多数实例和全部实验都经过了实验板的验证。

本书附赠一张配套光盘，光盘中包含书中所有实验的源程序。

本书可供具有一定单片机基础的初学者，以及广大从事单片机应用系统开发研制的工程技术人员阅读，也可以作为有关院校相关专业的教学参考用书。

图书在版编目(CIP)数据

从零开始学单片机 C 语言/刘建清主编；刘建清，寻立波，陈培军编著. —北京：国防工业出版社，2006.8

(从零开始学电子技术丛书)

ISBN 7-118-04643-4

Ⅰ. 从… Ⅱ. ①刘…②刘…③寻…④陈…

Ⅲ. 单片微型计算机-C 语言-程序设计

Ⅳ. TP368.1②TP312

中国版本图书馆 CIP 数据核字(2006)第 077660 号

※

国防工业出版社出版发行

(北京市海淀区紫竹院南路 23 号 邮政编码 100044)

北京奥鑫印刷厂印刷

新华书店经售

*

开本 787×1092 1/16 **印张** 19¼ **字数** 439 千字

2006 年 8 月第 1 版第 1 次印刷 **印数** 1—5000 册 **定价** 34.00 元(含光盘)

(本书如有印装错误，我社负责调换)

国防书店：(010)68428422 发行邮购：(010)68414474

发行传真：(010)68411535 发行业务：(010)68472764

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

丛书前言

我们所处的时代是一个知识爆炸的新时代。新产品、新技术层出不穷，电子技术的发展更是日新月异。可以毫不夸张地说，电子技术的应用无处不在，电子技术正在不断地改变着我们的生活，改变着我们的世界。

读者朋友：当你对妙趣横生的电子世界发生兴趣时；当你彷徨于就业的关口，想成为电子产业中的一名员工时；当你跃跃欲试，想成为一名工厂的技术革新能手时；当你面对“无所不能”的“单片机”，梦想成为一名自动化高手时；当你的头脑里冒出那么多的奇思妙想，急于把它们应用于或转化为产品时……都是那么急切地想补充自己有关电子技术方面的知识，这时，你首先想到的是找一套适合自己学习的电子技术图书阅读。《从零开始学电子技术丛书》正是为了满足广大读者特别是电子爱好者的实际需要和零起点入门的阅读要求而编著的。

和其他电子技术类图书相比，本丛书具有以下特点：

内容全面，体系完备。本丛书给出了广大电子爱好者学习电子技术的全方位解决方案，既有初学者必须掌握的电路基础、模拟电路和数字电路等基础理论，又有电子元器件检测、电子测量仪器的使用、电路仿真与设计等操作性较强的内容，还有电气控制与 PLC、单片机、CPLD 等综合应用方面的知识，因此，本丛书内容翔实，覆盖面广。

通俗易懂，重点突出。传统的电子技术图书和教材在介绍电路基础和模拟电子技术等内容时，大都借助高等数学这一工具进行分析，这就给电子爱好者自学电子技术设置了一道门槛，使大多数电子爱好者失去了学习的热情和兴趣。本丛书在编写时，完全考虑到了初学者的需要，不涉及高等数学方面的公式，尽可能地把复杂的理论通俗化和实用化，将烦琐的公式简易化，再辅以简明的分析及典型的实例，从而形成了本丛书通俗易懂的特点。为了满足不同层次读者的需求，本丛书对难点和扩展知识用“*”进行了标注，初学者可跳过此内容。

实例典型，实践性强。本丛书最大程度地强调了实践性，书中给出的例子大都经过了验证，可以实现，并且具有代表性；本丛书中每本书都配有光盘，光盘中收录了书中的实例、常用软件、实验程序和大量珍贵资料，以方便读者学习和使用。

内容新颖，风格活泼。本丛书所介绍的都是电子爱好者最为关心并且在业界获得普遍认同的内容，本丛书的每一分册都各有侧重，又互相补充，论述时疏密结合，重点突出。对于重点、难点和容易混淆的知识，书中还特别进行了标注和提示。

把握新知，结合实际。电子技术发展日新月异，为适应时代的发展，本丛书还对电子技术的新知识做了详细的介绍；本丛书中涉及的应用实例都是编著者开发经验的提炼和总结，相信一定会给读者带来很大的帮助。在讲述电路基础、模拟和数字电子技术时，还专门安排了计算机辅助软件的仿真实验，实验过程非常接近实际操作的效果，使电子技术的学习变得更为直观，使学习变得更加生动有趣，这可以加深读者对电路理论知识的认识。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

总之，对于需要学习电子技术的电子爱好者而言，选择《从零开始学电子技术丛书》不失为一个好的选择。本丛书一定能给你耳目一新的感觉，当你认真阅读之后将会发现，无论是你所读的书，还是读完书的你，都有所不同。

感谢本丛书的策划者——电子科普领域中的知名专家、中国电子学会高级会员刘午平先生，他与我们共同交流，共同探讨，达成了共识，确立了写作方向，并为本丛书的编排、修改和出版做了大量卓有成效的工作，他以丰富的专业知识和认真、敬业的态度为我们所敬佩；感谢山东持恒开关厂总经理陈培军先生和山东金曼克电气集团设计处总工程师高广海先生，他们对本丛书的编写提出了很多建设性的意见和建议，为本丛书的许多实验提供了强有力的支持与帮助，并参与了部分图书的编写工作；感谢网络，本丛书的许多新知识、新内容都是我们通过网络而获得的，我们在写作过程中遇到的许多疑难问题也大都通过网络得以顺利解决，对于这么多乐于助人、无私奉献的站主和作者们，无法在此一一列举，只能道一声“谢谢了！”感谢众多电子报刊、杂志的编辑和作者，他们为本丛书提供了许多有新意、有实用价值的参考文献，使得这套丛书能够别出心裁、与时俱进；感谢国防工业出版社，能与国内一流的出版社合作，我们感到万分的荣幸；感谢其他对本丛书的出版付出过辛勤工作的人士，没有他们的热心与支持，本丛书不知何时才能与读者见面！

最后，祝愿本丛书的每一位读者在学习电子技术的过程中，扬起风帆，乘风破浪！

丛书编者

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

前　言

C语言已成为当前举世公认的高效简洁而又贴近硬件的编程语言之一。将C语言向单片机上的移植，始于20世纪80年代的中后期。经过十几年的努力，C语言终于成为专业化单片机上的实用高级语言。用C语言编写的8051单片机的软件，可以大大缩短开发周期，且明显地增加软件的可读性，便于改进和扩充，从而研制出规模更大、性能更完善的系统。因此，不管是对于新进入这一领域的开发者来说，还是对于有多年单片机开发经验的人来说，学习单片机的C语言编程技术都是十分必要的。

本书简要介绍了8051单片机的主要构成、单片机Cx51开发与仿真环境的使用方法，透彻分析了Cx51的语法结构，并给出了大量实例。书中大多数实例和全部实验都经过了实验板的验证。

本书的主要读者对象是面向有一定单片机基础的初学者，因此建议读者在阅读本书之前，先阅读《从零开始学单片机技术》一书，以达到融会贯通的目的。从我们多年来学习、开发的经验来看，单片机C语言编程技术是一门实战性非常强的学科，除了不断地学习之外，更重要的是要不断地实践！

本书附有光盘，光盘中包含有书中所有实验的源程序。

由于时间仓促，书中错漏之处在所难免，敬请广大读者批评指正。

作　者

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

目　录

第一章　单片机基础知识 …… 1

第一节　51 系列单片机概述 …… 1

一、什么是单片机 …… 1

二、51 单片机的发展 …… 1

三、51 单片机主要技术指标 …… 2

四、51 单片机主要型号的特点 …… 2

第二节　80C51 单片机的内部结构和外部引脚 …… 3

一、80C51 单片机的内部结构框图 …… 3

二、单片机的外部引脚 …… 5

第三节　80C51 单片机内部存储器的配置 …… 7

一、程序存储器 ROM …… 7

二、片内数据存储器 RAM …… 8

三、片外数据存储器 RAM …… 10

第四节　51 系列单片机的并行输入/输出接口 …… 10

一、P0 口 …… 10

二、P1 口 …… 12

三、P2 口 …… 13

四、P3 口 …… 13

第五节　80C51 单片机的时钟电路和复位电路 …… 14

一、单片机的时钟电路 …… 14

二、单片机的复位电路 …… 15

三、单片机的低功耗方式 …… 16

第二章　单片机 C 语言入门 …… 19

第一节　认识 C 语言 …… 19

一、C 语言的发展过程 …… 19

二、C 语言的特点 …… 19

三、单片机采用 C 语言编程的好处 …… 19

第二节　简单 C 语言程序的构成 …… 20

一、简单 C 语言程序 …… 20

二、C 语言程序的构成 …… 23

第三节　单片机 C 语言开发步骤 …… 24

一、硬件系统设计 …… 24

二、编写 C 语言程序 …… 25

三、编译源程序 …… 25

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

四、应用程序的仿真调试 …… 25
五、单片机应用程序的烧写 …… 26
六、系统脱机运行检查 …… 26
第三章　单片机实验硬件环境的建立 …… 27
第一节　单片机实验板 …… 27
一、下载型实验板 …… 27
二、AT89C51 实验板 …… 31
第二节　单片机仿真器 …… 35
一、Insight SE-52 仿真器 …… 36
二、MON51 仿真器 …… 36
第三节　单片机编程器 …… 38
一、RF-810 编程器 …… 38
二、下载型编程器 …… 45
第四章　Keil Cx51 软件的使用 …… 47
第一节　Keil Cx51 简介 …… 47
一、Keil Cx51 的特点 …… 47
二、Cx51 与 ANSI C 的主要区别 …… 48
第二节　Keil Cx51 软件的安装 …… 48
第三节　uVision2 集成开发环境 …… 52
一、uVision2 软件的启动 …… 52
二、建立一个新工程 …… 53
三、工程的设置 …… 58
四、程序的编译和链接 …… 61
五、程序的调试 …… 62
六、用实验板进行仿真实验 …… 65
第四节　常用窗口介绍 …… 66
一、变量观察窗口 …… 66
二、存储器观察窗口 …… 67
三、寄存器观察窗口 …… 69
四、串口调试观察窗口 …… 69
五、反汇编观察窗口 …… 69
六、外围设备观察窗口 …… 70
七、代码作用范围分析窗口 …… 71
八、输出窗口 …… 72
第五章　Cx51 数据与运算 …… 73
第一节　标识符和关键字 …… 73
一、标识符 …… 73
二、关键字 …… 73
第二节　数据类型 …… 75

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

一、数据类型简介 …… 76
二、实验演练 …… 78
第三节　常量 …… 79
一、常量的数据类型 …… 79
二、用宏表示常数 …… 80
三、常量定义 …… 81
第四节　变量 …… 81
一、变量的初始化 …… 81
二、变量的定义及存储模式 …… 82
三、变量的种类 …… 84
四、实验演练 …… 85
第五节　运算符和表达式 …… 86
一、赋值运算符及其表达式 …… 86
二、算术运算符及其表达式 …… 86
三、关系运算符及其表达式 …… 87
四、逻辑运算符及其表达式 …… 88
五、位运算符及其表达式 …… 88
六、复合赋值运算符及其表达式 …… 89
七、其他运算符及其表达式 …… 90
八、实验演练 …… 93
第六章　Cx51 的基本语句 …… 100
第一节　表达式语句和复合语句 …… 100
一、表达式语句 …… 100
二、复合语句 …… 101
三、实验演练 …… 101
第二节　条件选择语句 …… 103
一、if 语句及实验演练 …… 103
二、switch 语句及实验演练 …… 106
第三节　循环语句 …… 107
一、while 语句及实验演练 …… 108
二、do while 语句及实验演练 …… 112
三、for 语句及实验演练 …… 114
四、break 和 continue 语句在循环体中的作用 …… 117
第七章　Cx51 函数 …… 121
第一节　函数的分类和定义 …… 121
一、函数的分类 …… 121
二、函数的定义 …… 122
三、实验演练 …… 124
第二节　函数的参数和返回值 …… 127

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

一、函数的参数 …… 127
二、函数的返回值 …… 128
三、实验演练 …… 128
第三节　函数的调用 …… 129
一、函数调用的一般形式 …… 129
二、函数调用的方式 …… 130
三、对被调用函数的说明 …… 130
四、函数的嵌套 …… 131
五、函数的递归调用与再入函数 …… 131
六、实验演练 …… 133
第四节　函数变量的存储方式和种类 …… 135
一、变量的存储方式 …… 135
二、变量的存储种类 …… 135
第八章　Cx51 构造数据类型 …… 140
第一节　数组 …… 140
一、一维数组 …… 140
二、二维数组 …… 142
三、字符数组 …… 143
四、实验演练 …… 144
第二节　指针 …… 147
一、指针的基本概念 …… 147
二、指针变量的定义 …… 149
三、初始化指针变量 …… 149
四、指针变量的运算 …… 150
五、指针和数组 …… 153
六、指针变量作为函数参数 …… 157
七、一般指针和基于存储器的指针 …… 158
八、绝对地址的访问 …… 159
九、实验演练 …… 160
第三节　结构 …… 164
一、结构的定义 …… 164
二、结构类型变量的说明 …… 165
三、结构变量的引用 …… 166
四、结构变量的赋值 …… 167
五、结构数组 …… 167
六、结构指针变量 …… 168
第四节　共用体 …… 169
一、共用体的定义 …… 170
二、共用体变量的说明 …… 170

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第五节　枚举 …… 171
一、枚举类型的定义 …… 171
二、枚举变量的说明 …… 172
三、枚举类型变量的赋值和使用 …… 173
第九章　单片机中断、定时器和串行口的C语言编程 …… 174
第一节　单片机中断系统C语言编程 …… 174
一、什么是中断 …… 174
二、中断源 …… 174
三、中断控制 …… 175
四、中断的响应 …… 178
五、中断的撤除 …… 180
六、实验演练 …… 181
第二节　定时/计数器及实验 …… 184
一、定时/计数器概述 …… 184
二、定时/计数器的控制寄存器 …… 186
三、定时/计数器的工作方式 …… 187
四、实验演练 …… 191
第三节　串行数据通信技术及实验 …… 204
一、串行数据通信概述 …… 204
二、串行口的基本结构 …… 208
三、串行通信控制寄存器 …… 209
四、串行口工作方式 …… 212
五、实验演练 …… 216
第十章　单片机实用接口C语言编程技术 …… 219
第一节　LED显示器接口 …… 219
一、8段LED显示器的结构及原理 …… 219
二、LED显示器的显示方式 …… 221
三、实验演练 …… 223
第二节　键盘接口 …… 232
一、键盘的工作原理 …… 232
二、键盘与单片机的连接 …… 234
三、键盘的工作方式 …… 242
四、实验演练 …… 243
第三节　LCD显示器接口 …… 245
一、字符型液晶显示器概述 …… 245
二、字符显示模块内部结构 …… 246
三、字符型液晶控制器的指令 …… 249
四、字符显示实验演练 …… 252
第四节　I^2C总线接口 …… 259

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

一、I^2C 总线及其软件包 …… 259
二、I^2C 总线串行存储器 AT24Cxx …… 262
三、实验演练 …… 270
第五节　A/D 和 D/A 转换接口 …… 271
一、D/A(数模)转换接口 …… 271
二、A/D(模数)转换接口 …… 274
第六节　单片机应用技术综合实例 …… 277
一、999 计数器 …… 277
二、计数显示器 …… 278
三、单片机时钟 …… 280
附录一　运算符的优先级和结合性 …… 293
附录二　Keil Cx51 编译器常见警告与错误信息的解决方法 …… 294
参考文献 …… 296

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第一章　单片机基础知识

随着电子技术的迅速发展，单片机已深入渗透到我们的生活中，许多电子爱好者开始学习单片机知识。本章主要以 Intel 公司生产的 51 单片机为例，介绍单片机的特点、内部结构、外部引脚、存储器配置、并行 I/O 端口、外围时钟和复位电路等内容。

第一节　51 系列单片机概述

一、什么是单片机

单片机因将其主要组成部分集成在一个芯片上而得名。具体说就是把中央处理器(CPU)、随机存储器(RAM)、只读存储器(ROM)、中断系统、定时/计数器以及输入/输出口(I/O)电路等主要微型机部件，集成在一块芯片上。虽然单片机只是一个芯片，但从组成和功能上看，它已具有了计算机系统的属性，为此称它为单片微型计算机(SCMC，Single Chip Microcomputer)，简称单片机。

单片机主要应用于控制领域，用以实现各种测试和控制功能。为了强调其控制属性，也可以把单片机称为微控制器(MCU，Micro Controller Unit)。在国际上，“微控制器”的叫法似乎更通用一些，而在我国则比较习惯于“单片机”这一名称，因此本书使用“单片机”一词。

由于单片机在应用时通常是处于被控系统的核心地位并融入其中，即以嵌入的方式进行使用，为了强调其“嵌入”的特点，也常常将单片机称为嵌入式微控制器(EMCU，Embedded Micro Controller Unit)。在单片机的电路和结构中有许多嵌入式应用的特点。

二、51 单片机的发展

虽然目前单片机的品种很多，但其中最具代表性的当属 Intel 公司的 MCS-51(简称 51)单片机系列。51 单片机以其典型的结构、完善的总线专用寄存器集中管理、众多的逻辑位操作功能及面向控制的丰富的指令系统，堪称为一代“名机”，为以后的其他单片机的发展奠定了基础。正因为其优越的性能和完善的结构，导致后来的许多厂商多沿用或参考了其体系结构，有许多世界大的电气商丰富和发展了 51 单片机，像 PHILIPS、Dallas、ATMEL 等著名的半导体公司都推出了兼容 MCS-51 的单片机产品，就连我国台湾的 WINBOND 公司也发展了兼容 51 单片机品种。

近年来 51 单片机获得了飞速的发展，51 的发源公司 Intel 由于忙于开发 PC 及高端微处理器而无精力继续发展自己的单片机，由其他厂商将其发展，最典型的是 PHILIPS 和 ATMEL 公司。

PHILIPS 公司主要是改善其性能，在原来的基础上发展了高速 I/O 口、A/D 转换

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

器、PWM(脉宽调制)、WDT(看门狗)等增强功能,并对低电压、微功耗、扩展串行总线(I^2C)和控制网络总线(CAN)等功能加以完善。

ATMEL 公司推出的 AT89Cxx 系列兼容 51 的单片机,完美地将 Flash ROM 与 80C51 内核结合起来,仍采用 51 的总体结构和指令系统,Flash ROM 的可反擦写程序存储器能有效地降低开发费用,并能使单片机多次重复使用。

SIEMENS 公司推出的 C500 系列单片机在保持与 80C51 兼容的前提下,增强了各项性能,尤其是增强了电磁兼容性能,增加了 CAN 总线接口,特别适用于工业控制、汽车电子、通信和家电领域。

台湾的 WINBOND 公司也开发了一系列兼容 C51 的单片机,其产品通常具备丰富的功能特性,而且质优价廉,在市场也占有一定的份额。

三、51 单片机主要技术指标

8051/80C51 系列又分成 51 和 52 两个子系列,并以芯片型号的最末位数字作为标志。其中 51 子系列是基本型,而 52 子系列则属增强型。8051/80C51 单片机系列芯片的技术指标如表 1-1 所列。

表 1-1　80C51 单片机系列芯片的技术指标

系列	典型芯片	片内 ROM 形式	片内 RAM	并行 I/O 口	定时/计数器	中断源	串行 I/O 口
51 子系列	8031/80C31	无	128B	4×8	2×16	5	1
	8051/80C51	4KB 掩膜 ROM	128B	4×8	2×16	5	1
	8751/87C51	4KBEPROM	128B	4×8	2×16	5	1
	89C51	4KBFlash ROM	128B	4×8	2×16	5	1
	89LV51	4KBFlash ROM	128B	4×8	2×16	5	1
52 子系列	8032/80C32	无	256B	4×8	3×16	6	1
	8052/80C52	8KB 掩膜 ROM	256B	4×8	3×16	6	1
	8752/87C52	8KBEP ROM	256B	4×8	3×16	6	1
	89C52	8KB Flash ROM	256B	4×8	3×16	6	1
	89SLV52	8KB Flash ROM	256B	4×8	3×16	6	1
	89S8252	8KB Flash ROM	256B	4×8	3×16	6	1
1051	89C1051	1KB Flash ROM	64B	15	1×16	3	1
2051	89C2051	2KB Flash ROM	128B	15	2×16	5	1

四、51 单片机主要型号的特点

1. 8031/80C31

8031/80C31 片内不带程序存储器 ROM,使用时用户需外接程序存储器和一片逻辑电路 373,外接的程序存储器多为 EPROM 的 2764 系列。用户若想对写入到 EPROM 中的程序进行修改,必须先用一种特殊的紫外线灯将其照射擦除,之后可再写入。写入到外接程序存储器的程序代码没有保密性。

2. 8051/80C51

8051/80C51 片内有 4KB ROM,无需外接外存储器和 373,更能体现“单片”的简练。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

但是程序无法烧写到其 ROM 中，只有将程序交芯片厂代为烧写，并且是一次性的，今后用户和芯片厂都不能改写其内容。

3. 8751/87C51

8751/87C51 与 8051 基本一样，但 8751 片内有 4KB 的 EPROM，用户可以将自己编写的程序写入单片机的 EPROM 中进行现场实验与应用，EPROM 的改写同样需要用紫外线灯照射一定时间擦除后再烧写。

由于上述类型的单片机应用得早，影响很大，已成为事实上的工业标准。后来很多芯片厂商以各种方式与 Intel 公司合作，也推出了同类型的单片机，如同一种单片机的多个版本一样，虽都在不断地改变制造工艺，但内核却一样。也就是说，这类单片机指令系统完全兼容，绝大多数管脚也兼容；在使用上基本可以直接互换。人们统称这些与 8051 内核相同的单片机为“51 系列单片机”。对于学习者来说，学了其中一种，便会所有的 51 系列。

4. 89 系列

在众多的 51 系列单片机中，ATMEL 公司的 AT89xx 更实用，因它不但和 8051 指令、管脚完全兼容，而且其片内的程序存储器是 Flash 工艺的，这种工艺的存储器用户可以用电的方式瞬间擦除、改写，一般专为 ATMEL AT89xx 做的编程器均带有这些功能。显而易见，这种单片机对开发设备的要求很低，开发时间也大大缩短；写入单片机内的程序还可以进行加密，可很好地保护用户的劳动成果；再者，AT89C51 目前的售价比 8031 低，市场供应也很充足。

89 系列单片机有多种型号，主要有 AT89C5l、AT89LV51、AT89C52、AT89LV52、AT89C1051、AT89C2051 和 AT89S8252 等。其中 AT89LV51 和 AT89LV52 分别是 AT89C51 和 AT89C52 的低电压产品，最低电压可以低至 2.7V。而 AT89C1051 和 AT89C2051 则是低档型低电压产品，它们只有 20 条引脚，最低电压也为 2.7 V。AT89S8252 属高档型，除了 8KB Flash 存储器外，AT89S8252 还含有一个 2KB 的 EEPROM，从而提高了存储容量。

ATMEL 的 89 系列有多种封装，如 AT89C51 有 DIP、PLCC 和 PQFP/TQFP 等封装；2051/1051 有 DIP 和 SOIC 封装等。

第二节　80C51 单片机的内部结构和外部引脚

一、80C51 单片机的内部结构框图

80C51 内部包含中央处理器、程序存储器(ROM)、数据存储器(RAM)、定时/计数器、并行接口、串行接口和中断系统，以及数据总线、地址总线和控制总线等，其内部结构框图如图 1-1 所示。

从框图中可以看出，80C51 单片机虽然只是一个芯片，但作为计算机应该具有的基本部件几乎都包括，因此，80C51 单片机实际上已经是一个简单的微型计算机系统。下面对其主要部件分别加以说明。

1. 中央处理器(CPU)

中央处理器(CPU)是整个单片机的核心部件，是 8 位数据宽度的处理器，能处理 8 位

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

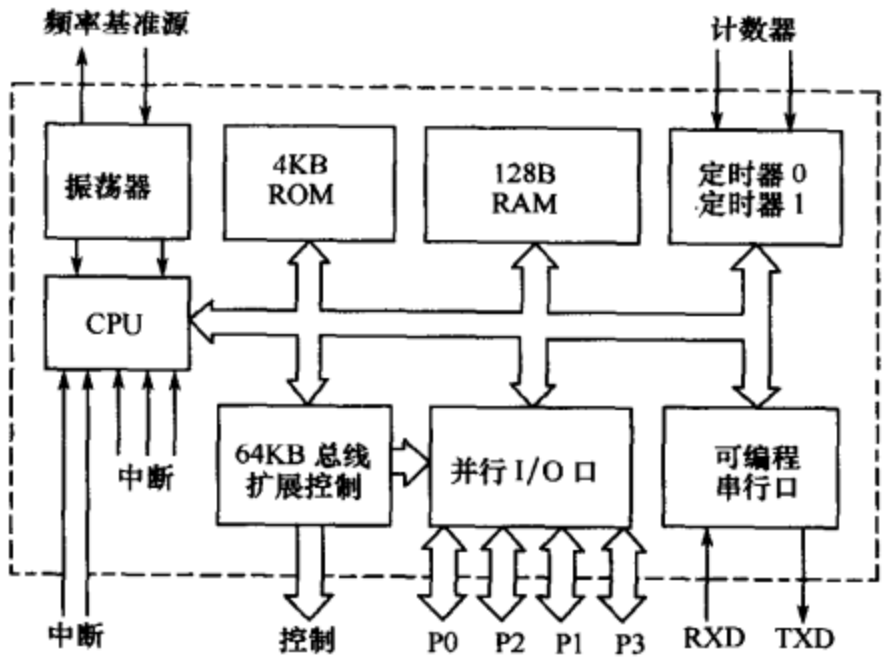


图 1-1　80C51 单片机内部结构框图

二进制数据或代码，CPU 负责控制、指挥和调度整个单元系统协调的工作，完成运算和控制输入输出功能等操作。中央处理器主要由运算器和控制器两部分组成。

(1)运算器

运算器是单片机的运算部件，用于实现算术和逻辑运算。运算器主要由算术/逻辑运算部件 ALU、暂存器 TMP、累加器 ACC、寄存器 B、程序状态标志寄存器 PSW 及布尔处理器(位处理器)等组成。累加器 ACC 是一个 8 位寄存器，它是 CPU 中工作最频繁的寄存器。在进行算术、逻辑运算时，累加器 ACC 往往在运算前暂存一个操作数(如被加数)，而运算后又保存其结果(如代数和)。寄存器 B 主要用于乘法和除法操作。标志寄存器 PSW 也是一个 8 位寄存器，用来存放运算结果的一些特征，如有无进位、借位等。

(2)控制器

控制器是单片机的指挥控制部件，保证单片机各部分能自动而协调地工作。控制器主要包括定时控制逻辑电路、指令寄存器、译码器、地址指针 DPTR 及程序计数器 PC、堆栈指针 SP 等。

这里的程序计数器 PC 是由 16 位寄存器构成的计数器。要单片机执行一个程序，就必须把该程序按顺序预先装入存储器 ROM 的某个区域。单片机动作时应按顺序一条条取出指令来加以执行。因此，必须有一个电路能找出指令所在的单元地址，该电路就是程序计数器 PC。当单片机开始执行程序时，给 PC 装入第一条指令所在地址，它每取出一条指令(如为多字节指令，则每取出一个指令字节)，PC 的内容就自动加 1，以指向下一条指令的地址，使指令能顺序执行。只有当程序遇到转移指令、子程序调用指令，或遇到中断时，PC 才转到所需要的地方去。

80C51 CPU 按 PC 指定的地址，从 ROM 相应单元中取出指令字节放在指令寄存器中寄存，然后，指令寄存器中的指令代码被译码器译成各种形式的控制信号，这些信号与单片机时钟振荡器产生的时钟脉冲在定时与控制电路中相结合，形成按一定时间节拍变化的电平和时钟，即所谓控制信息，在 CPU 内部协调寄存器之间的数据传输、运算等操作。

2. 存储器

存储器又分为只读存储器(ROM)和数据存储器(RAM)两种，前者存放调试好的固

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

定程序和常数，后者存放一些随时有可能变动的数据。

80C51 共有 4096 个 8 位(4KB)掩膜 ROM，有 128 个 8 位用户数据存储单元和 128 个专用寄存器单元，专用寄存器只能用于存放控制指令数据，用户只能访问，而不能存放用户数据，所以，用户能使用的 RAM 只有 128 个。

3. 定时/计数器

单片机除了进行运算外，还要完成控制功能，所以离不开计数和定时。因此，在单片机中设置有定时器兼计数器。80C51 有两个 16 位的可编程定时/计数器，以实现定时或计数。

4. 并行输入/输出(I/O)口

80C51 共有 4 组 8 位 I/O 口(P0、P2、P1 和 P3)，用于和外部数据进行并行传输。

5. 全双工串行口

80C51 内置一个全双工串行通信口，用于与其他设备间的串行数据传送，该串行口既可以用作异步通信收发器，也可以当同步移位器使用。

6. 中断系统

中断系统相当于“传达室”，当单片机控制对象的参数到达某个需要加以干预的状态时，就可经此“传达室”通报给 CPU，使 CPU 根据外部事态的轻重缓急来采取适当的应付措施。

80C51 具备较完善的中断功能，有两个外中断、两个定时/计数器中断和一个串行中断，可满足不同的控制要求，并具有 2 级的优先级别选择。

7. 时钟电路

单片机里还有一个时钟电路，使单片机在进行运算和控制时，能有节奏地进行。80C51 内置最高频率达 12MHz 的时钟电路，用于产生整个单片机运行的脉冲时序，但 80C51 单片机需外置振荡电容。

综上所述，实际上单片机内部有一条将它们连接起来的“纽带”，即所谓的“内部总线”。而 CPU、ROM、RAM、I/O 口、中断系统等就分布在此“总线”的两旁，并和它连通，从而使一切指令、数据都可经内部总线传送。

二、单片机的外部引脚

80C51 单片机采用 40PIN(脚)封装的双列直接 DIP 结构，其引脚配置如图 1-2 所示。

40 个引脚中，正电源和地线 2 个引脚，外置石英振荡器的时钟线 2 个引脚，4 组 8 位共32 个I/O 口(中断口线与 P3 口线复用)；控制引脚 4 个。下面对这些引脚的功能加以说明。

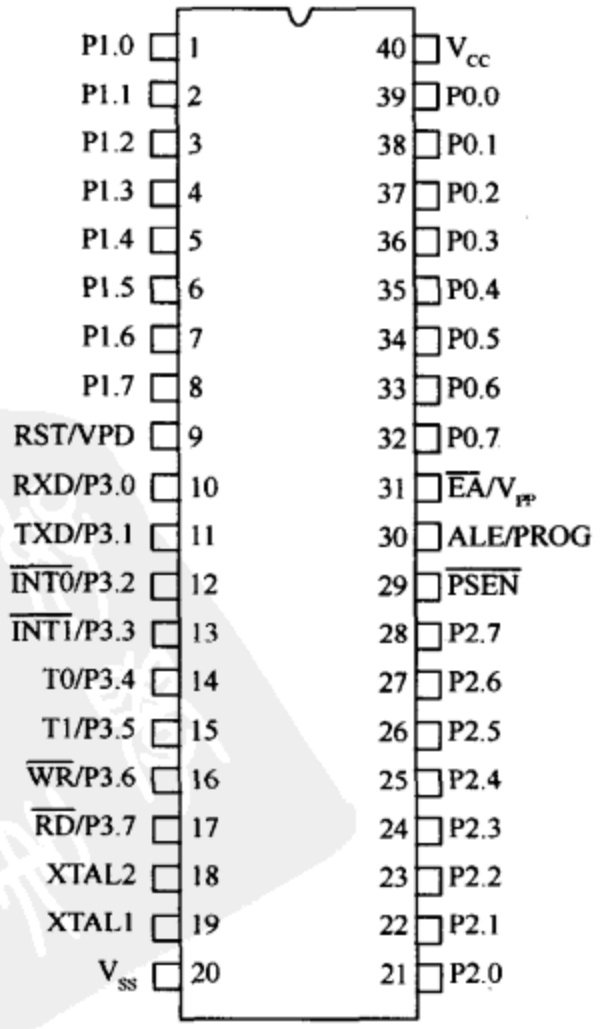


图 1-2　80C51 外部引脚配置图

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

1. 电源和接地引脚(2 个)

V_{SS}(PIN20):接地脚。

V_{CC}(PIN40):正电源脚,正常工作或对片内 EPROM 烧写程序时,接+5V 电源。

2. 外接晶体引脚(2 个)

XTAL1(PIN19):时钟 XTAL1 脚,片内振荡电路的输入端。

XTAL2(PIN18):时钟 XTAL2 脚,片内振荡电路的输出端。

重点提示 8051 单片机的时钟有两种方式:一种是片内时钟振荡方式,但需在 18 和 19 脚外接石英晶体(2MHz~12MHz)和振荡电容,振荡电容的值一般取 30pF;另外一种是外部时钟方式,即将 XTAL1 接地,外部时钟信号从 XTAL2 脚输入。对于 80C51 单片机,情况有些不同。外引脉冲信号需从 XTAL1 引脚注入,而 XTAL2 引脚悬空。

3. 输入/输出引脚(32 个)

P0.0~P0.7(PIN39~PIN32):P0 口是一个 8 位漏极开路的双向 I/O 口,它是一个多功能口。在访问外部存储器时,用作分时多路转换地址(低 8 位)和数据总线,P0 口常用作此方式。在没有外部存储器时,P0 口可作为并行 I/O 口使用,但须外接上拉电阻。它的带负载能力为 8 个 LSTTL 门电路。

P1.0~P1.7(PIN1~PIN8):P1 口是一个带有内部上拉电阻的 8 位准双向 I/O 口。它通常用作通用 I/O 口,能带动 4 个 LSTTL 门电路。

P2.0~P2.7(PIN21~PIN28):P2 口是一个带有内部上拉电阻的 8 位准双向 I/O 口,它是一个多功能口。在访问外部存储器时,它送出地址的高 8 位;在没有外部存储器时,可作为通用 I/O 口使用。可带动 4 个 LSTTL 门电路。

P3.0~P3.7(PIN10~PIN17):P3 口是一个带有内部上拉电阻的 8 位准双向 I/O 口,它是一个多功能口。P3 口的第一功能是作为通用 I/O 口,第二功能如表 1-2 所列。

表 1-2 P3 口的第二功能

引脚	第二功能	引脚	第二功能
P3.0	串行数据输入(RXD)	P3.4	定时/计数器 0 外部输入(T0)
P3.1	串行数据输出(TXD)	P3.5	定时/计数器 1 外部输入(T1)
P3.2	外部中断 0 输入(INT0)	P3.6	外部 RAM 写选通信号($\overline{WR}$)
P3.3	外部中断 1 输入(INT1)	P3.7	外部 RAM 读选通信号($\overline{WR}$)

4. 控制引脚

(1)RST/VPD(PIN9)

复位信号引脚。当振荡器运行时,在此引脚上出现两个机器周期以上的高电平将使单片机复位。一般在此引脚与 V_{SS}之间连接一个下拉电阻,与 V_{CC}引脚之间连接一个电容。此外,RST/VPD 还是一个复用脚,V_{CC}掉电期间,此脚可接上备用电源,当电源电压下降到下限值时,备用电源经此端向内部 RAM 提供电压,以保证内部 RAM 中的信息不丢失。

(2)ALE/PROG(PIN30)

地址锁存允许信号。当访问外部存储器时,ALE(允许地址锁存)的输出用于锁存地

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

址的低 8 位。即使不访问外部存储器，ALE 端仍以不变的频率周期性地输出脉冲信号，此频率为石英晶振振荡频率的 1/6。因此，它可用作对外输出的时钟，或用于定时的目的。另外，ALE/PROG 还是一个复用脚，在编程期间，PROG 将用于输入编程脉冲。

(3)$\overline{\text{PSEN}}$(PIN29)

外部程序存储器的读选通信号。在读外部 ROM 时，$\overline{\text{PSEN}}$有效(低电平)，以实现对外部程序存储器的读操作。

(4)$\overline{\text{EA}}$/Vpp(PIN31)

程序存储器的内外部选通线。当$\overline{\text{EA}}$信号接低电平时，对 ROM 的读操作(执行程序)限定在外部程序存储器；当$\overline{\text{EA}}$接高电平时，对 ROM 的读操作(执行程序)从内部开始。对于 80C51 单片机，内置有 4KB 的程序存储器，当$\overline{\text{EA}}$为高电平并且程序地址小于 4KB 时，读取内部程序存储器指令数据，而超过 4kB 地址则读取外部指令数据。如$\overline{\text{EA}}$为低电平，则不管地址大小，一律读取外部程序存储器指令。显然，对内部无程序存储器的 8031，$\overline{\text{EA}}$端必须接地。另外，$\overline{\text{AE}}$/V_{PP}还是一个复用脚，在编程时，V_{PP}脚需加上 21V 的编程电压。

重点提示 由于工艺及标准化等方面的原因，芯片的引脚数目是有限的，例如，80C51 把引脚数目限定为 40 条，但单片机为实现其功能，所需要的信号数目都远远超过此数，因此就出现了需要与可能的矛盾。为了解决这个矛盾，给一些信号赋予双重功能即“兼职”。如果把前述信号定义为第一功能的话，则根据需要再定义的信号就是第二功能。像 30 脚(ALE/PROG)，31 脚($\overline{\text{EA}}$/VPD)和 9 脚(RST/V_{PP})都具有第二功能。

一个信号引脚，又是第一功能又是第二功能，会不会在使用时引起混乱造成错误呢？不会的。对于 9、30 和 31 各引脚，由于第一功能信号与第二功能信号是单片机在不同工作方式下的信号，因此不会发生使用上的矛盾。对于 P3 口线，在实际使用时，总是先按需要优先选用它的第二功能，剩下不用的才作为口线使用。

第三节 80C51 单片机内部存储器的配置

存储器是单片机的一个重要组成部分，图 1-3 给出了 80C51 存储器的配置图，从图中可以看出，80C51 的存储器主要有 4 个物理存储空间，即片内程序存储器、片外程序存储器、片内数据存储器和片外数据存储器。

一、程序存储器 ROM

对于普通 8051 单片机，程序存储器 ROM 空间大小为 64KB，用于存放程序代码和一些表格常数，称为 CODE 空间。普通 8051 可采用“代码分组”(CODE BANK)设计技术，将 ROM 空间扩展到 32×64KB，新型 Philips80C51Mx 单片机的 ROM 空间最大可扩展到 16MB，称为 ECODE 和 HCONST 空间。8051 单片机专门提供一个引脚“$\overline{\text{EA}}$”来区分片内 ROM 和片外 ROM，$\overline{\text{EA}}$引脚接高电平时，单片机从片内 ROM 中读取指令，当指令地址超过片内 ROM 空间范围后，就自动地转向片外 ROM 读取指令；$\overline{\text{EA}}$引脚接低电平时，所有的取指操作均对片外 ROM 进行。

注意事项 程序存储器的某些地址单元是保留给系统使用的。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

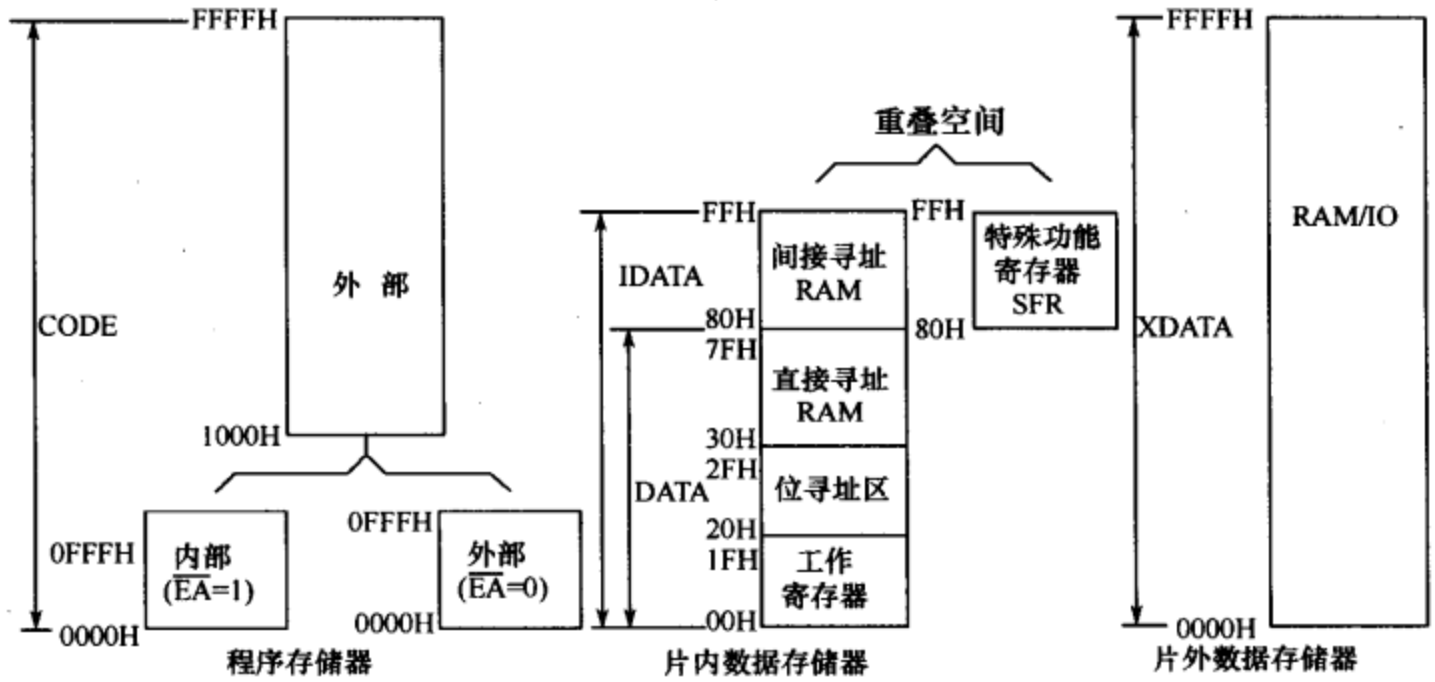


图 1-3　80C51 存储器配置图

一组是 0000H～0002H 单元，系统复位后，PC 值为 0000H，单片机从 0000H 单元开始执行程序，如果程序不是从 0000H 单元开始，则应在这三个单元中存放一条无条件转移指令，让 CPU 直接去执行用户指定的程序。

另一组特殊单元是 0003H～002AH，这 40 个单元各有用途，它们被均匀地分为 5 段，各段的定义如下：

0003H～000AH：外部中断 0 中断地址区；

000BH～0012H：定时器/计数器 0 中断地址区；

0013H～001AH：外部中断 1 中断地址区；

001BH～0022H：定时器/计数器 1 中断地址区；

0023H～002AH：串行中断地址区。

一些新型 8051 单片机增加了更多的中断源，它们的中断入口地址也相应增加。

二、片内数据存储器 RAM

对于普通 8051 单片机，片内数据存储器 RAM 空间最大为 256B，用于存放程序执行过程的各种变量及临时数据。片内 RAM 的低 128 个字节可用直接寻址方式进行访问，也可用间接寻址方式访问，称为 DATA 区。其中，00H～1FH 地址范围平均分为 4 组，每组都有 8 个工作寄存器 R0～R7，称为工作寄存器区(Register Banks)。20H～2FH 地址范围中，每个存储器单元的每一位都可以用位处理指令直接操作，该段地址范围称为位寻址区(BDATA 区)，其中每一位称为一个 bit。30H～7FH 地址范围中，是可以给用户使用的 RAM 单元，称为直接寻址 RAM。80H～FFH 地址范围中，是专门用于特殊功能寄存器(AFR)的区域，在特殊功能寄存器区中，有 22 个特殊功能寄存器，它们不连续地分散在内部 RAM 的高 128 单元之中，尽管其中还有许多空闲地址，但用户不能使用。22 个特殊功能寄存器中，有 21 个是可寻址的，其可寻址位称为 sbit。这些可寻址的寄存器的名称、符号及地址如表 1-3 所列。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

表 1-3　80C51 特殊功能寄存器一览表

序号	寄存器符号	寄存器地址	寄存器名称	说明
1	ACC	0E0H	累加器	可位寻址
2	B	0F0H	B 寄存器	可位寻址
3	PSW	0D0H	程序状态字	可位寻址
4	SP	81H	堆栈指示器	
5	DPL	82H	数据指针低 8 位	
6	DPH	83H	数据指针高 8 位	
7	IE	0A8H	中断允许控制寄存器	可位寻址
8	IP	0B8H	中断优先控制寄存器	可位寻址
9	P0	80H	输入/输出口 0	可位寻址
10	P1	90H	输入/输出口 1	可位寻址
11	P2	0A0H	输入/输出口 2	可位寻址
12	P3	0B0H	输入/输出口 3	可位寻址
13	PCON	87H	电源控制及波特率选择寄存器	
14	SCON	98H	串行口控制寄存器	可位寻址
15	SBUF	99H	串行数据缓冲寄存器	
16	TCON	88H	定时器控制寄存器	可位寻址
17	TMOD	89H	定时器方式选择寄存器	
18	TL0	8AH	定时器 0 低 8 位	
19	TL1	8BH	定时器 1 低 8 位	
20	TH0	8CH	定时器 0 高 8 位	
21	TH1	8DH	定时器 1 高 8 位	

从表 1-3 中可以看出，80C51 单片机中，有 11 个寄存器不仅可以字节寻址，也可以进行位寻址。凡是能进行位寻址的 SFR，其特征是字节地址都能被 8 整除（字节地址的末位是 0 或 8）。80C51 单片机的位地址表如表 1-4 所列。

表 1-4　80C51 单片机的位地址表

寄存器符号	MSB	位地址/位名称					LSB
B	0F7H	0F6H	0F5H	0F4H	0F3H	0F2H	0F1H	0F0H
A	0E7H	0E6H	0E5H	0E4H	0E3H	0E2H	0E1H	0E0H
PSW	0D7H	0D6H	0D5H	0D4H	0D3H	0D2H	0D1H	0D0H
	CY	AC	F0	RS1	RS0	OV	—	P
IP	0BFH	0BEH	0BDH	0BCH	0BBH	0BAH	0B9H	0B8H
	—	—	—	PS	PT1	PX1	PT0	PX0
IE	0AFH	0AEH	0ADH	0ACH	0ABH	0AAH	0A9H	0A8H
	EA	—	—	ES	ET1	EX1	ET0	EX0
SCON	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

（续）

寄存器符号	MSB			位地址/位名称				LSB
TCON	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
P0	87H	86H	85H	84H	83H	82H	81H	80H
	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1	97H	96H	95H	94H	93H	92H	91H	90H
	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2	0A7H	0A6H	0A5H	0A4H	0A3H	0A2H	0A1H	0A0H
	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3	0B7H	0B6H	0B5H	0B4H	0B3H	0B2H	0B1H	0B0H
	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

从表 1-4 中可以看出，IP 中有 3 位、IE 中有 2 位、PSW 中有 1 位对用户无实际意义，所以直接寻址位为 82 位；再加上数据存储器中的 128 位，80C51 共计有 210 位可寻址位。

重点提示 对于 51 子系列单片机，整个片内 RAM 地址范围 00H～FFH 称为 IDATA 区。另外需要说明的是，PHILIPS 公司推出的新型单片机 80C51Mx，其片内 RAM 最大可扩充到 64KB，称为 EDATA 区。

三、片外数据存储器 RAM

对于普通 8051 单片机，片外数据存储器 RAM 空间大小为 64KB，称为 XDATA 区。在 XDATA 空间内进行分页寻址操作时，称为 PDATA 区。有些新型 80C51 单片机的扩充片内 RAM 需要用专门的特殊功能寄存器“映像”(MAP)到 XDATA 地址空间，还有一些新型 80C51 单片机可以将片外 RAM 最大扩展到 16MB，称为 HDATD 区。

第四节　51 系列单片机的并行输入/输出接口

80C51 有 4 组 8 位 I/O 口：P0、P1、P2 和 P3 口，P1、P2 和 P3 为准双向口，P0 口则为双向三态输入输出口，这 4 个口除了按字节寻址之外，还可以按位寻址。4 个口在结构和功能上各具特点，下面分别介绍这几个口线。

一、P0 口

P0 口的字节地址为 80H，位地址为 80H～87H。口的各位口线具有完全相同但又相互独立的逻辑电路，如图 1-4 所示。

1. P0 口的组成

P0 口逻辑电路主要由以下几部分组成：

(1)2 个三态输入缓冲器

2 个三态输入缓冲器分别用于锁存器数据和引脚数据的输入缓冲。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

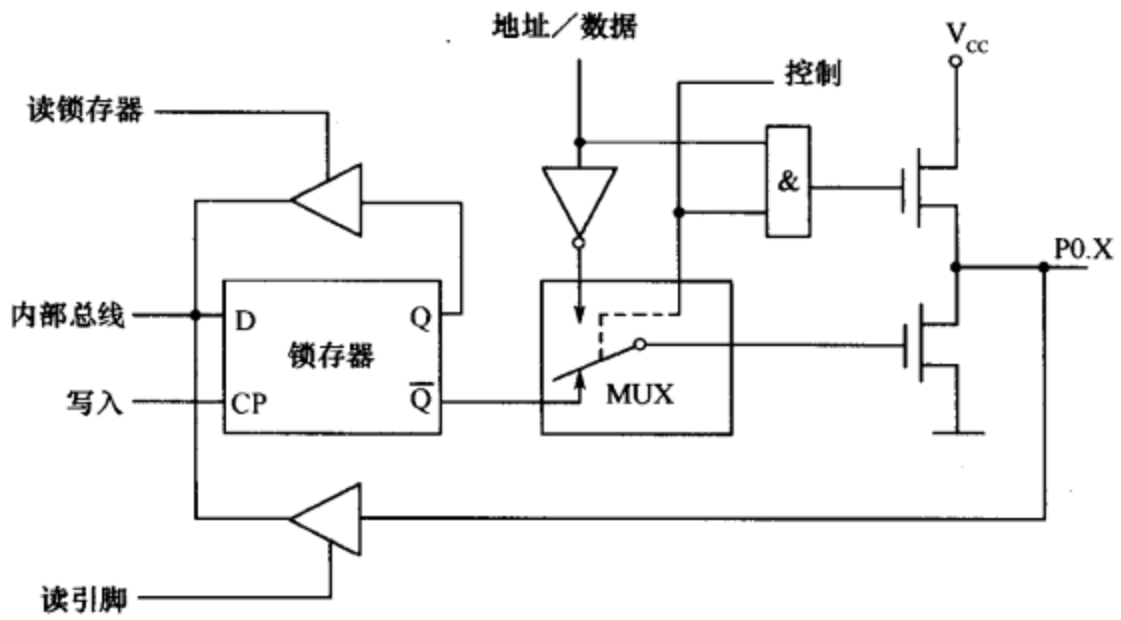


图 1-4　P0 口逻辑电路

(2)1 个多路转接开关 MUX

多路转接开关 MUX 的一个输入来自锁存器，另一个输入为“地址/数据”。输入转接由“控制”信号控制。之所以设置多路转接开关，是因为 P0 口既可以作为通用的 I/O 口进行数据的输入输出，又可以作为单片机系统的地址/数据线使用。即在控制信号的作用下，由 MUX 实现锁存器输出和地址/数据线之间的接通转接。

(3)1 个数据输出锁存器

数据输出锁存器用于进行数据位的锁存。在不同时刻，不同的部件需要不同的信号。比如某一时刻 P0.0 要求输出高电平并要求保持若干时间，在这段时间里，CPU 不能停在那里，它还需要与其他部件联络，因此这根数据线上的电平未必能保持原来的值不变，这样输出将会发生变化。为解决这一问题，在每一个输出端加一个锁存器。若要某个 I/O 口输出数据，只要将待输出的数据写入相应的 I/O 口(实际是写入相应的锁存器)即可；然后 CPU 就可以去做其他事情，不必再考虑输出的状态了。锁存器会把数据“锁”住，直到 CPU 下一次改写数据为止。

(4)数据输出的驱动电路

数据输出的驱动电路由两只场效应管(FET)组成，上面的场效应管构成上拉电路。

2. P0 口的使用

在实际应用中，P0 口绝大多数情况下都是作为单片机系统的地址/数据线使用。当传送地址或数据时，控制信号为高电平“1”，转换开关把反相器输出端与下拉 FET 接通，同时与门开锁。输出的地址或数据信号既通过与门去驱动上拉 FET，又通过反相器去驱动下拉 FET。这时的输出驱动电路由于上下两个 FET 处于反相，形成推拉式电路结构，大大地提高了负载能力。而当输入数据时，数据信号则直接从引脚通过输入缓冲器进入内部总线。

P0 口也可作为通用 I/O 口使用，这时，CPU 发来的控制信号为低电平“0”，封锁了与门，并将输出驱动电路的上拉场效应管截止，而多路转接开关 MUX 接通锁存器 Q 端的输出通路。下面简要介绍输出和输入操作的过程。

(1)输出操作(写操作)

当 P0 口作为输出口(写)使用时，由锁存器和驱动电路构成数据输出通路。由于通

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

路中已有输出锁存器，因此数据输出可以与外设直接连接，无需再加数据锁存电路。进行数据输出时，来自 CPU 的写脉冲加在 D 触发器的 CP 端，数据写入锁存器，并向端口引脚输出。但要注意，由于输出电路是漏极开路电路，必须外接上拉电阻才能有高电平输出。

(2)输入操作(读操作)

当 P0 口作为输入口使用时，应区分读引脚和读锁存器(端口)两种情况。为此，在电路中有两个用于读入的三态缓冲器。

读引脚。所谓读引脚就是读芯片引脚上的数据，也就是直接读取外部数据。这时使用锁存器下方的缓冲器，由“读引脚”信号把缓冲器打开，引脚上的数据经缓冲器通过内部总线读进来。需要说明的是，在 P0 口作为输入口读引脚使用时，应先向锁存器写“1”(一般用传送指令)，使输出级的 2 个 FET 截止(系统复位时 P0＝FFH)。

读锁存器。在端口已处于输出状态的情况下读锁存器。读锁存器是通过上方的缓冲器读锁存器 Q 端的状态。在端口已处于输出状态的情况下，不能正常读取引脚的信号，只能读取锁存器的状态。这样安排的目的是适应对端口进行“读—修改—写”操作指令的需要。例如“ANL P0,A”就属于这类指令，执行时先读入 P0 口锁存器中的数据，然后与 A 的内容进行逻辑“与”，再把结果送到 P0 口输出。从这种意义上说，该指令又可看作是输出指令。除 MOV 类指令外的其他口操作指令都属于这种情况。

注意事项 P0 口既可作为单片机系统的地址/数据线使用，也可作为通用 I/O 口使用。当作为通用 I/O 口输出时，由于输出电路是漏极开路电路，必须外接上拉电阻才能有高电平输出。作输入使用时，应区分读引脚和读锁存器。读引脚时，必须先向电路中的锁存器写入“1”，使 FET 截止，引脚处于悬浮状态，可作为高阻抗输入。因为，若 FET 导通，引脚上的电位始终被钳在“0”电平上，输入数据不可能被正确地读入。

二、P1 口

P1 口的字节地址为 90H，位地址为 90H～97H。P1 口逻辑电路如图 1-5 所示。

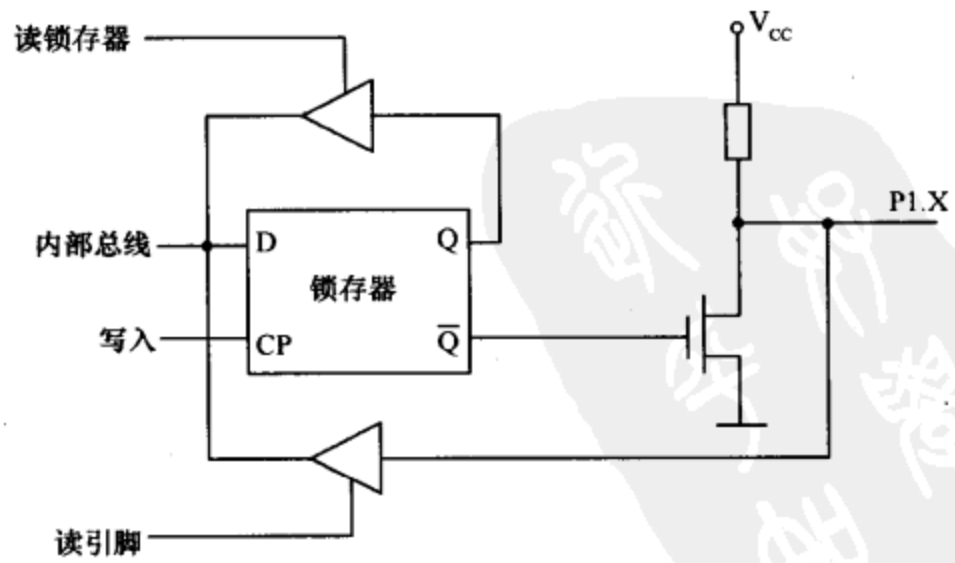


图 1-5　P1 口逻辑电路

P1 口只能作为通用 I/O 口使用，所以在电路结构上与 P0 口有些不同。首先，因为它只传送数据，所以不再需要多路转接开关 MUX；其次，由于只用来传送数据，因此输出电路中有上拉电阻，上拉电阻与场效应管共同组成输出驱动电路。因为这样电路的输出不

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

是三态的，所以 Pl 口是准双向口。

注意事项 P1 口只能作为通用 I/O 口使用，当 P1 口作为输出口使用时，已能对外提供推拉电流负载，外电路无需再接上拉电阻。当 P1 口作为输入口使用时，应区分读引脚和读锁存器，读引脚时，应先向其锁存器写入“1”，使输出驱动电路的 FET 截止。

三、P2 口

P2 口的字节地址为 0A0H，位地址为 0A0H～0A7H。P2 口的逻辑电路如图 1-6 所示。

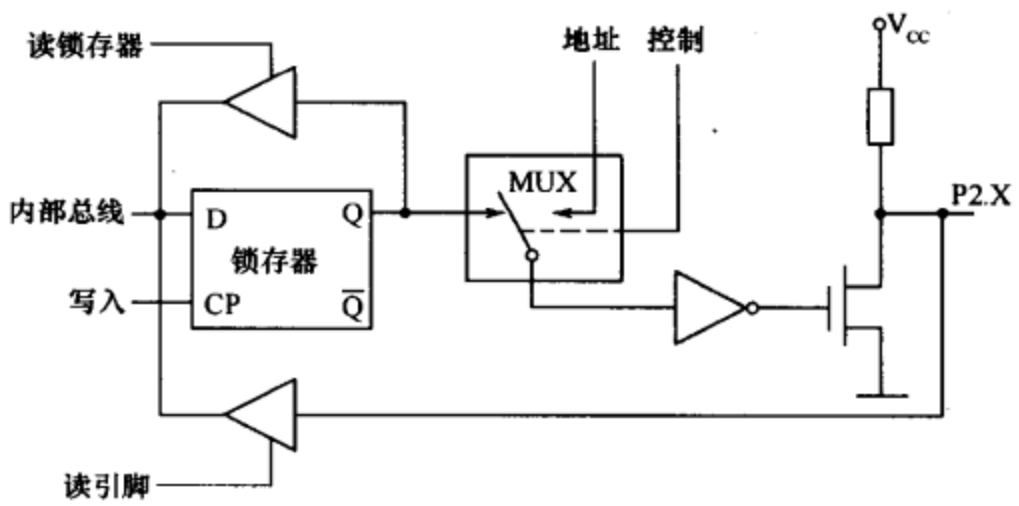


图 1-6 P2 口逻辑电路

因为在实际应用中 P2 口用于为系统提供高位地址，因此同 P0 口一样，在口电路中有一个多路转接开关 MUX。但 MUX 的一个输入端不再是“地址/数据”，而是单一的“地址”，这是因为 P2 口只作为地址线使用而不作为数据线使用。当 P2 口作为高位地址线使用时，多路转接开关应倒向“地址”端。正因为只作为地址线使用，口的输出用不着是三态的，所以 P2 口也是一个准双向口。

注意事项 P2 口既可作为地址总线(高位地址)，也可以作为通用 I/O 口使用，当作为通用 I/O 口使用时，多路转接开关 MUX 倒向锁存器 Q 端。其使用方法与 P1 口类似，作输出口使用时无需接上拉电阻，作输入口使用时，应区分读引脚和读锁存器，读引脚时应先向锁存器写“1”。

四、P3 口

P3 口的字节地址为 0B0H，位地址为 0B0H～0B7H。P3 口逻辑电路如图 1-7 所示。

P3 口不但可作为通用 I/O 口使用，而且还具有第二功能，因此，P3 口在内电路中增加了第二功能控制逻辑。由于第二功能信号有输入和输出两类，因此分两种情况说明。

对于输出的第二功能信号引脚，当作为通用 I/O 口使用时，电路中的“第二输出功能”信号线应保持高电平，与非门开通，以维持从锁存器到输出端数据输出通路的畅通。当输出第二功能信号时，该锁存器应预先置“1”，使与非门对第二功能信号的输出是畅通的，从而实现第二功能信号的输出。

对于第二功能为输入信号的引脚，在口线的输入通路上增加了一个缓冲器，输入的信号就从这个缓冲器的输出端取得。而作为通用 I/O 口线使用的数据输入，仍取自三态缓

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

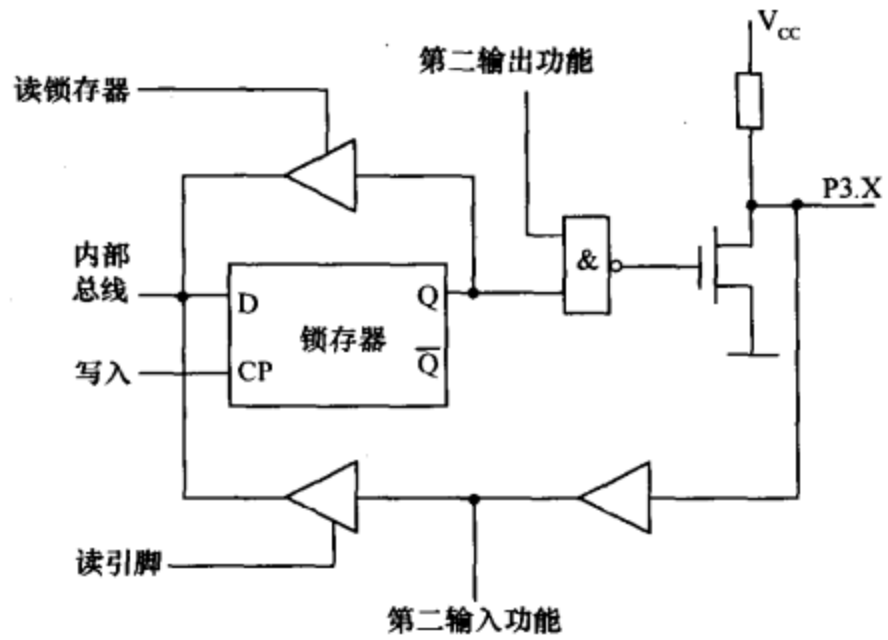


图 1-7　P3 口逻辑电路

冲器的输出端。

注意事项　P3 口既可作为第二功能使用，又可作为通用 I/O 口使用，当某些口作为第二功能使用时，不能再把他当作通用 I/O 口使用，但其他未使用的口线可作为通用 I/O 口线。

P3 口作为通用 I/O 口使用时，其使用方法与 P1 口类似，作输出口使用时无需接上拉电阻，作输入口使用时，应区分读引脚和读锁存器，读引脚时应先向锁存器写“1”。

归纳总结　① P0、Pl、P2、P3 都是并行 I/O 口，都可用于数据的输入/输出传送，但 P0 口和 P2 口除了可进行数据的输入/输出外，通常是用来构建系统的数据总线和地址总线，所以，在口电路逻辑中有一个多路转接开关 MUX，以便进行两种用途的转换。而 P1 和 P3 口没有构建数据和地址总线的功能，因此在电路中没有多路转接开关 MUX。② 在 4 个口中，只有 P0 口是真正的双向口，而其余的 3 个口都是准双向口。原因是在应用系统中，P0 口作为系统的数据总线使用时，为了保证正确的数据传送，需要解决芯片内外的隔离问题，即只有在数据传送时芯片内外才接通；不进行数据传送时，芯片内外应处于隔离状态。为此就要求 P0 口的输出缓冲器是一个三态门。在 P0 中，输出三态门是由两个场效应管(FET)组成的，所以说它是一个真正的双向口。而其他 3 个口中，上拉电阻代替了 P0 口中的场效应管，输出缓冲器不是三态的，因此不是真正的双向口，而只称其为准双向口。③ P3 口的口线具有第二功能，为系统提供一些控制信号。因此在 P3 口电路中增加了第二功能控制逻辑。这是 P3 口与其他各口不同之处。

第五节　80C51 单片机的时钟电路和复位电路

一、单片机的时钟电路

时钟电路用于产生单片机工作所需要的时钟信号，单片机本身就是一个复杂的同步时序电路，为了保证同步工作方式的实现，电路应在唯一的时钟信号控制下严格地按时序进行工作。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

在 80C51 芯片内部有一个高增益反相放大器，其输入端为芯片引脚 XTAL1，输出端为引脚 XTAL2，在芯片的外部通过这两个引脚跨接晶体振荡器和微调电容，形成反馈电路，就构成了一个稳定的自激振荡器，如图 1-8 所示。

电路中对电容 C1 和 C2 的要求不是很严格，如使用高质的晶振，则不管频率多少，C1、C2 一般都选择 30 pF。晶体的振荡频率范围通常是 1.2 MHz～12 MHz，晶体振荡频率高，则系统的时钟频率也高，单片机运行速度也就快。振荡电路产生的振荡脉冲并不直接使用，而是经分频后再为系统所用，如图 1-9 所示。

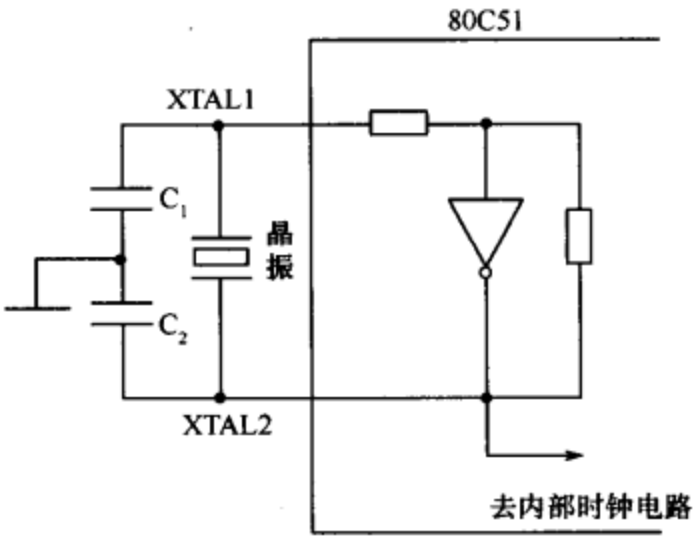


图 1-8　80C51 单片机的振荡电路

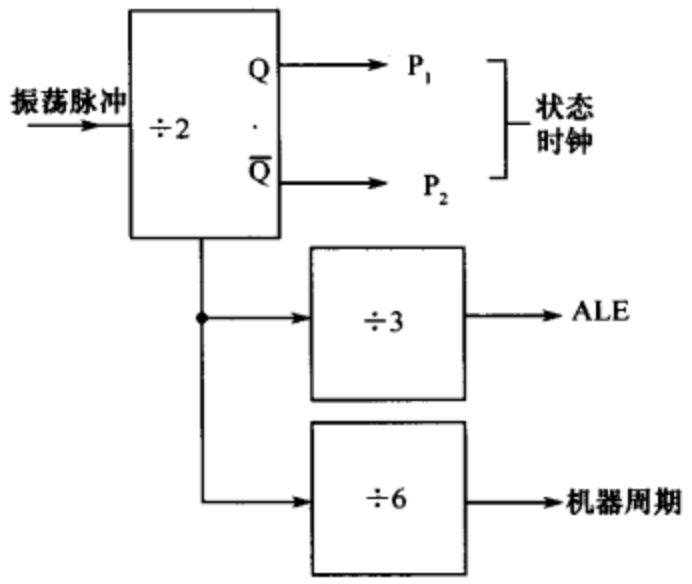


图 1-9　时钟的分频

振荡脉冲经过二分频后作为系统的时钟信号，在二分频的基础上再三分频产生 ALE 信号，即 ALE 为晶振固定频率的 1/6，在二分频的基础上再六分频得到机器周期信号，即机器周期为晶振固定频率的 1/12。

二、单片机的复位电路

复位是单片机的初始化操作，其主要功能是把 PC 初始化为 0000H，使单片机从 0000H 单元开始执行程序。除了进入系统的正常初始化之外，当由于程序运行出错或操作错误使系统处于死锁状态时，也需按复位键以重新启动。

1. 复位对专用寄存器的影响

除 PC 之外，复位操作还对其他一些专用寄存器有影响，如表 1-5 所列。

表 1-5　专用寄存器的复位状态

寄存器	复位状态	寄存器	复位状态
PC	0000H	TCON	00H
ACC	00H	TL0	00H
PSW	00H	TH0	00H
SP	07H	TL1	00H
DPTR	0000H	TH1	00H
P0～P3	FFH	SCON	00H
IP	xxx00000B	SBUF	不定
IE	0xx00000B	PCON	0xxx0000B
TMOD	00H		

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

2. 复位电路

80C51 单片机的 RST 引脚是复位信号的输入端，复位信号是高电平有效，其有效时间应持续 24 个振荡脉冲周期(即 2 个机器周期)以上，若使用频率为 6 MHz 的晶振，则复位信号持续时间应超过 4μs 才能完成复位操作。

复位操作有上电自动复位和按键手动复位两种方式，如图 1-10 所示。

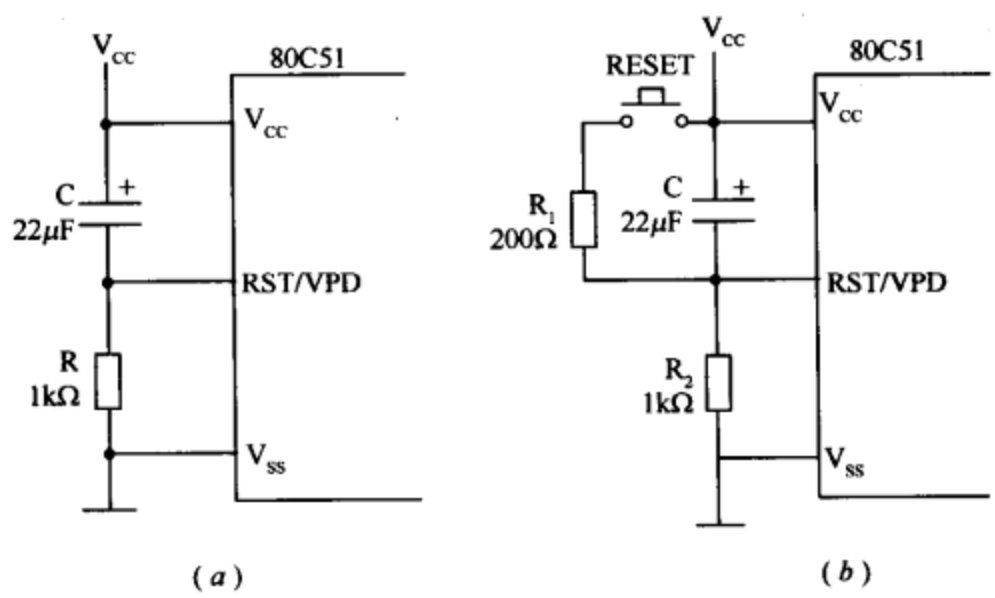


图 1-10　复位电路

(a)上电自动复位；(b)按键手动复位。

(1)上电自动复位

上电自动复位是通过外部复位电路的电容充电来实现的，如图 1-10(a)所示。对于 CMOS 型单片机，由于在 RST 端内部有一个下拉电阻，可将外部电阻去掉。

上电自动复位的过程是在加电时，复位电路通过电容加给 RST 端一个短暂的高电平信号，此高电平信号随着 V_{CC}对电容的充电过程而逐渐回落，即 RST 端的高电平持续时间取决于电容的充电时间。为了保证系统能够可靠地复位，RST 端的高电平信号必须维持足够长的时间。

(2)按键手动复位

按键手动复位需要人为地在复位输入端 RST 上加入高电平。一般采用的办法是在 RST 端和正电源 V_{CC}之间接一个按钮，如图 1-10(b)所示。当按下按钮时，则 V_{CC}的 +5 V电平就会直接加到 RST 端。即使按下按钮的动作较快，也会使按钮保持接通达数十毫秒，所以，可以满足复位的时间要求。

三、单片机的低功耗方式

80C51 有两种低功耗方式，即待机(或称空闲)方式和掉电保护(或称停机)方式，目的是为了尽可能地降低功耗。

待机方式和掉电保护方式都是由特殊功能寄存器 PCON(电源控制寄存器)中的有关位来控制。其各位作用如下：

PCON	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
(87H)	SMOD	—	—	—	GF1	GF0	PD	IDL

SMOD：波特率倍增位，在串行通信时使用。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

GFl 和 GF0：通用标志位，由软件置、复位。

PD：掉电方式位。若 PD＝1，进入掉电工作方式。

IDL：待机方式位。若 IDL＝1，进入待机工作方式。

如果 PD 和 IDL 同时为 1，则进入掉电工作方式。复位时 PCON 中所有定义位均为 0。要想使单片机进入待机或掉电工作方式，只要执行一条使 IDL 或 PD 位为“1”的指令即可。下面介绍两种低功耗方式操作过程。

1. 待机方式

若写一个字节到 PCON，使 IDL＝1，单片机即进入待机方式。例如，执行过“ORL PCON，＃1”指令后，单片机即进入待机方式，此指令即为待机方式的启动指令。

在待机方式下，振荡器仍然工作，并向中断逻辑、串行口和定时/计数器电路提供时钟，但向 CPU 提供时钟的电路被阻断，因此 CPU 不能工作，与 CPU 有关的如 SP、PC、PWS、ACC 以及全部通用寄存器也都被“冻结”在原状态。

终止待机方式的方法有以下两种：

一是通过硬件复位。由于在待机方式下时钟振荡器一直在运行，RST 引脚上的有效信号只需保持两个时钟周期就能使 IDL 置 0，单片机即退出待机状态，从它停止运行的地方恢复程序的执行，即从空闲方式的启动指令之后继续执行。

二是通过中断方法。若在待机期间，任何一个允许的中断被触发，IDL 都会被硬件置 0，从而结束待机方式。其实在中断服务程序中只需安排一条 RETI 指令，就可以使单片机恢复正常工作后返回断点继续执行程序。

2. 掉电保护方式

当 CPU 执行一条置 PCON 寄存器的 PD 为 l 的指令后，系统进入掉电工作方式。在这种工作方式下，内部振荡器停止工作。由于没有振荡时钟，因此，所有的功能部件都停止工作。但内部 RAM 区和特殊功能寄存器区的内容被保留，而端口的输出状态值都被存在对应的 SFR 中。

退出掉电方式的唯一方法是硬件复位。复位后所有特殊功能寄存器的内容初始化，但不改变内部 RAM 中的数据。

在掉电工作方式下，V_{CC}可以下降到 2 V；但在进入掉电方式以前，V_{CC}不能降低。而在准备退出掉电方式之前，V_{CC}必须恢复正常的工作电压，并保持一段时间（10ms），使振荡器重新启动并稳定后方可退出掉电方式。

方法技巧 单片机在运行过程中，如发生掉电故障，将会使系统数据丢失，为此，80C51 单片机设置有掉电保护措施，进行掉电保护处理。其具体作法如下。

（1）数据转存

所谓数据转存是指当电源出现故障时，应立即将系统的有用数据转存到内部 RAM 中。数据转存是通过中断服务程序完成的，即通常所说的“掉电中断”。

因为单片机电源端（V_{CC}）都接有滤波电容，掉电后电容储存的电能尚能维持有效电压几毫秒，足以完成一次掉电中断操作。为此应在单片机系统中设置一个电压检测电路，一旦检测到电源电压下降，立即通过 INT0 或 INT1 产生外部中断请求，中断响应后执行中断服务程序，把有用数据送内部 RAM 中保护起来。

（2）接通备用电源

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

为了保存转存后的有用数据，掉电后应给内部 RAM 供电。为此，系统应预先装有备用电源，并在掉电后立即接通备用电源。备用电源由单片机的 RST/VPD 引脚接入。为了在掉电时能及时接通备用电源，系统中还需具有备用电源与 V_{CC} 电源的自动切换电路，如图 1-11 所示。

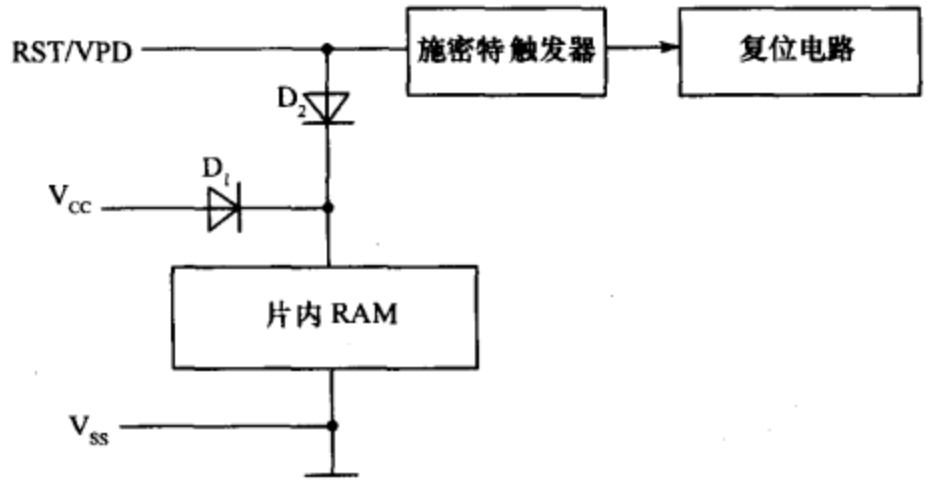


图 1-11　备用电源与 V_{CC} 电源的自动切换电路

切换电路由两个二极管组成，当电源电压 V_{CC} 高于 RST/VPD 引脚的备用电源电压时，D1 导通，D2 截止，内部 RAM 由 V_{CC} 电源供电。而当 V_{CC} 电源降至备用电源电压以下时，则 D1 截止，D2 导通，内部 RAM 由备用电源供电。这时单片机就进入掉电保护方式。

由于备用电源容量有限，为减少消耗，掉电后时钟电路和 CPU 皆停止工作，只有内部 RAM 单元和专用寄存器继续工作，以保持其内容。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第二章　单片机C语言入门

学习单片机不是一件容易的事，一来要购买实验板、编程器、仿真器等实验设备；二来要学习编程语言；此外，还有众多种类单片机的选择。在众多单片机中，51单片机风行很久，学习资料很多，是初学者较好的选择之一。51单片机的编程语言常用的有两种：一种是汇编语言，另一种是C语言。汇编语言的机器代码生成效率很高，但可读性不强，复杂一点的程序就更难读懂。而C语言在大多数情况下其机器代码生成效率和汇编语言相当，但可读性和可移植性却远远超过汇编语言；而且C语言还可以嵌入汇编语言来解决高时效性的代码编写问题。对于开发周期来说，中大型软件的编写用C语言的开发周期通常大大短于汇编语言。因此，在掌握一定汇编语言的基础上，就需要进一步学习C语言编程了。

第一节　认识C语言

一、C语言的发展过程

C语言是在20世纪70年代初问世的。1978年，美国电话电报公司(AT&T)贝尔实验室正式发表了C语言。同时由B. W. Kernighan和D. M. Ritchit合著了著名的《THE C PROGRAMMING LANGUAGE》一书。通常简称为《K&R》，也有人称之为《K&R》标准。但是，在《K&R》中并没有定义一个完整的标准C语言，后来美国国家标准协会(ANSI)在此基础上制定了一个C语言标准，于1983年发表，通常称之为ANSI C。到1987年，ANSI又公布了新标准——87 ANSI C，即现行的C语言标准。

二、C语言的特点

C语言是一种结构化语言，它层次清晰，便于按模块化方式组织程序，易于调试和维护。C语言的表现能力和处理能力极强，它不仅具有丰富的运算符和数据类型，便于实现各类复杂的数据结构；还可以直接访问内存的物理地址，进行位(bit)一级的操作。由于C语言实现了对硬件的编程操作，因此，C语言集高级语言和低级语言的功能于一体，效率高，可移植性强，特别适合单片机系统的编程与开发。

三、单片机采用C语言编程的好处

与汇编语言相比，C语言在功能上、结构性、可读性、可维护性上有明显的优势，因而易学易用。用过汇编语言后再使用C语言来开发，体会更加深刻。单片机采用C语言编程有以下几点好处：

① 编程调试灵活方便。C语言编程灵活，同时，当前几乎所有单片机都有相应的C

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

语言级别的仿真调试系统，调试十分方便。

② 生成的代码编译效率高。当前较好的 C 语言编译系统编译出来的代码效率只比直接使用汇编语言低 20%左右，如果使用优化编译选项，还可以更低。

③ 模块化开发。目前的软硬件开发都向模块化、可复用性的目标发展。不管是硬件还是软件，都希望其有比较通用的接口，以便在以后的开发中如果需要实现相同或者相近的功能，就可以直接使用以前开发过的模块，尽量不做或者少做改动，以减少重复劳动。如果使用 C 语言开发，数据交换可方便地通过约定实现，有利于多人协同进行大项目的合作开发。同时，C 语言的模块化开发方式使开发出来的程序模块可不经修改，直接被其他项目所用，这样就可以很好地利用已有的大量 C 程序资源与丰富的库函数，从而最大程度地实现资源共享。

④ 可移植性好。由于不同系列的单片机的 C 语言编译工具都是以 ANSI C 为基础进行开发的，因此，一种 C 语言环境下所编写的 C 语言程序，只需将部分与硬件相关的地方和编译连接的参数进行适当修改，就可方便地移植到另外一种系列上，例如，C51 下编写的程序通过改写头文件以及少量的程序行，就可以方便地移植到 PIC 系列上。

⑤ 便于项目维护管理。用 C 语言开发的代码便于开发小组计划项目、灵活管理、分工合作以及后期维护，基本上可以杜绝因开发人员变化而给项目进度、后期维护或升级所带来的影响，从而保证整个系统的品质、可靠性以及可升级性。

第二节　简单C 语言程序的构成

一、简单 C 语言程序

假设在单片机 AT89C51 的 P1.0 脚接有一只发光二极管，二极管的负极接 P1.0 脚，正极通过限流电阻接+5V，现在让发光二极管每隔 0.5s 闪烁一次，用 C 语言编写的程序如下：

```
#include<reg51.h>
sbit P10=P1^0;                    //定义位变量
void Delay(unsigned int i)        //延时程序，i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)                 //变量 i 由实际参数传入一个值，因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
}
void main()
{
    for(;;)
    { P10=1;                      //灯灭
    Delay(500);                   // 将实际参数 500 传递给形式参数 i，延时 0.5s
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
        P10=0;                          //灯亮
        Delay(500);                     //将实际参数 500 传递给形式参数 i,延时 0.5s
        }
}
```

下面对这个程序进行简要的分析。

程序的第一行是“文件包含”,所谓“文件包含”是指一个文件将另外一个文件的内容全部包含进来。所以,这里的程序虽然只有几行,但 C 编译器在处理的时候却要处理几十行或几百行。为加深理解,可以用任何一个文本编辑器打开 keil\c51\inc 文件夹下面的 reg51.h 来看一看里面有什么内容,在 C 编译器处理这个程序时,这些内容也会被处理。这个程序包含 reg51.h 的目的就是为了使用 P1 这个符号,即通知 C 编译器程序中所写的 P1 是指 80C51 单片机的 P1 端口,而不是其他变量,这是如何做到的呢? 用写字板程序打开 reg51.h 显示如下:

```
/*--------------------------------------------
REG51.H

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

#ifndef __REG51_H__
#define __REG51_H__

/*    BYTE Register    */
sfr P0     = 0x80;
sfr P1     = 0x90;
sfr P2     = 0xA0;
sfr P3     = 0xB0;
sfr PSW    = 0xD0;
sfr ACC    = 0xE0;
sfr B= 0xF0;
sfr SP     = 0x81;
sfr DPL    = 0x82;
sfr DPH    = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0    = 0x8A;
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
sfr TL1   = 0x8B;
sfr TH0   = 0x8C;
sfr TH1   = 0x8D;
sfr IE    = 0xA8;
sfr IP    = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;
/*   BIT Register    */
/*   PSW     */
sbit CY    = 0xD7;
sbit AC    = 0xD6;
sbit F0    = 0xD5;
sbit RS1   = 0xD4;
sbit RS0   = 0xD3;
sbit OV    = 0xD2;
sbit P= 0xD0;
/*   TCON    */
sbit TF1   = 0x8F;
sbit TR1   = 0x8E;
sbit TF0   = 0x8D;
sbit TR0   = 0x8C;
sbit IE1   = 0x8B;
sbit IT1   = 0x8A;
sbit IE0   = 0x89;
sbit IT0   = 0x88;
/*   IE     */
sbit EA    = 0xAF;
sbit ES    = 0xAC;
sbit ET1   = 0xAB;
sbit EX1   = 0xAA;
sbit ET0   = 0xA9;
sbit EX0   = 0xA8;
/*   IP     */
sbit PS    = 0xBC;
sbit PT1   = 0xBB;
sbit PX1   = 0xBA;
sbit PT0   = 0xB9;
sbit PX0   = 0xB8;
/*   P3    */
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
sbit RD   = 0xB7;
sbit WR   = 0xB6;
sbit T1   = 0xB5;
sbit T0   = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD  = 0xB1;
sbit RXD  = 0xB0;
/*  SCON  */
sbit SM0  = 0x9F;
sbit SM1  = 0x9E;
sbit SM2  = 0x9D;
sbit REN  = 0x9C;
sbit TB8  = 0x9B;
sbit RB8  = 0x9A;
sbit TI   = 0x99;
sbit RI   = 0x98;
#endif
```

可以看到:“sfr P1= 0x90;”,即定义符号 P1 与地址 0x90 对应,熟悉 80C51 内部结构的读者不难看出,P1 口的地址就是 0x90。

程序的第二行用符号 P10 来表示 P1.0 引脚,在 C 语言里,如果直接写 P1.0,C 编译器并不能识别,而且 P1.0 也不是一个合法的 C 语言变量名,所以得给它另起一个名字,这里起的名为 P10,可是 P10 是否就是 P1.0 呢,所以必须给它们建立联系,这里使用了 Keil C 的保留字 sbit 来定义。

main 称为“主函数”,每一个 C 语言程序有且只有一个主函数,函数后面一定有一对大括号“{}”、在大括号里面书写其他程序。

Delay(500)的用途是延时,由于单片机执行指令的速度很快,如果不进行延时,灯亮之后马上就灭,灭了之后马上就亮,速度太快,人眼根本无法分辨,所以需要进行适当的延时,这里采用自定义函数 Delay(500),以延时 0.5s 的时间。

需要说明的是,Delay(500)并不是由 Keil C 提供的,即你不能在任何情况下写这样一行程序以实现延时,如果在编写其他程序时写上这么一行,会发现编译通不过。注意观察本程序会发现,在使用 Delay(500)之前,第 3 行~9 行已对 Delay 函数进行了事先定义,因此,在主程序中才能采用 Delay(500)。

Delay 后面有一个小括号,小括号里有数据“500”,这个“500”被称为“实际参数”,用它可以在一定范围内调整延时时间的长短,这里用“500”来要求延时时间为 0.5s。

二、C 语言程序的构成

通过以上简单的 C 语言程序,可以总结出以下几点:

① C 程序是由函数构成的,一个 C 源程序至少包括一个函数,一个 C 源程序有且只

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

有一个名为 main()的函数，也可能包含其他函数，因此，函数是 C 程序的基本单位。主程序通过直接书写语句和调用其他函数来实现有关功能，这些其他函数可以是由 C 语言本身提供给用户的，这样的函数称之为库函数(本例未使用库函数)，也可以是用户自己编写的(如本例中的 Delay(…)函数)，这样的函数称之为用户自定义函数。那么库函数和用户自定义函数有什么区别呢？简单地说，任何使用 C 语言的人，都可以直接调用 C 的库函数而不需要为这个函数写任何代码，只需要包含具有该函数说明的相应的头文件即可；而自定义函数则是完全个性化的，是用户根据自己需要而编写的。

② 一个函数由两部分组成：

• 函数的首部，即函数的第一行。包括函数名、函数类型、函数属性、函数参数(形式参数)名、参数类型。

例如：void Delay(unsigned int i)

一个函数名后面必须跟一对圆括号，即便没有任何参数也是如此。

• 函数体，即函数首部下面的大括号“{}”内的部分。如果一个函数内有多个大括号，则最外层的一对“{}”为函数体的范围。

③ 一个 C 语言程序，总是从 main 函数开始执行的，而不管物理位置上这个 main()放在什么地方。

④ 主程序中的 Delay 如果写成 delay 就会编译出错，即 C 语言区分大小写，书写时一定要注意。

⑤ C 语言书写的格式自由，可以在一行写多个语句，也可以把一个语句写在多行。没有行号(但可以有标号)，书写的缩进没有要求。但是建议用户自己按一定的规范来写，可以给自己带来方便。

⑥ 每个语句和资料定义的最后必须有一个分号，分号是 C 语句的必要组成部分。

⑦ 可以用/*…*/的形式为 C 程序的任何一部分作注释，在“/*”开始后，一直到“*/”为止的中间的任何内容都被认为是注释，所以，在书写特别是修改源程序时特别要注意，有时无意之中删掉一个“*/”，结果，从这里开始一直到遇到下一个“*/”中的全部内容都被认为是注释了。

需要说明的是，如果使用的是 Keil Cx51 开发软件，那么，该软件也支持 C++风格的注释，就是用“//”引导的后面的语句是注释。这种风格的注释，只对本行有效，所以不会出现上面的问题，而且书写比较方便，因此在只需要一行注释的时候，往往采用这种格式。但要注意，只有 Keil? C 支持这种格式，早期的 Franklin? C 软件以及 PC 机上用的 TC 都不支持这种格式的注释，用上这种注释，编译时通不过，会报告编译错误。

第三节　单片机C 语言开发步骤

用 C 语言程序开发单片机，除了进行软件的设计、调试和软硬件联调外，还要根据功能要求进行硬件电路设计。具体来说，开发步骤包括以下几点。

一、硬件系统设计

单片机是一个功能强大的智能微控制器，它把传统的单元功能电路高密度集成在一

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

片集成电路内，因此，用单片机开发产品简化了硬件电路，很容易实现产品的小型化，智能化。

一个单片机应用系统的硬件设计包括两大部分内容：一是单片机系统的扩展部分设计。它包括存储器扩展和接口扩展。二是各功能模块的设计。如信号测量功能模块、信号控制功能模块、人机对话功能模块、通信功能模块等，根据系统功能要求配置相应的A/D、D/A、键盘、显示器、打印机等外围设备。

在进行硬件设计时，所涉及的具体电路可借鉴他人在这方面进行的工作。因为经过别人调试和考验过的电路往往具有一定的合理性。如果在此基础上，结合自己的设计目的进行一些修改，则是一种简便、快捷的做法。

二、编写C语言程序

想让单片机按用户的想法完成一项任务，必须先编写供其使用的程序，编写单片机的程序应使用该单片机可以识别的“语言”。

Keil Cx51 软件带一个集成开发环境 uVision2，uVision2 是一种文件管理编译环境，集成了文本编辑处理、编译链接、项目管理和仿真等多种功能，是相当强大的 Cx51 开发工具，十分方便 C 语言程序的编写和修改。

三、编译源程序

用户编写的 C 语言(扩展名为 .c)程序，只是给自己看的，这个程序还必须经过 Keil Cx51 编译器编译成单片机可以识别的机器码，生成 .hex(十六进制)或 .bin(二进制)目标文件，以便用编程器烧写到单片机中。

四、应用程序的仿真调试

编译通过只是说明源程序没有语法错误，至于源程序中存在的其他错误，往往还需要通过反复的仿真调试才能发现。所谓仿真即是对目标样机进行排错、调试和检查，一般分为硬件仿真和软件仿真两种。

硬件仿真是通过仿真器(仿真机)与目标样机联机进行实时在线仿真，如图 2-1 所示。一块单片机应用电路板包括单片机部分及为达到使用目的而设计的应用电路。硬件仿真就是利用仿真器来代替应用电路板(称目标样机)的单片机部分，由仿真器向目标样机的应用电路部分提供各种信号、数据进行测试、调试的方法。这种仿真可以通过单步执行、连续执行等多种方式来运行程序，并能观察到单片机内部的变化，便于修改程序中的错误。图中，将仿真插头插到电路板上的单片机插座上，此时可将仿真器看做是一个独立的单片机，通过运行 PC 上的仿真软件(如 Keil Cx51 软件)，使目标样机处于一个真实的工作环境之中，可模拟开发单片机的各种功能。显然，这种仿真因为需要仿真器、电路板等硬件装置，因而投资较大。

软件仿真是指在 PC 上运行仿真软件来实现对单片机的硬件模拟、指令模拟和运行状态模拟，故这种仿真方法又称为软件(程序)模拟调试。它不需要硬件，简单易行。软件仿真的缺点是不适用于实时性很强的单片机应用系统的调试，在实时性要求不高的场合，软件仿真已被广泛应用。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

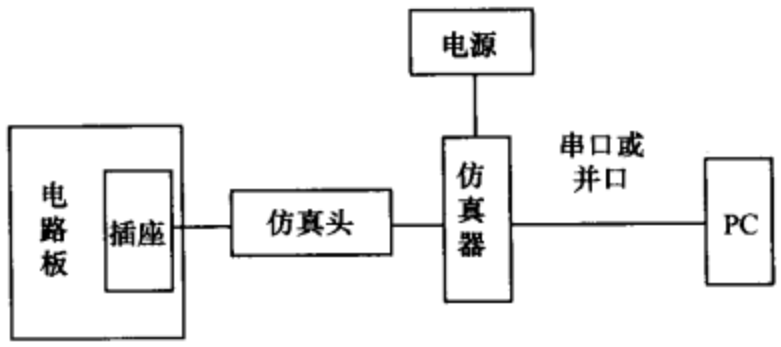


图 2-1　硬件仿真连接图

Keil Cx51 等很多软件都具有软件模拟仿真的功能，通过适当设置，可方便地进行软件模拟仿真。

五、单片机应用程序的烧写

仿真调试通过后，要将扩展名为 *.hex 或 *.bin 的代码文件送到单片机系统中，单片机在电路中才能按用户的“计划”去工作。

使用编程器可将 .hex 或 *.bin 代码文件烧写到单片机(或外接的可编程 ROM)中。编程器一般通过并口或串口与 PC 连接，具有相应的服务程序；在连接好 PC 与编程器后运行其服务程序，在服务程序中先选择所要编程的单片机型号，再调入前面所得到的 .hex 目标文件，编程器就将这个目标文件烧写到单片机中。烧写后再将单片机插入到电路板上相应的插座上，如符合设计要求，则完成工作。

六、系统脱机运行检查

用户不可能一次就把自己的“计划”用单片机的语言将源程序完美正确地写好，这就需要反复修改源程序，反复编译、烧写到单片机中，反复将单片机装到电路中去实验，针对硬件或软件出现的问题，进行修改，逐步进行完善。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第三章　单片机实验硬件环境的建立

单片机和 PC 机一样，是实践性很强的一门技术，有人说“计算机是玩出来的”，单片机也一样，只有多“玩”，也就是多练习、多实际操作，才能真正掌握它。因此，本章将介绍一些适用于初学者且性价比较高的单片机开发系统，主要包括实验板、仿真器和编程器。

第一节　单片机实验板

单片机是操作性、实践性很强的技术，需要反复练习和实践，因此，学习时需要一块 51 实验板，通过做一系列的实验，可以比较容易地领会单片机中枯燥、难懂的专业术语。下面主要介绍两种性价比较好的实验板。

一、下载型实验板

下载型实验板是平凡单片机工作室(http://www.mcustudio.com)开发的一种实用的实验板，外观如图 3-1 所示，其电路原理图在附赠的光盘中。

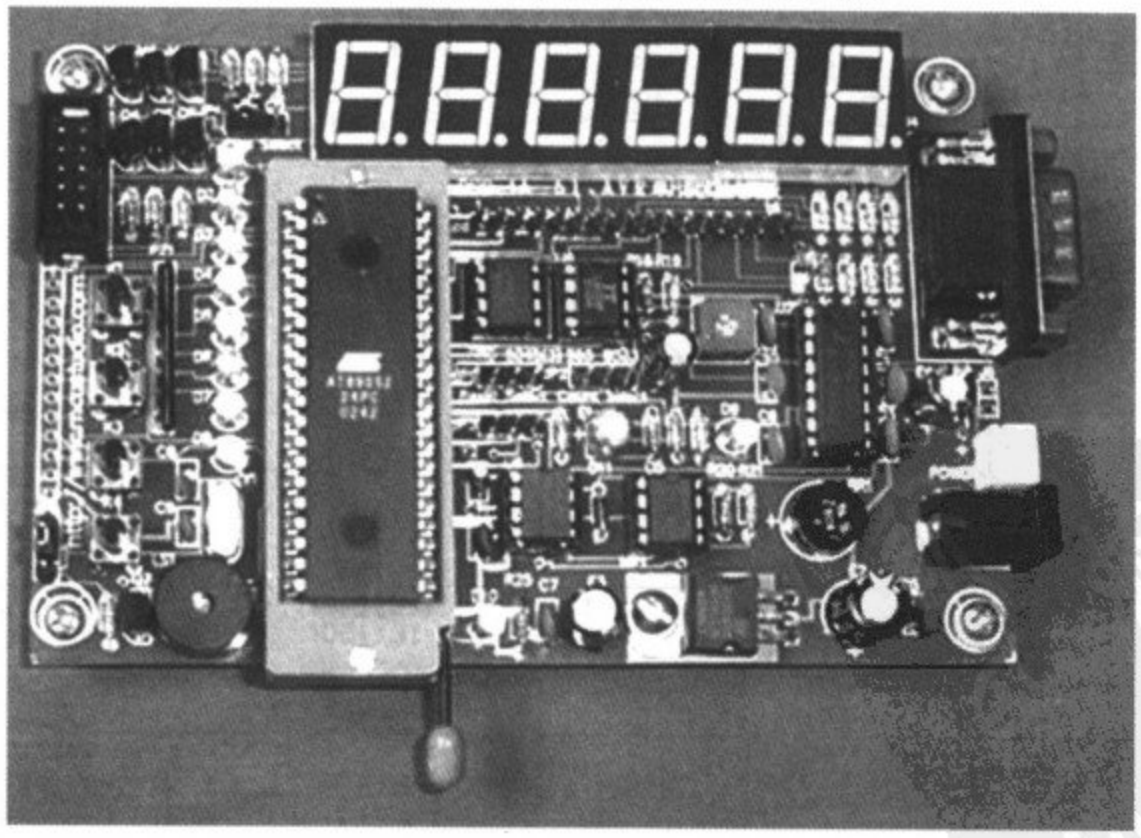


图 3-1　下载型实验板

实验电路板上安装了 6 位数码管，8 个发光二极管，4 个按钮开关，1 个简单的音响电路，1 个用于计数实验的振荡器，AT24CXXX 芯片插座，X5045 芯片插座，RS-232 串行接口，PCF8563 实时钟芯片插座，字符型 LCD 插座，带有标准 ISP 插座，可用下载线对 AT89S5X 或 AT90S8515 单片机编程。本板还具有其他一些特点，如可使用 89 系列芯片

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

学习 51 系列单片机，也可使用 AT90S8515 芯片学习 AVR 单片机；板上还设有扩展接口，可方便地扩展其他串行接口芯片。

使用这块实验板可以进行流水灯、人机界面程序设计，音响、中断、计数器等基本编程练习，还可以学习 I^2C 接口芯片使用、SPI 接口芯片使用、字符型液晶接口技术、与 PC 机进行串行通信等目前较为流行的技术。

1. 硬件结构

(1)发光二极管

单片机的 P1 端口接了 8 个发光二极管，这些发光二极管的负极接到 P1 端口各引脚，而正极则通过一个排电阻接到正电源端。发光二极管亮的条件是 P1 口相应的引脚为低电平，即如果 P1 口某引脚输出为 0，则相应的灯亮；如果输出为 1，则相应的灯灭。

(2)数码管

单片机的 P0 口和 P2 口的部分引脚构成了 6 位 LED 数码管驱动电路。这里 LED 数码管采用了共阳型，使用 6 只 PNP 型三极管作为片选端的驱动。基极通过限流电阻分别接 P2.2～P2.7，集电极分别向 6 只数码管供电。图 3-2 是显示部分的电路原理图。

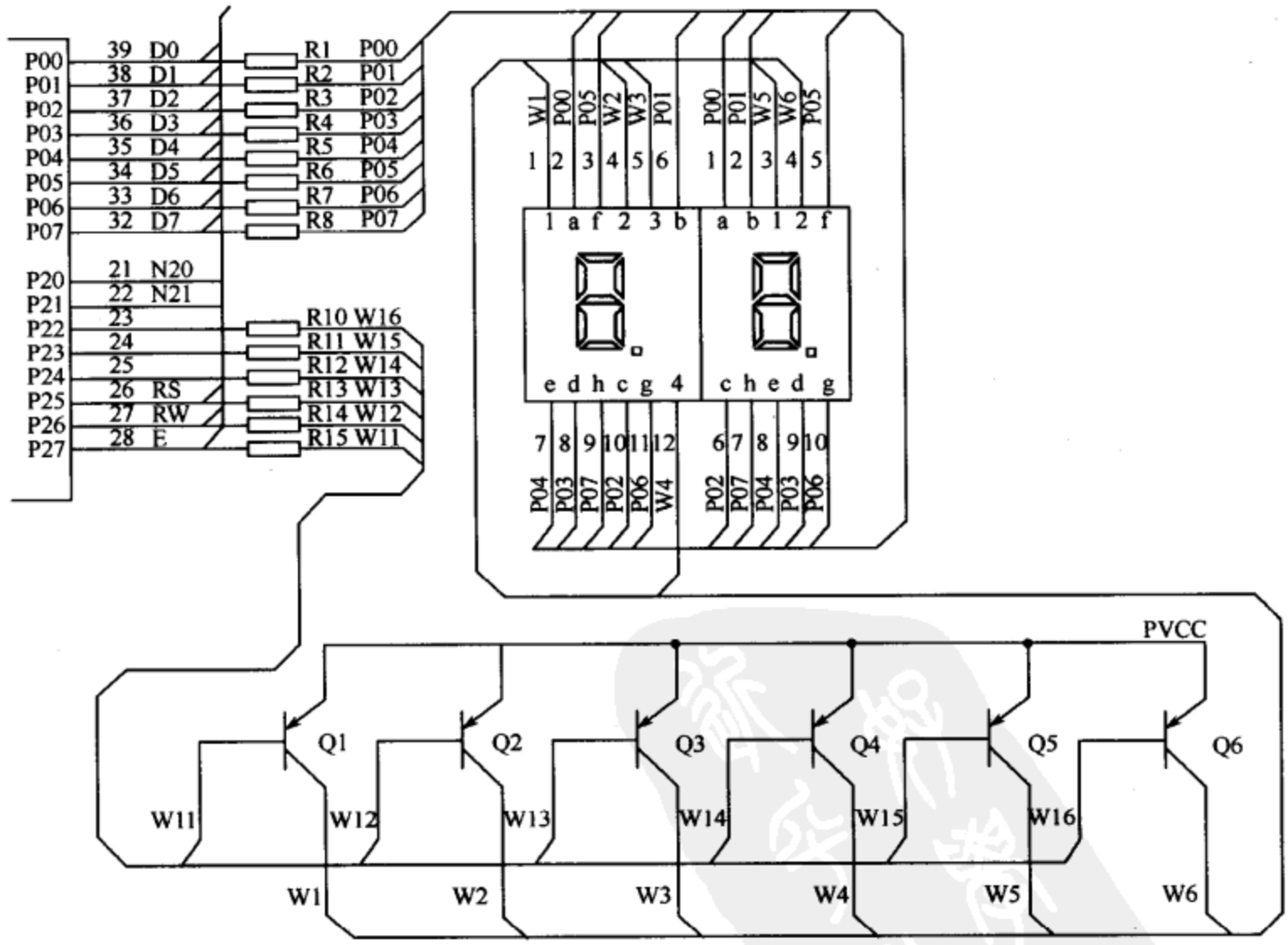


图 3-2　显示部分电路原理图

(3)串行接口

串行通信功能是目前单片机应用中经常要用到的功能，80C51 系列单片机 P3.0 和 P3.1 引脚的第二功能是串行口 RXD 与 TXD，其内部的串行接口电路具有全双工异步通信功能，但是单片机输出的信号是 TTL 电平，为获得电平匹配，实验板上扩充了一块

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

HIN232 芯片，利用该芯片进行电平转换，该芯片内部有电荷泵，只要单一的 5V 电源供电即可自行产生 RS-232 所需的高电压，使用方便。

(4)按键输入

P3 口的 P3.2～P3.5 接 K1～K4 按钮开关，用作键盘。

(5)计数源

实验板有两路脉冲信号产生，其中一路由 555 集成电路及相关阻容元件构成典型的多谐振荡电路，输出方波，在输出端接有发光二极管，用于指示振荡器的输出；另一路由 PCF8563 集成电路提供。PCF8563 是实时钟芯片，通过编程可输出 1Hz、32Hz、1024Hz、32768Hz 的脉冲信号。在 PCF8563 的输出端接有发光二极管，用于指示振荡器的输出。通过 JP2 插针座可分别选择这两个脉冲信号中的一个作为单片机的计数信号。

(6)音响接口

P3.2 引脚经过三极管驱动一个无源蜂鸣器，构成一个简单的音响电路，如图 3-3 所示。由于 P3.2 同时作为按键输入使用，为了避免按键操作对发声电路的影响，使用 JP5 插针，只在需要时才用短路子将两个引脚连起来，这时 P3.2 作为输出口来使用。

(7)AT24CXXX 芯片接口

在单片机应用系统中，经常会有一些数据需要长期保存，传统的方法是用 RAM 加后备电池的方法，但这种方法成本较高，电路也较复杂；近年来，多采用 EEPROM 来长期保存数据。与 RAM 相比，EEPROM 不能够无限次地擦除和写入数据，这是其缺点，但是，在断电之后，不需要特殊的供电方式，这是其优点。AT24CXXX 是 EEPROM 中的一种，仅 8 个引脚，采用 I^2C 总线接口。为学习该芯片的使用，在实验板上设计有 I^2C 接口电路，可进行芯片的读写实验。实验板中 P3.6 引脚接串行时钟线(SCL)，P3.7 接串行数据线(SDA)。

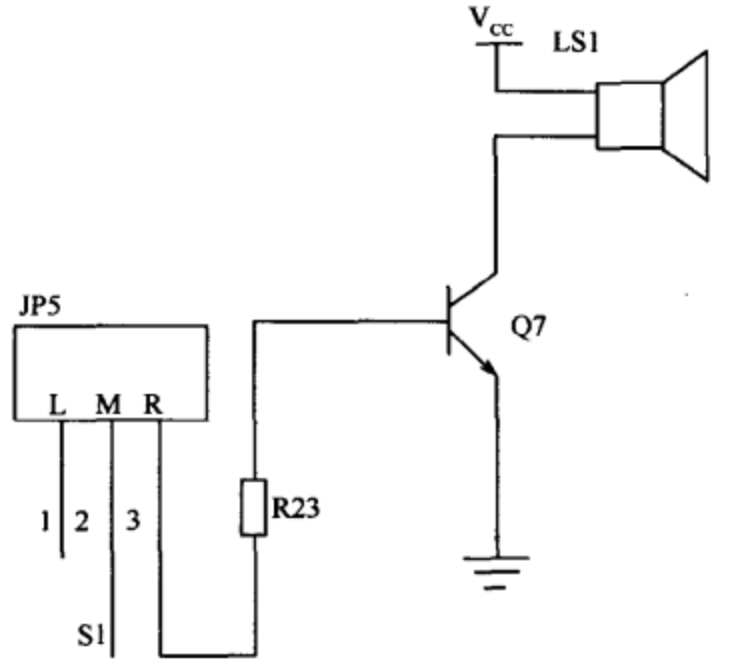


图 3-3　音响电路

(8)实时钟 PCF8563 芯片接口

PCF8563 是目前常用的低功耗的 CMOS 实时时钟/日历芯片。它提供一个可编程时钟输出、一个中断输出和掉电检测器，所有的地址和数据通过 I^2C 总线接口串行传递。实验板上设计的 PCF8563 与 AT24CXXX 芯片共同挂接在 I^2C 总线上，通过插针选择是否向单片机送出中断信号，以及是否将振荡信号送到单片机的计数端。

(9)X5045 接口

X5045 是一块多功能的芯片，它具有以下一些功能：上电复位、电压跌落检测、看门狗定时器、512 字节的 EEPROM。该芯片采用三线制 SPI 接口方式与单片机相连，这也是目前应用比较广泛的一个芯片。通过学习这块芯片与单片机接口的方法，还可以了解和掌握三线制 SPI 总线接口的工作原理及一般编程方法。

硬件电路上，P2.1 接 X5045 的 CS 端，P3.7 接 X5045 的 SI 和 SO 端，P3.6 接 X5045

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

的 SCK 端，P2.0 接 X5045 的 WP 端。

(10)字符型液晶接口

液晶显示器由于体积小、质量小、功耗低等优点，日渐成为各种便携式电子产品的理想显示器。从液晶显示器显示内容来分，可分为段式、字符式和点阵式三种。其中字符式液晶显示器以其价廉、显示内容丰富、美观、无需定制、使用方便等特点成为 LED 显示器的理想替代品。字符型液晶显示器专门用于显示数字、字母、图形符号，并可显示少量自定义符号。这类显示器均把 LCD 控制器、点阵驱动器、字符存贮器等做在一块板上，再与液晶屏一起组成一个显示模块，因此，这类显示器安装与使用都较简单。

字符型液晶显示器一般均采用 HD44780 及兼容芯片作为控制器，因此，其接口方式基本是标准的。实验板上带有 LCD 接口，可直接与字符型液晶显示器相连。实验板上数据线被连到 P0 口，P2.5 接 RS 端，P2.6 接 RW 端，P2.7 接 E 端。

(11)扩展接口

标号为 J1 的接口为 12 芯插座，是扩展接口。该接口将 P1 口、T1 计数器端、INT1 外中断接口端及电源端引出，供扩展板使用。

2. 使用简介

(1)电源提供

实验板需外接电源，可以通过插座 J5 向实验板供电，要求输出的直流电压在 9V～15 V 之间，电流不小于 300 mA。由于板上装有整流全桥，因此不必考虑电源的输出极性，直接将插头插入其中即可。

(2)复位选择

实验板提供了两种复位电路，即 RC 复位与外接芯片复位。JPl 用于复位选择，在该插针座下标有 Reset Select 字样，很容易辨认。在该插针座上方左侧标有 RC 字样，右侧标有 X5045(3)字样。如果用短路子插于左侧，则选择 RC 复位电路，可避免刚开始对 X5045 芯片不熟悉而影响学习；如果用短路子插于右侧，则选择 X5045 复位，可用于测试 X5045 芯片的看门狗特性。不论短路子插于哪一侧，X5045 芯片内部的 EEPROM 存储器总是可用的。

注意 在使用 ISP 在线可编程功能时，必须将短路子拔除，既不选择 RC 复位，也不选择 X5045 复位，由下载线控制复位端。

(3)计数源选择

实验板提供了两个计数源，可供单片机做计数实验。第一个计数源由 555 电路提供，第二个计数源由 PCF8563 提供，通过 JP2 选择。JP2 的下方标有 Count Select 字样，上方左侧标有“555”，右侧标有“8563”，分别代表选择这两个计数源。

注意 当 P3.3 作为输入端使用时，应将短路子取下，不接于任何一方。

当使用 PCF8563 作为时钟源时，需对 PCF8563 芯片编程。通过编程，可提供 1Hz、32Hz、1024Hz 和 32768Hz 等多个标准信号源。

(4)音响电路工作选择

JP5 用于选择 P3.2 究竟工作于输出方式还是输入方式。当需要将 P3.2 作为驱动音响电路工作的输出端时，应将短路子插于 JP5 的下方(插座旁印有“]”标志)；反之应将短路子插于 JP5 的上方。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

(5)字符型 LCD 实验

进行 LCD 实验时,需断开数码管的供电电路。JP4 用于选择显示器。在 JP4 的下方标有 Disp Select 字样,上方分别标有 LED 和 LCD 字样。将短路子插于 LED 一方,选择 LED 作为显示器;插于 LCD 一方,则选择 LCD 作为显示器。

本板提供了供 LCD 使用的 16 针标准接线插座,标号为 J3。在 J3 下方注有 LCD 字样;在 J3 的上方用数字 1~16,标出了接线的序号,安装时应注意与液晶显示模块上的引脚序号对应,下方的蓝色电位器用于调整 LCD 的对比度。

(6)ISP 功能的使用

拔去复位插座上的短路子,使复位端悬空。标号为 J1 的插座为 ISP 下载插座,将下载电缆与实验板正确连接。本插座采用标准接法,具体接法如表 3-1 所列。

表 3-1　ISP 插座接线

标号	名称	功能	标号	名称	功能
1	SCK	串行时钟	9	MOSI	主器件输出—从器件输入
3	MISO	主器件输入—从器件输出	2、10	GND	地
4	V_{CC}	电源	6、7、8	NC	未接
5	RST	复位端			

二、AT89C51 实验板

AT89C51 实验板是电子制作实验室(http://www.xie-gang.com)设计开发的,其外观如图 3-4 所示,电路原理图如图 3-5 所示。

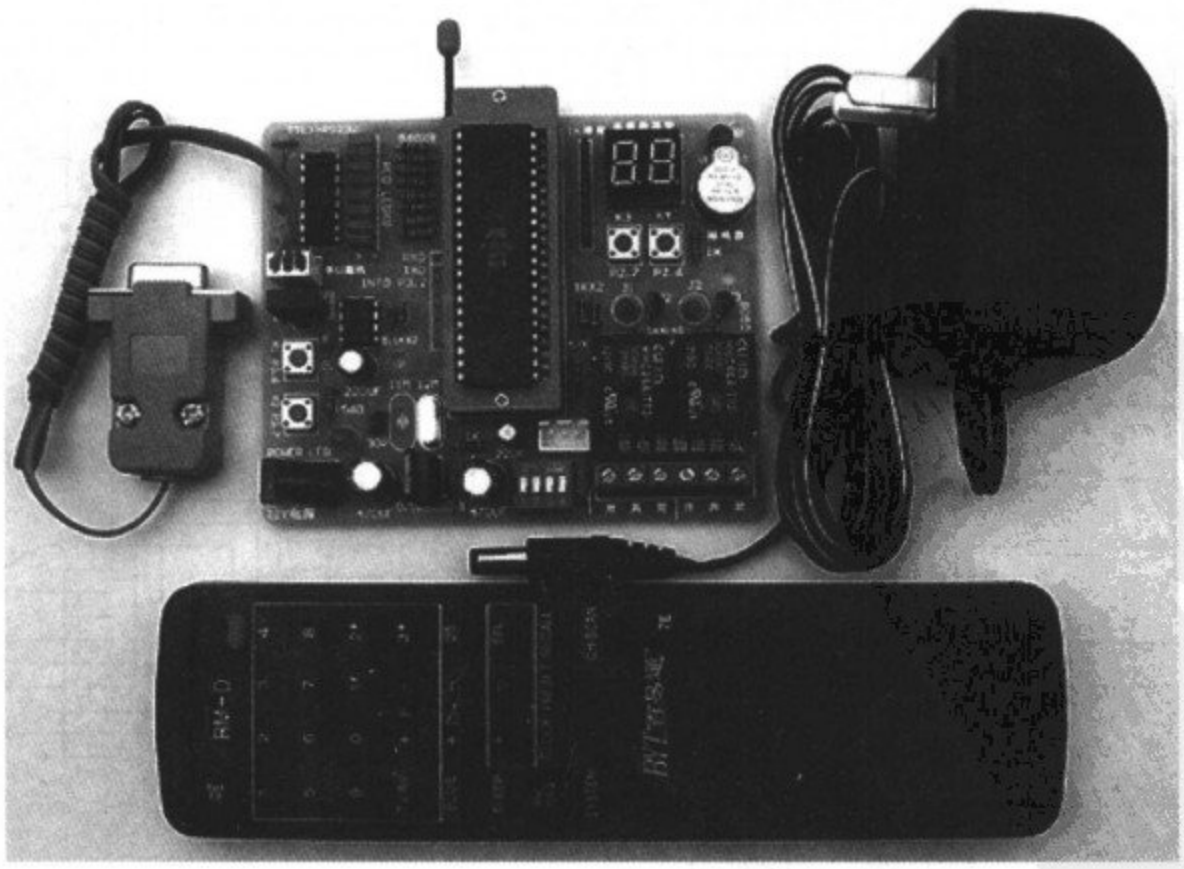


图 3-4　AT89C51 单片机实验板外观

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

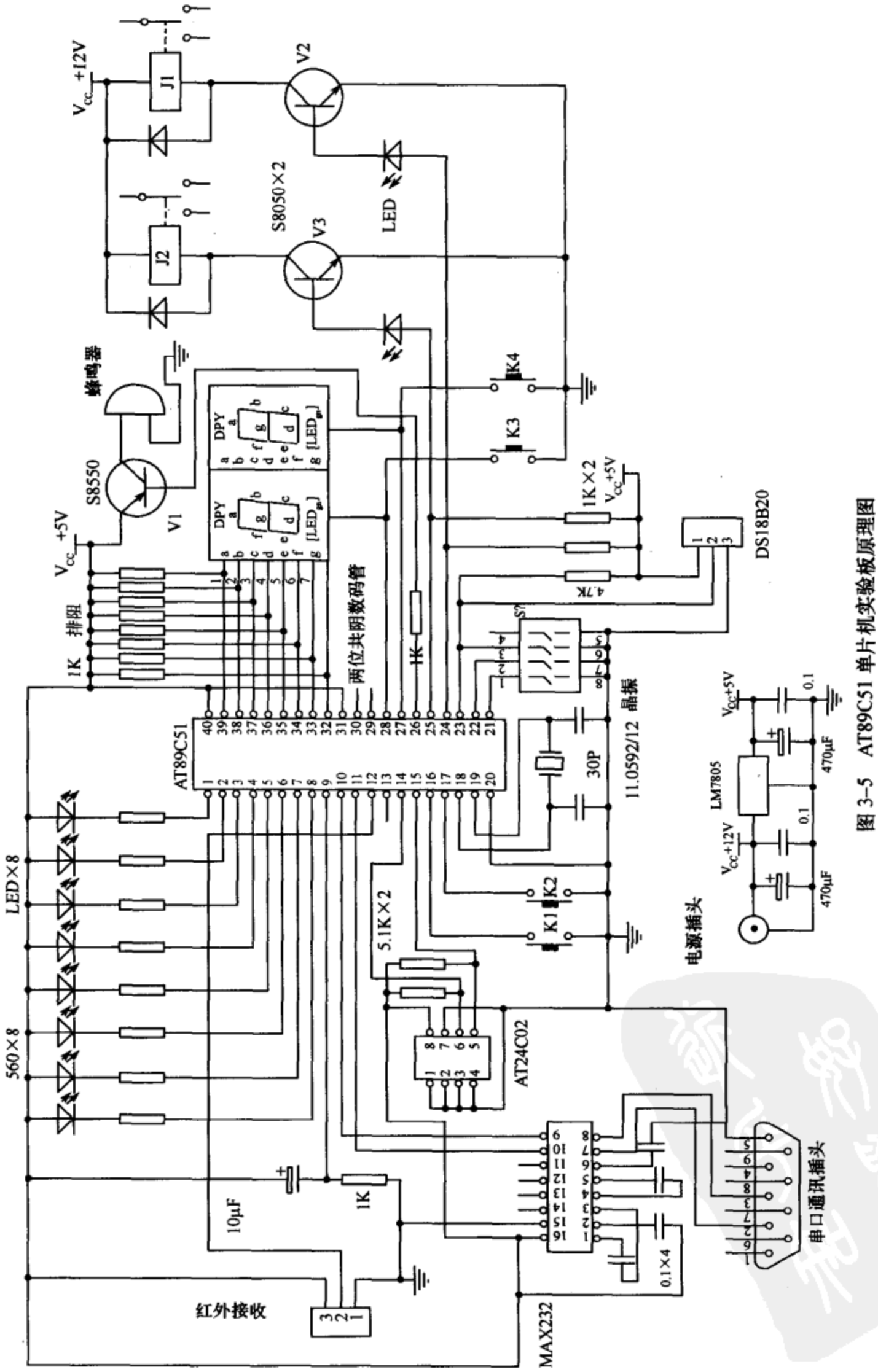


图 3-5 AT89C51 单片机实验板原理图

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

1. 简介

AT89C51 单片机实验开发板上安装了 AT89C51、发光管、数码管、蜂鸣器、继电器等器件。板上有独立的稳压电路；只要向单片机中烧好程序，接上配套的电源就可以运行。板上的发光二极管是为进行单片机 I/O 口控制实验而设，用它可以进行流水灯实验、二进制显示以及其他实验的指示。板上的两只数码管用于 LED 显示器程序；同时也是学习其他程序的窗口，例如定时器显示、外来数据显示、定时记数显示等。4 只按钮开关，用于外界对程序的控制，拨码开关可以模拟外来数据，即将拨码开关的数据让单片机读入。蜂鸣器用于单片机发声实验，同时也是其他实验声音提示的设备。另外，板上还设有红外遥控接收器，可用于各种红外线遥控和红外线数据传输。

2. 硬件电路

(1)电源电路

AT89C51 实验开发板需要配备输出直流电压为 10V～15V 的稳压电源，并且插头极性为内正外负的电源，稳压电源输出的直流电压通过专门的电源插座把直流电压引入实验开发板，加到实验开发板的电源电路，如图 3-6 所示。

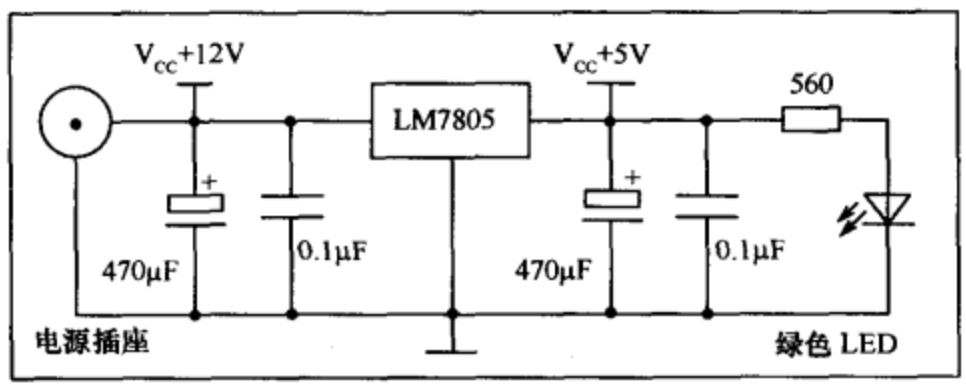


图 3-6　实验开发板内的电源电路

电源部分分两路，一路直接提供 12V 的直流电源，主要是提供给继电器使用的；另一路通过三端稳压芯片 7805 稳压成 5V 直流电源，提供给单片机系统使用，电阻和绿色的 LED 组成 5V 电源的工作指示电路，只要电源部分正常，绿色的 LED 就会点亮，据此可以判断整个电源部分工作是否正常。

(2)时钟电路和复位电路

单片机的时钟电路有一个 12MHz 的晶振和两个 30pF 的小电容组成，它们决定了单片机的工作时间精度为 1μs(微秒)。复位电路由 22μF 的电容和 1kΩ 的电阻组成，以使单片机可靠复位。

方法技巧　判断单片机芯片及时钟系统是否正常工作有一个简单的办法，就是用万用表测量单片机晶振引脚(18 脚、19 脚)的对地电压。以正常工作的单片机用数字万用表测量为例，18 脚对地约 2.24V，19 脚对地约 2.09V。若怀疑复位电路有故障，可以采用模拟复位的方法来判断，单片机正常工作时第 9 脚对地电压为零，可以用导线短时间和+5V连接一下，模拟一下上电复位，如果单片机能正常工作，说明这个复位电路有问题。

(3)AT89C51 的 31 脚($\overline{EA}$)

实验开发板上采用的是 AT89C51 芯片，它内部自带 4KB 的 Flash 程序存储器，一般情况下，这 4KB 的存储空间足够使用，所以将 AT89C51 芯片的第 31 脚固定接高电平，只

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

用芯片内部的 4KB 程序存储器。

(4)发光二极管接口

图 3-7 是实验开发板的发光二极管电路原理图，从图中可以看到，AT89C51 的 P1 端口安装了 8 个发光二极管，这些发光二极管的负端接到端口引脚，而其正端则通过 8 个 560Ω 的电阻接到正电源端，这样，这些发光二极管发光的条件就是相应的端口为低电平。也就是说，如果 P1 端口某引脚输出为 0，则相应的灯亮；如果输出为 1，则相应的灯灭。

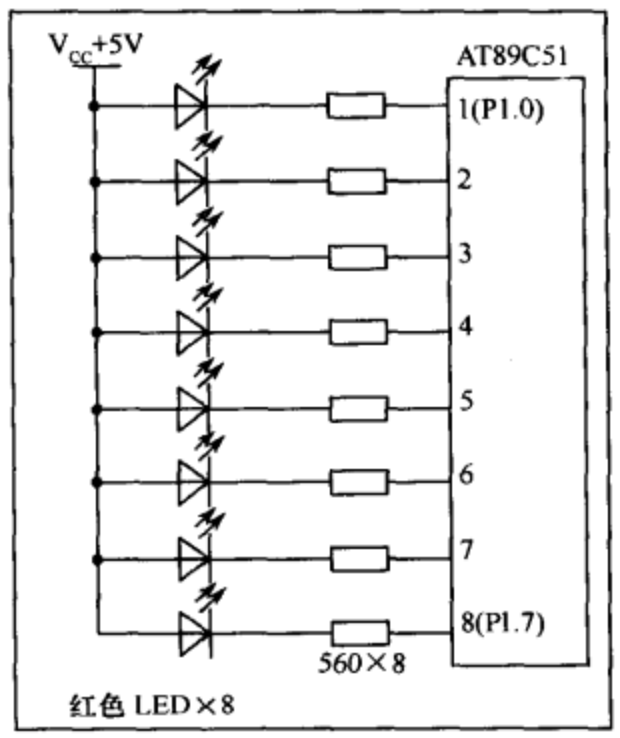


图 3-7　发光二极管接口

(5)数码管接口

P0 口和 P2 口的部分(P2.6、P2.7)引脚构成了 2 位 LED 数码管驱动电路，数码管有共阳和共阴，这里采用了共阴型。共阴型数码管对应的 abcdefh 是二极管的正极，所有二极管的负极连在一起，构成公共端，即片选端。

+5V 通过 1kΩ 的排阻直接给数码管的 8 个段位供电，P2.6(27 脚))和 P2.7(28 脚)端口分别控制数码管的个位和十位的供电，当相应的端口变成低电平时，相应的位可以吸入电流。单片机的 P0 口(32～39 脚)输出的数据相当于将数码管不要显示的数字段对地短路，这样数码管就会显示需要的数字。

(6)音响接口

AT89C51 单片机的 P2.5(26 脚)口控制一个 8550 的三极管，三极管控制电磁蜂鸣器的电源通断。

由于声音的频谱范围约在几十到几千赫兹，若能利用程序来控制单片机某个口线的高电平或低电平，则在该口线上就能产生一定频率的矩形波，接上喇叭就能发出一定频率的声音；若再利用延时程序控制“高”、“低”电平的持续时间，就能改变输出频率，从而改变音调。

(7)按键输入电路

P3 口的 P3.6(16 脚)、P3.7(17 脚)和 P2 口的 P2.6(27 脚)、P2.7(28 脚)接了 4 个按钮开关 K1～K4，由于是实验板，没有过多考虑抗干扰等问题，所以没有给这 4 个引脚接上拉电阻，直接利用了单片机芯片内部的弱上拉。

正常情况下，单片机的 P3.6、P3.7 和 P2.6、P2.7 端口都被程序初始化时置“1”，当有按键按下时，对应的单片机引脚被按钮开关下拉为“0”，这种方法比较直观，而且比较简单，在按键数量不多的场合下使用很广泛。因为机械开关在开关时有抖动，所以需要在程序中加一个软件去抖动程序。

(8)4 位拨码开关

4 位拨码开关接在 P2 口的 P2.0～P2.2，当开关拨到“ON”一侧时，对应的那路就会接通，反之断开。它在单片机中一般用于设置初始参数，而且不经常改变的场合。这里因为单片机引脚资源不够，所以只使用了拨码开关的第 1、2、3 位，第 4 位闲置。3 个开关可

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

以逻辑组合出 8 种状态。

(9)继电器电路

两个继电器由 P2 口的 P2.3、P2.4 控制，有了继电器，就可以控制一些负载，使实验开发板的实用功能大大增强。

这里继电器由相应的 S8050 三极管来驱动，开机时，单片机初始化后的 P2.3、P2.4 为高电平，+5V 电源通过电阻使三极管导通，所以开机后继电器始终处于吸合状态，如果在程序中给单片机一条指令：CLR P2.3 或者 CLR P2.4 的指令的话，相应三极管的基极就会被拉低到 0V 左右，使相应的三极管截至，继电器就会断电释放，每个继电器都有一个常开转常闭的接点，便于在其他电路中使用，继电器线圈两端反相并联的二极管是起到吸收反向电动势的功能，保护相应的驱动三极管，这种继电器驱动方式硬件结构比较简单。

(10)串行接口

通信功能是目前单片机应用中经常要用到的，本实验开发板设计了 MAX232 接口芯片，这样，该板就可以和 PC 机进行通信了。51 单片机的 P3 口的引脚 P3.0(10 脚)和 P3.1(11 脚)的第二功能是串行口 RXD 与 TXD，其内部的串行接口电路具有异步通信功能，但是单片机输出的信号是 TTL 电平，也必须以 TTL 电平作为输入，而 PC 机是标准的 RS-232 接口，以 12V 为高电平，OV 为低电平。为获得电平匹配，实验板上扩充了一片 MAX232 芯片，该芯片是一个电平变换芯片，利用该芯片，可以将单片机的 TTL 电子变换为 RS-232 所要求的电平，该芯片内部有电荷泵，只要单一 5V 电源供电即可自行产生高电压，使用非常方便。MAX232 的外围电路也很简单，只要 4 只电容即可，电容的值可以从 0.1μF 到 2μF，这里使用 4 只 0.1μF 的电容，实践证明，它们工作得很好。

要让实验板与 PC 机进行通信，必须配通信电缆，通信电缆可以使用普通的 3 芯线来制作，3 芯线的一端安装 3 芯插座，以便插到实验电路板上，另一端接一个 DB9 插座(母)，以便插入 PC 机后的串行口(COM1 或 COM2)中。

(11)红外遥控接口

AT89C51 的 P3.2(12 脚)外接一个红外线接收器，红外线接收器集红外线接收和放大于一体，体积小巧，适合于各种红外线遥控和红外线数据传输。

第二节　单片机仿真器

单片机仿真器的作用是在产品开发阶段用来替代单片机进行软硬件调试的开发工具。使用单片机仿真器可以对单片机程序进行单步、全速、断点等调试，检查程序运行中单片机 RAM、寄存器内容的变化，观察程序的运行情况；与此同时，还可以对硬件电路进行实时的调试。使用单片机仿真器可以迅速发现、纠正程序中的错误，从而大大缩短单片机开发的周期。

如果不使用单片机仿真器，而是利用单片机烧录器反复烧写单片机进行开发，对于程序设计中的错误就只能通过分析、猜测，然后修改程序、重新烧录、重新实验来完成，这样就大大增加了调试的难度，延长了开发时间，特别对于单片机开发经验并不丰富的开发者来说尤为困难。

目前市场上传统的仿真器价格都在几百元以上，由于仿真器是用来提高调试程序效

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

率的，并不是非需不可；但是，如果想模拟实验的真实性，最好还是拥有一台仿真器。

一、Insight SE-52 仿真器

Insight 系列单片机仿真开发系统是万利(Manley)公司在仿真器领域开发的产品，可实时仿真 51 全系列单片机，并且对目标系统没有任何限制：不占用单片机任何资源，完全实时仿真 P3.6、P3.7、P0、P2 口的特性，完全实时仿真中断特性和定时器特性，完全实时仿真休闲方式和掉电方式；可清晰地看到仿真器实际运行的全过程。

Insight SE-52 仿真器的外形如图 3-8 所示。

使用万利 Insight SE-52 仿真器时，需配合万利公司的 MedWin 集成开发系统。

① 购买 Insight SE-52 仿真器时，会随机赠送 MedWin 集成开发系统光盘(也可从万利网站下载)，将光盘插入光驱，按照提示要求即可安装 MedWin。安装好后，在桌面上生成 MedWin 图标。

② 拆下实验板上的 CPU，将仿真器的仿真头插入到 CPU 的位置，注意不要插反，并连接好仿真头与实验开发板上的地线。最后，将 SE-52 连接到 PC 机的并口上，连接示意图如图 3-9 所示。

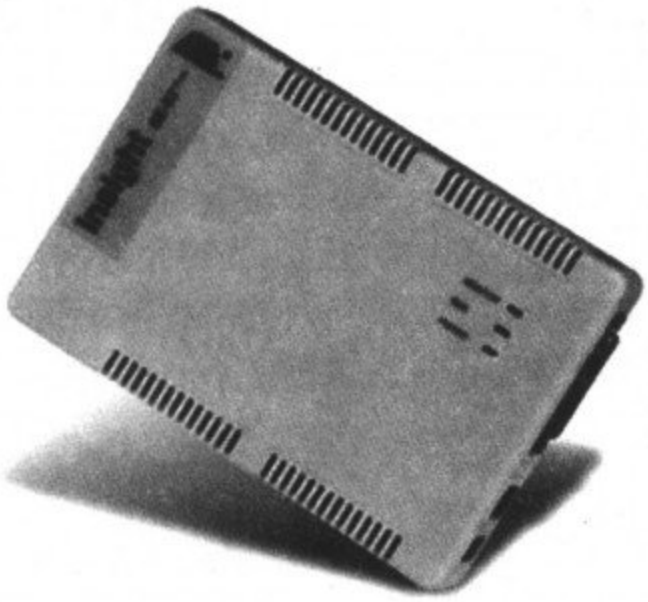


图 3-8　Insight SE-52 仿真器的外形

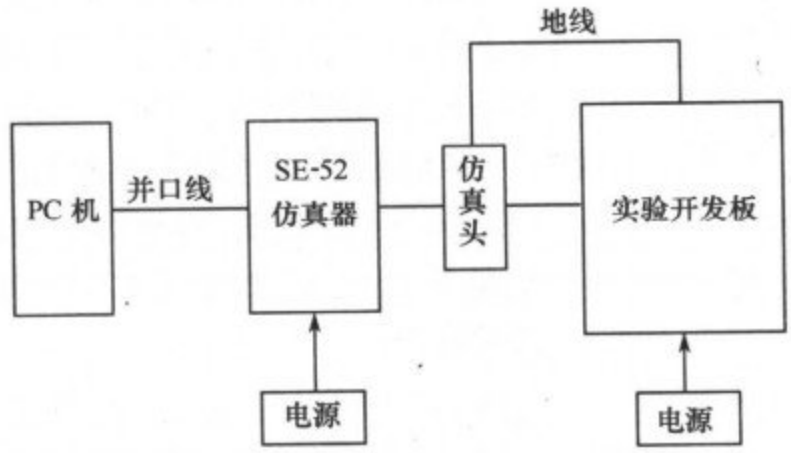


图 3-9　仿真器的连接示意图

③ 点击桌面上的 MedWin 图标，启动 MedWin 开发系统，启动后，出现如图 3-10 所示的对话框。

因为要做实时在线仿真，所以，出现的对话框中选择“仿真器”。

④ 使用 MedWin 集成开发环境，配合 Insight SE-52 仿真器进行在线仿真时，必须对仿真器产品进行注册，点击菜单“帮助”→“注册仿真器”，在出现的对话框中输入正确的注册码后，即可注册成功。

关于文件的编译与调试方法请参考随机附送的说明书，这里不作详细介绍。

二、MON51 仿真器

MON51 是一款利用 Keil Cx51 的 IDE 集成开发环境作为仿真环境的廉价仿真器，它体积小巧，价格低廉，特别适合使用 Keil 软件的用户使用。

这种仿真器的仿真 CPU 使用 SST 公司的 SST89C58 或 SST89C54(其他相容的芯片

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

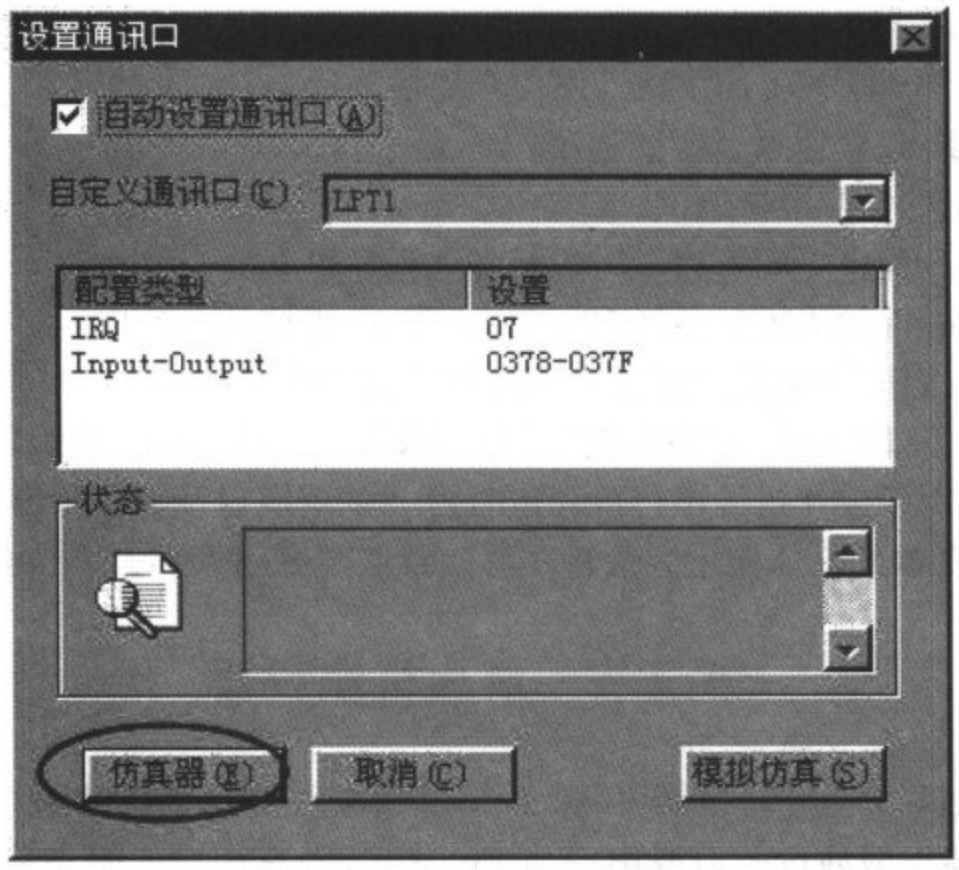


图 3-10　端口选择对话框

也可)。SST89C58 片内带有 32KB 的 Flash ROM,仿真程序时可以使用的地址是 0000H～6FFFH,总共 28KB,对于一般的应用开发已足够使用了。7000H～7FFFH 这 4KB 被仿真器监控程序占用,监控程序用于和上位机软件 Keli 进行联络,7000H～7FFFH 用户不能使用,如果强行使用会造成仿真器监控程序覆盖修改,甚至导致仿真器不能正常工作,此时需要用可以写 SST89C58 RB1 位的编程器重新写入监控程序。不过因为在开发过程中做了特殊的烧录设置,用户的外部程序就不能强行使用和存取7000H～7FFFH,这样就能够保证仿真器芯片内部的监控程序不被修改。

MON51 仿真器的外形如图 3-11 所示。

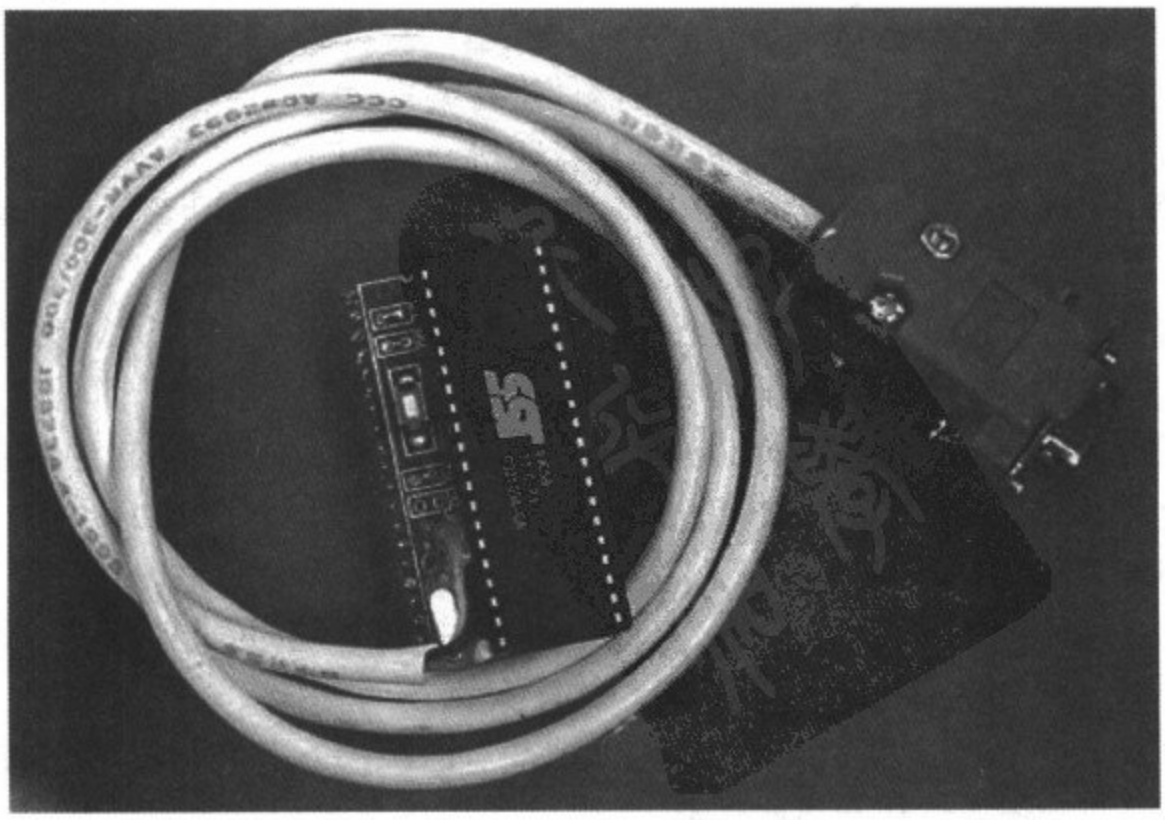


图 3-11　MON51 仿真器的外形

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

MON 51 仿真器的主要性能如下：

① 直接支持 Keil Cx51 的集成开发仿真环境。

② 可单步、断点、全速运行调试程序。

③ 支持汇编、C 语言混合调试。

④ 片内 28KB 程序空间可以随时进行在线程序更新。

⑤ 可以仿真标准的 89C51、89C52、89C58 等 51 内核的单片机。

⑥ 带有复位按钮，可避免系统复位不良而引起的故障。

⑦ 仿真频率 6MHz～33 MHz 晶振可选，系统配置 11.0592 MHz。

⑧ 仿真器占用了单片机的串行口和定时器 2 的资源以及部分程序空间。

MON 51 仿真器的具体使用方法将在第四章结合 Keil 软件进行介绍。

第三节　单片机编程器

仿真调试通过后，使用编程器可将 .hex 目标文件烧写到单片机（或外接的可编程 ROM）中。编程器一般通过并口或串口与 PC 连接，具有相应的服务程序；在连接好 PC 与编程器后运行其服务程序，在服务程序中先选择所要编程的单片机型号，再调入 .hex 目标文件，编程器就将目标文件烧写到单片机中。烧写后，再将单片机插入到实验板上相应的插座上，则单片机可将程序的功能通过实验板表现出来。

目前，常用的编程器主要有两种：一种是通用编程器（如 RF-810 编程器）；另一种是下载型编程器。

一、RF-810 编程器

RF-810 编程器是一款性能较好的编程器，工作相当稳定，性价比也很好。RF-810 编程器及其配件外观如图 3-12 所示。

RF-810 编程器性能特点如下：

① 可对 100 余厂家的 1000 多种常用器件进行编程、测试。

② 采用 40 脚锁紧插座，与计算机并口（打印机）联机工作。

③ 可自行调整烧录电压的参数，具有芯片损坏、插反检测功能，可以有效地保护芯片。

④ 支持 EPROM、EEPROM、FLASHROM、串行 EEPROM、PLD、MPU 编程功能。

⑤ 支持 TTL74/54、CMOS40/45、SRAM 测试功能。

RF-810 编程器配备全中文 Windows 驱动软件。软件功能齐备，对芯片的编程不需要人工干预，软件用户界面易学，使用相当方便。

1. RF-810 编程器的安装

RF-810 编程器套件包括：RF-810 编程器主机、并口电缆、并口匹配器、AC/DC 电源适配器等。RF-810 的安装包括硬件安装和软件安装。

硬件安装时，首先关闭计算机的电源，先把并口匹配器安装在计算机的打印口上，然后将联机电缆的一端连接在并口匹配器上，另一端连接在编程器主机的接口上。接通交流电源，打开编程器的电源开关，编程器主机上的电源灯亮（红灯），表示电源开始供电。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

图 3-12　RF-810 编程器及其配件外观

RF-810 编程器的软件安装也很简单，和普通软件的安装方法相同，下载最新版的软件后，运行 setup. exe 安装程序即可。软件安装完毕后，自动在桌面上形成 RF-810 编程器的图标。

2. RF-810 编程软件的功能

软件和硬件安装完毕后，点击 RF-810 编程器的图标，进入图 3-13 所示的主菜单。

主菜单包括五大部分：功能项菜单栏、工具栏、工作区、信息栏和状态栏。系统把常用的功能做成了快捷方式，放在工具栏中，点击这些图标，就可以直接进行该操作。这些快捷方式有：

：选择与芯片相对应的厂家、类型、型号、容量等。

：对缓存区内容进行修改、浏览操作。

：按擦除、查空、编程、校验等操作顺序自动完成对器件的全部操作过程。

：检查器件是否处于空白状态。

：对于可擦除的器件内容进行删除。

：把缓存区内的数据写入到芯片内并进行校验。

：把器件内容读入到缓存区。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

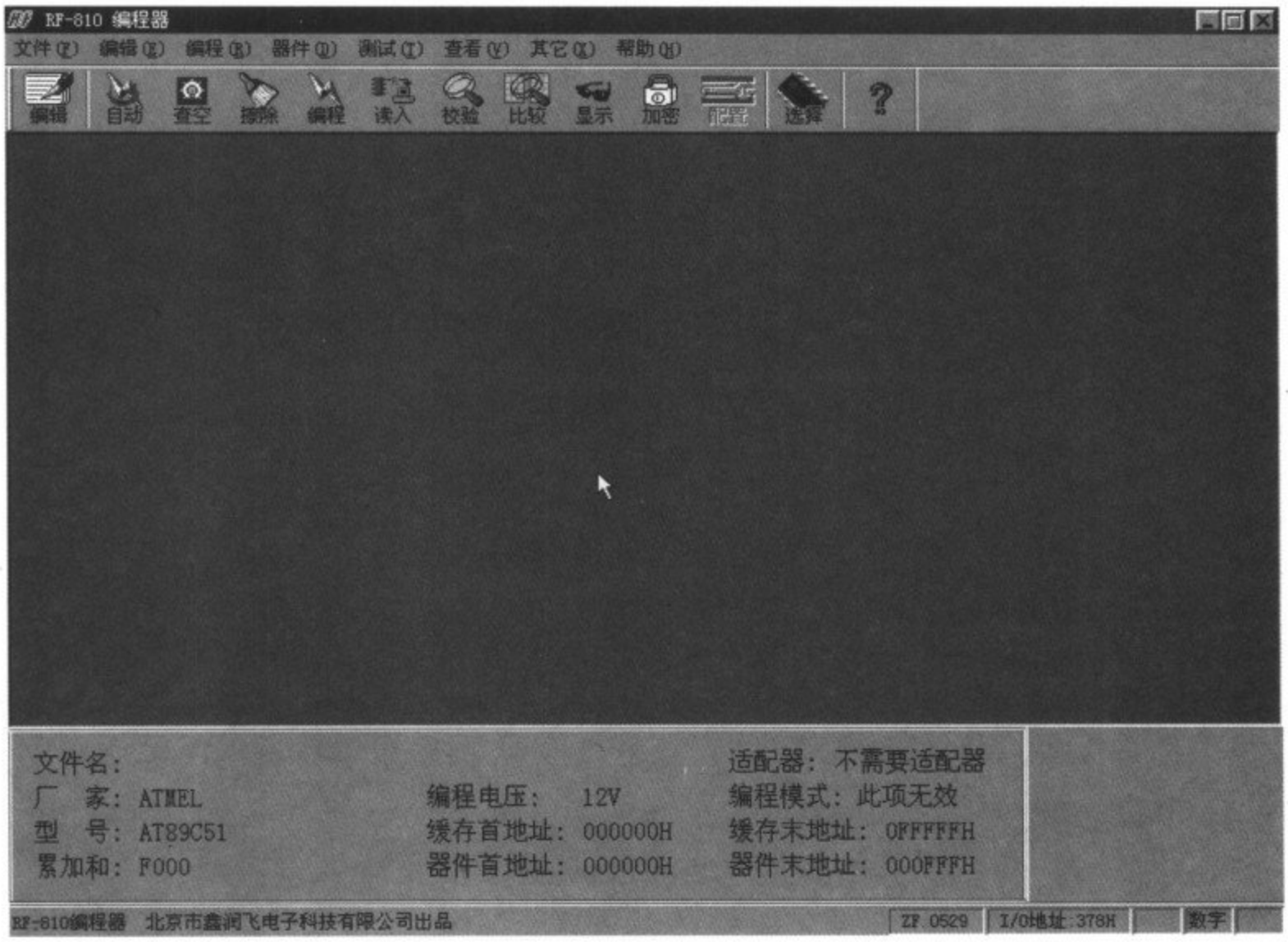


图 3-13 RF-810 编程软件的主菜单

校验:校对器件内容和缓存区内容是否一致,并列出有差异的第一个单元的地址。

比较:逐单元比较器件内容和缓存区内容有无差异,并将有差异的单元列表显示。

显示:将器件内容显示在屏幕上供浏览。

3. RF-810 编程软件的使用

下面以读、写一块 AT89C51 单片机芯片为例,介绍 RF-810 编程软件的使用方法。

把待读写的 AT89C51 插到编程器上的 ZIP 插座上,在 ZIP 插座旁有几块芯片重叠的示意图,它告诉你芯片插入的正确方向。在半导体芯片上,其陶瓷封装的一边有一个缺口,表示芯片管脚的排列方向,插入时,要注意芯片上缺口的位置和编程器上 ZIP 插座旁芯片方向示意图对应。

(1)备份操作

为了确保操作万无一失,不管进行什么操作,一般都要把芯片的内容备份下来。

在进行读、写操作前,都必须首先选择器件。点击工具栏上的选择按钮,弹出"器件选择"窗口,在 EPROM、SPROM、EEPROM、PLD、MPU/MCU 等几种类型中选择所需的芯片类型(MPU/MCU),然后进行厂家选择,在厂家选择区中找到芯片对应的厂家名称并点击(如 ATMEL)。如果芯片类型选择不当,因不同类型的芯片管脚工作电压不同,有可能烧毁芯片;选择芯片的厂家后,再进行器件选择,在器件型号选择区中找到所需的型号并点击(如 AT89C51),如图 3-14 所示。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

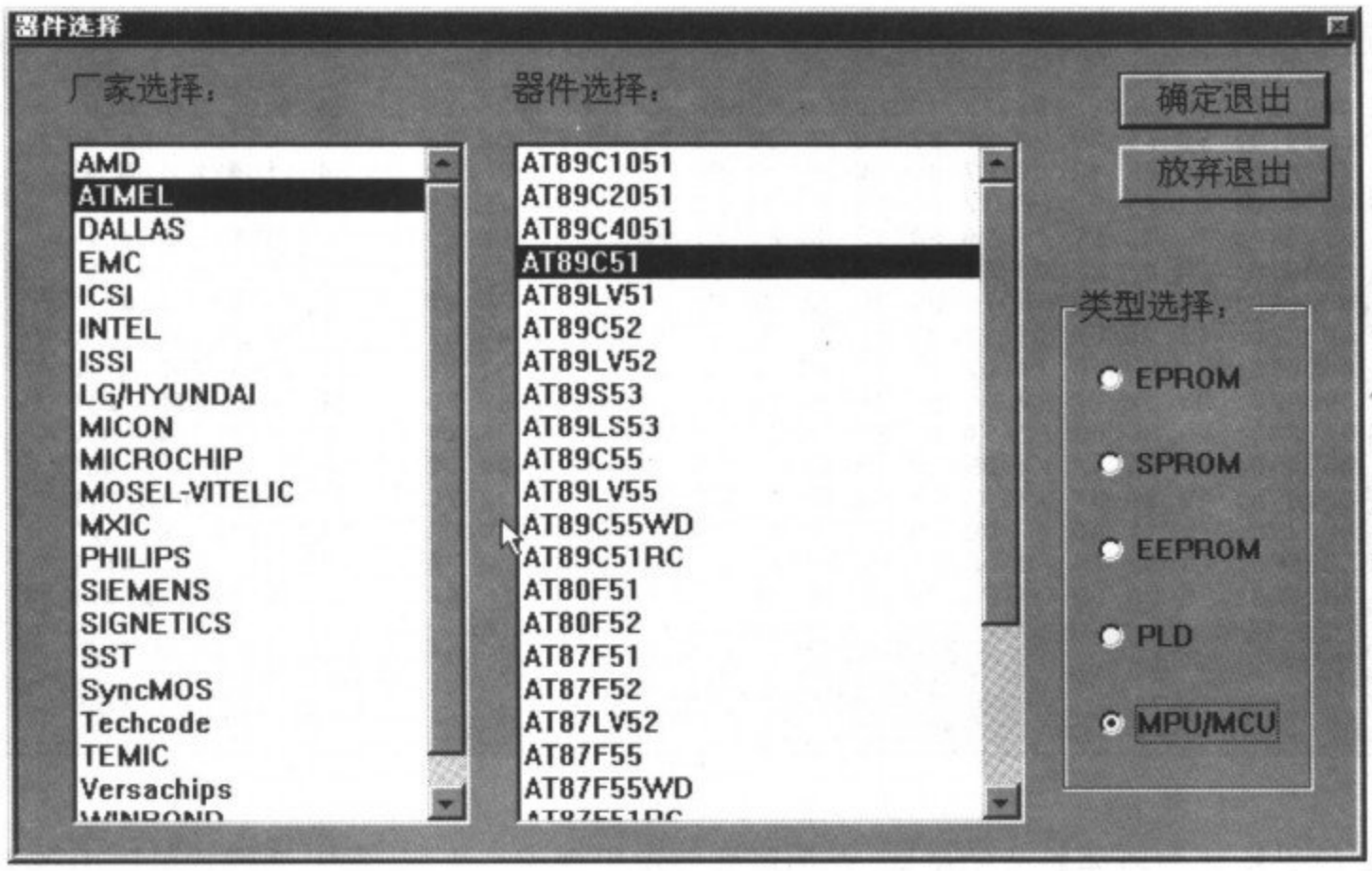


图 3-14　器件选择窗口

在全部选择完毕后，点击“确定退出”按钮完成器件选择操作。此时选定的型号作为当前型号在主菜单信息栏中列出，与其相应的其他信息也一并列出。

点击按钮，弹出“读入操作”窗口。如果刚才选择的芯片的类型和容量都正确，点击“确定”按钮，确认器件的起始地址、长度和缓冲区的起始地址后，编程器会把芯片内的数据读入 PC 机的内存中，如图 3-15 所示。

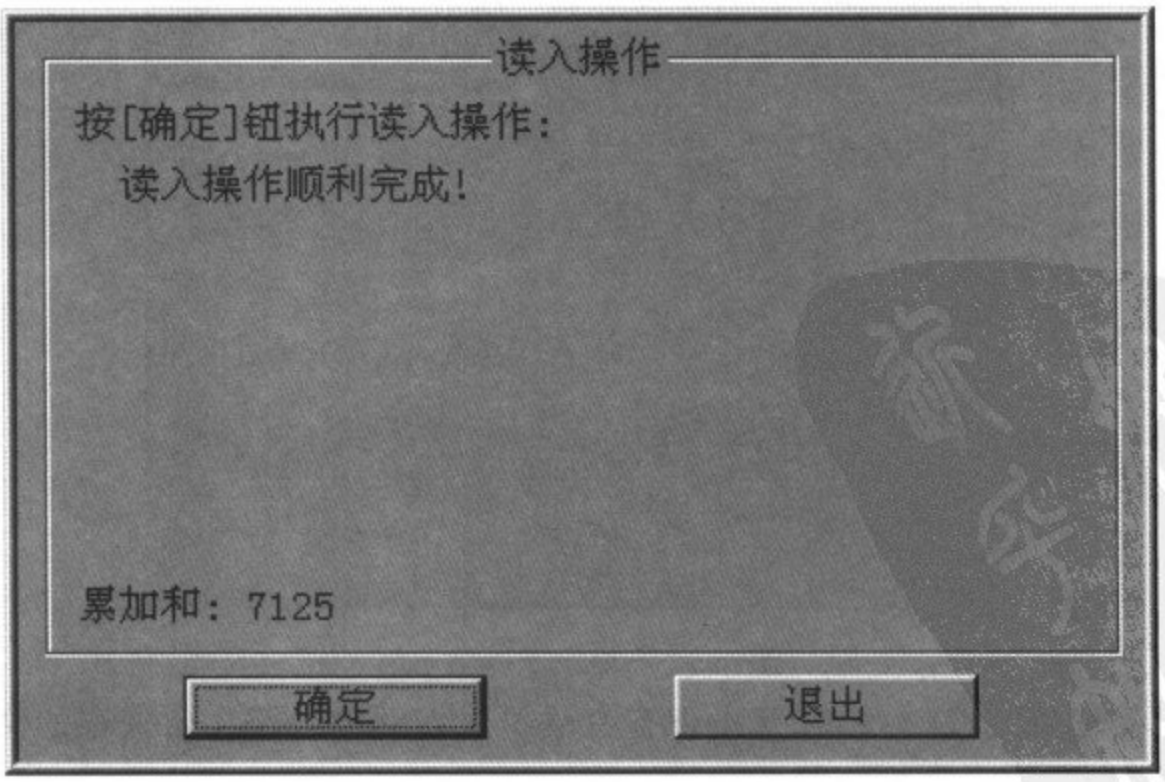


图 3-15　读入窗口

若要对缓存区内的数据进行编辑，可点击工具栏上的 按钮，出现编辑画面，如图 3-16所示。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

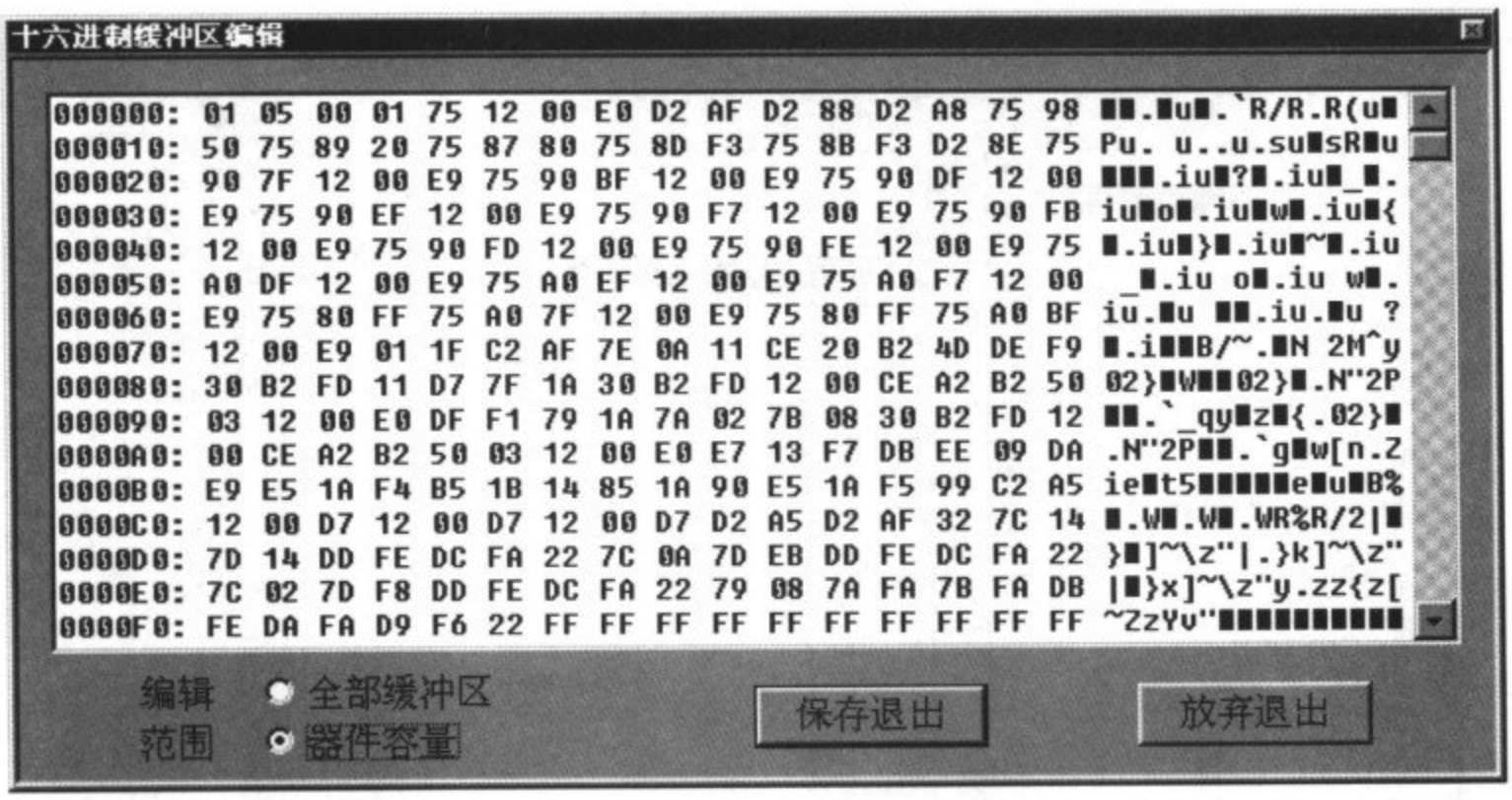


图 3-16　编辑画面

将光标移向所要修改的位置，点击鼠标左键出现闪烁光标，在闪烁位置可以用键盘输入十六进制字符串。注意，如果没有特别的需要，建议不要对内容做任何改动。

为了防止以后的操作失误，把缓存区内的数据保存成文件。选择“文件”菜单中的“保存文件”选项，点击“保存文件”操作项，弹出所要存盘的缓存区的首末地址及存盘格式窗口，如图 3-17 所示，默认的文件是二进制格式，扩展名为 bin，这里选择十六进制(Intel hex)，扩展名为 hex。

图 3-17　保存文件操作项

“确定”后，出现如图 3-18 所示的保存文件位置对话框。输入要保存的文件名后，单击“保存”按钮。

(2)写入操作

在写入目标代码之前，要先把芯片内的数据进行擦除，使其处于空白状态。点击 按钮，弹出“擦除操作”窗口，如图 3-19 所示。如果器件已正确放置，点击“确定”按钮开始

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

图 3-18　保存文件位置对话框

擦除操作。

擦除之后要进行查空，以确认当前芯片中的数据信息是否全部为空，这时需要点击 查空 按钮，弹出“查空操作”窗口，以检查器件放置是否正确，点击“确定”按钮开始查空操作，系统开始对芯片进行空片检查。在检查过程中，滚动条不断显示检查单元的总数、百分比和字节总数，按 ESC 键可终止查空检查操作。检查完毕后，如果器件为空白，则提示“查空操作顺利完成”，如图 3-20 所示。此时，缓冲区内的数据信息应显示为“FF”，也就是没有任何数据。如果数据信息不全为“FF”，会提示“地址××××××查空错误”，表示芯片中的部分代码清除不掉，该芯片质量有问题或没有擦除干净。

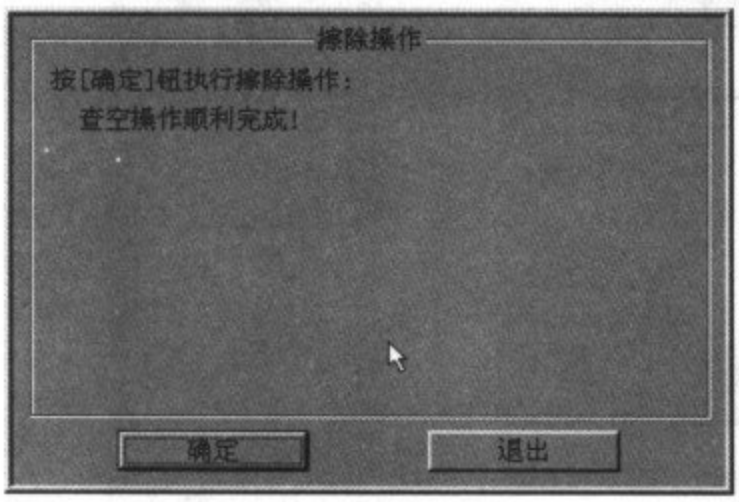


图 3-19　擦除操作窗口

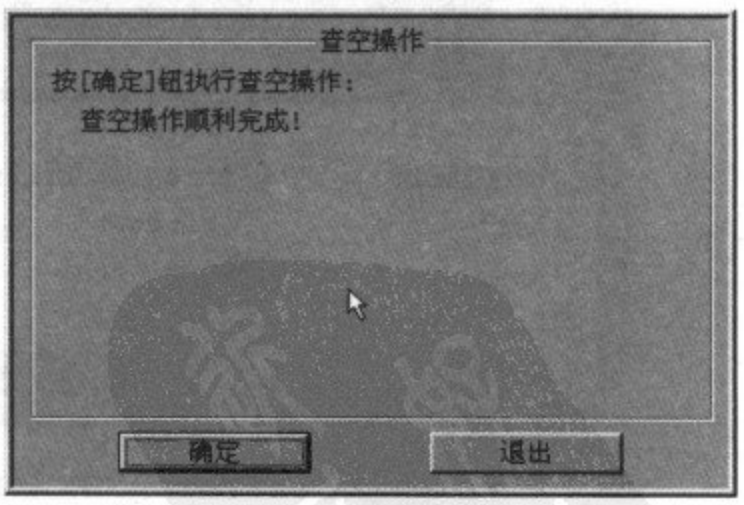


图 3-20　查空操作窗口

确认芯片已完成清空后，点击菜单栏中的文件项，在下拉菜单中选择“读入文件”操作项，弹出文件格式选择菜单，如图 3-21 所示，共有 5 种格式可以选择，分别是二进制格式(Binary)、Intel hex、Motorola hexEMCCDS、JEDEC 格式。

点击所需的格式(Intel hex)，确认装入数据缓冲区的首址(默认地址为 000000，一般不需修改)和读入方式(默认方式为全部读入)，如图 3-22 所示。

按“确定”钮，出现如图 3-23 所示读入文件对话框，单击“打开”按钮，从硬盘上把以前保存的文件 one. hex 读入到内存中。文件调入缓存区后，计算机根据该文件调入缓存区

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

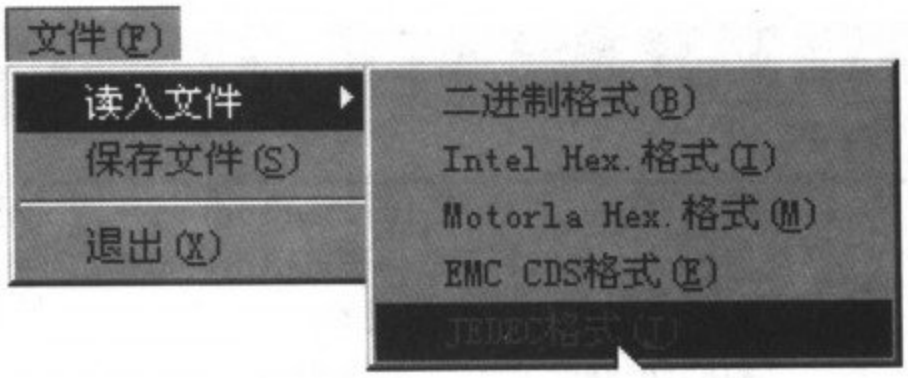


图 3-21　文件格式选择菜单

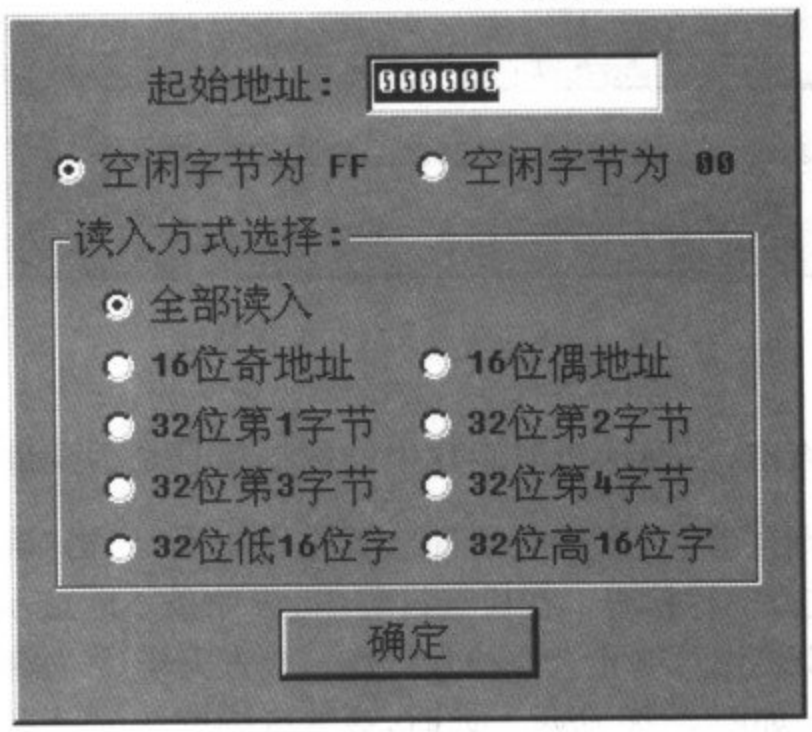


图 3-22　读入方式

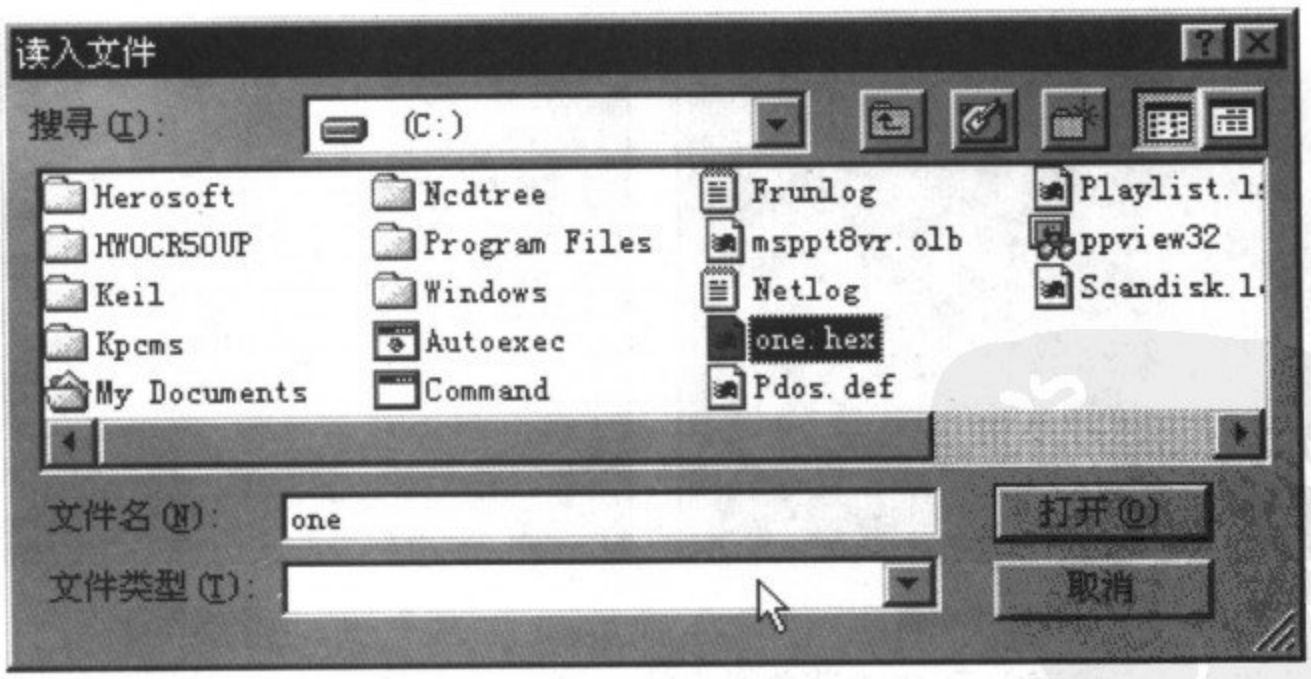


图 3-23　读入文件对话框

的起始地址和文件长度，计算出文件在缓存区中的安放位置并提示。

点击 按钮，弹出“编程操作”窗口。点击“确定”按钮开始编程操作，系统确认缓存区的起始地址、长度和器件的起始地址后，把当前内存中的数据写入芯片中，如图 3-24 所示。屏幕提示编程进度，编程操作结束后，自动进行校验操作。如某一操作失败，将提示错误信息。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

写入完毕后，为了保证数据的完整性和检验写入操作是否正确，可再点击 按钮，弹出“校验操作”窗口，点击“确定”按钮开始校验操作。RF-810 编程器的最大的特点是可自行调整烧录电压的参数及作 VCC、VCC±5%甚至最严格的 VCC±10%的验证，在校验过程中，编程器读出芯片内容和缓存区内的数据进行对比，滚动条会不断显示完成校验的数据的百分比。如数据校验通过，完全无误后，屏幕会出现“校验操作顺利完成”的字样，如图 3-25 所示。

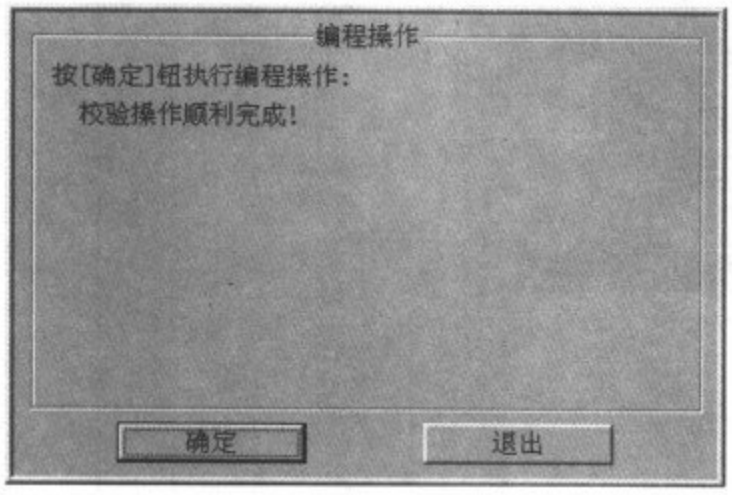


图 3-24　编程操作窗口

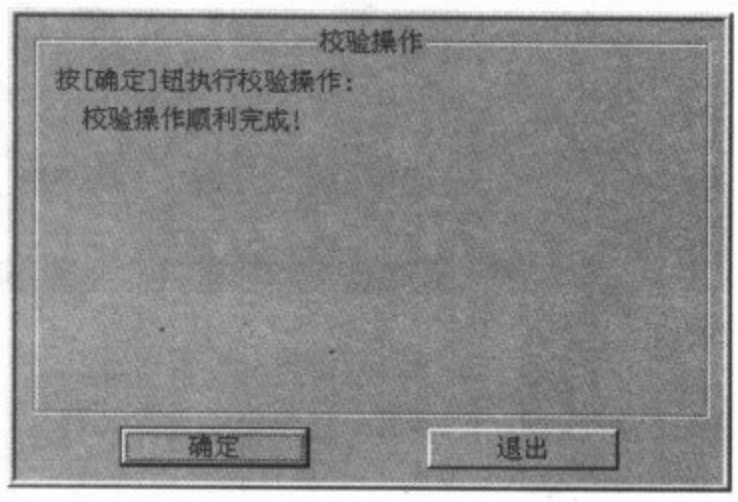


图 3-25　校验操作窗口

二、下载型编程器

ISP(In-System Programming)在系统可编程，指电路板上的空白器件可以编程写入最终用户代码，而不需要从电路板上取下器件，已经编程的器件也可以用 ISP 方式擦除或再编程。ISP 技术是未来发展方向。

ISPro 编程器全称 ISPro 下载型编程器，编程器套件如图 3-26 所示。

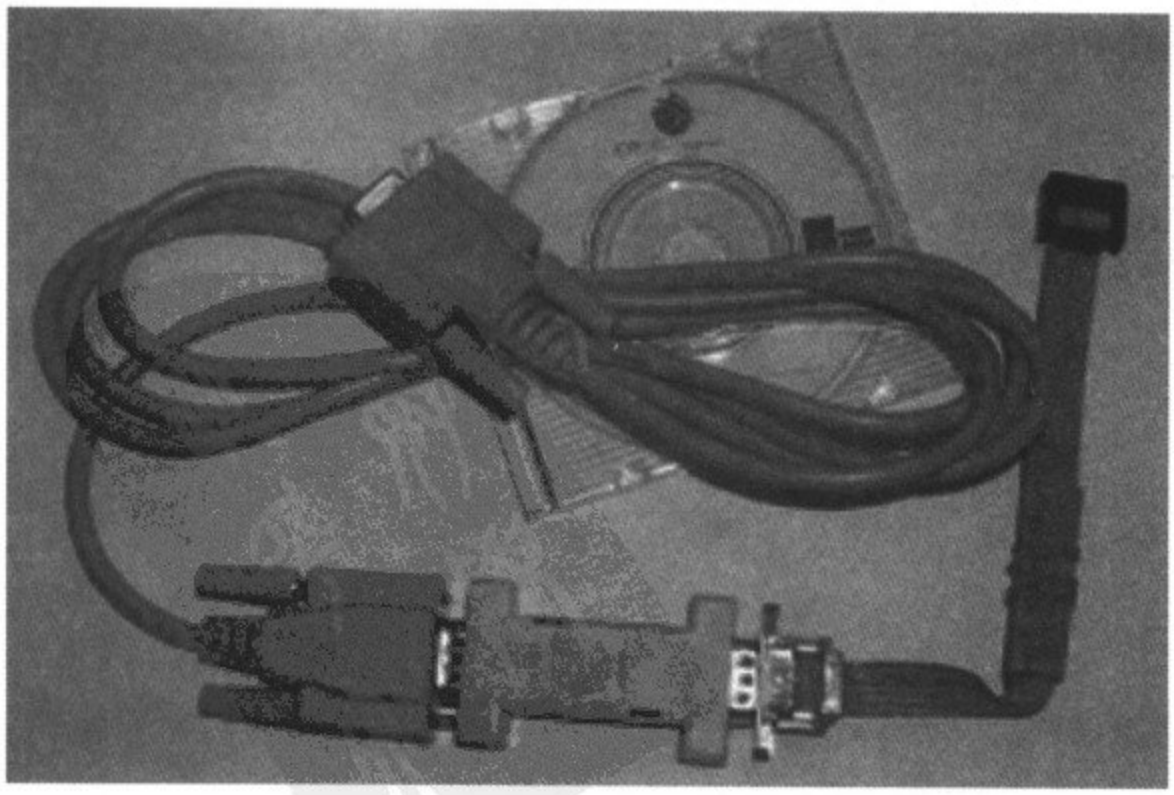


图 3-26　ISPro 套件

购买 ISPro 编程器时，会随机赠送安装光盘。安装时，将光盘插入光驱，运行 SETUP. EXE 即可。如果提醒您系统文件过旧，则需要重新启动计算机。重启后继续安装。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

安装后，在桌面上建立一个“ISPro. exe 下载型编程器”图标，双击该图标，即可启动编程软件。启动后的画面如图 3-27 所示。

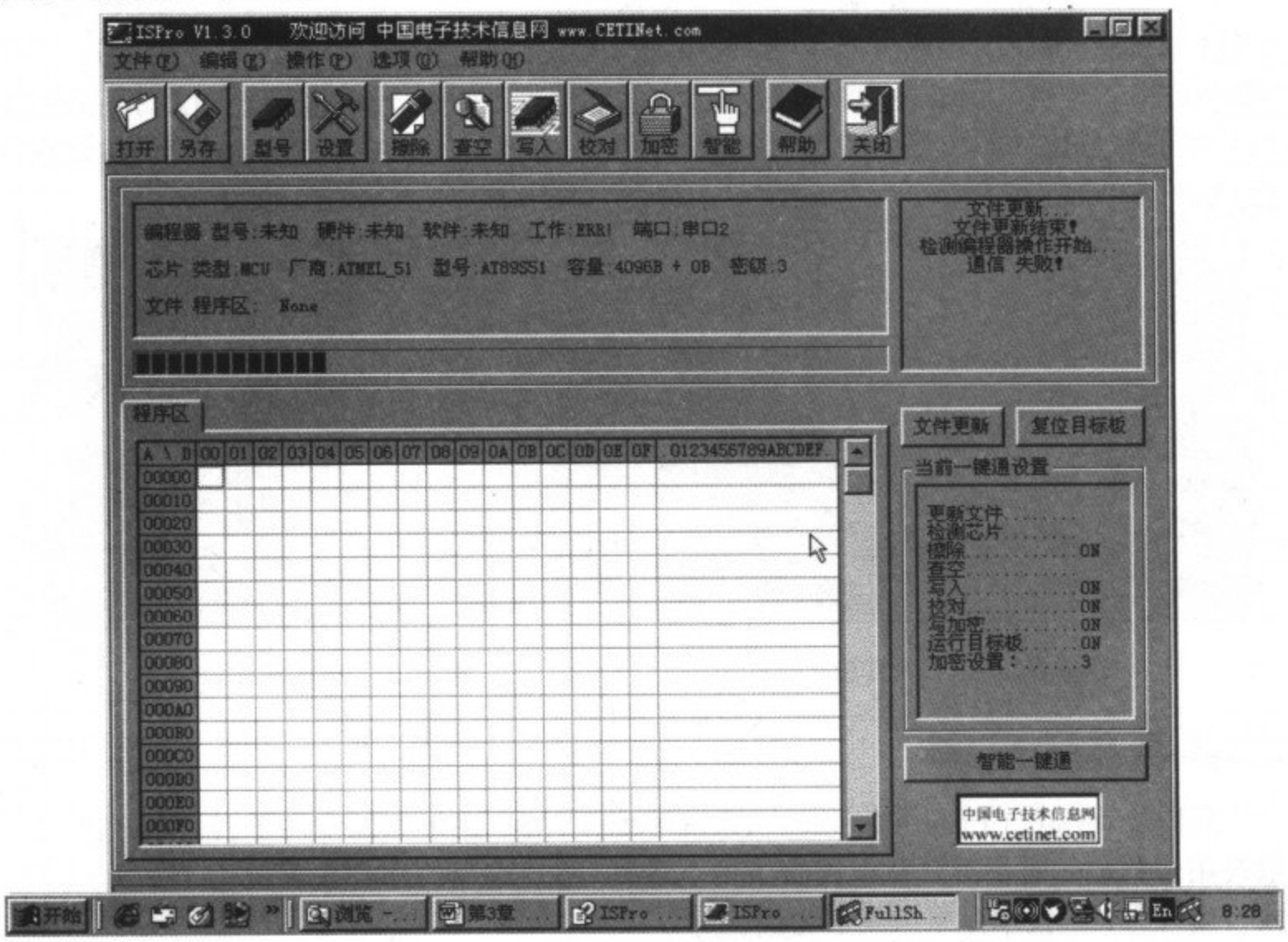


图 3-27　启动后的画面

该软件和 RF-810 软件的使用方法基本相同，这里不再具体介绍。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第四章　Keil Cx51 软件的使用

Keil Cx51 是德国开发的一个 51 单片机开发软件平台，最开始只是一个支持 C 语言和汇编语言的编译器软件。后来随着开发人员的不断努力以及版本的不断升级，它已经成为一个重要的单片机开发平台。Keil Cx51 的界面和常用的微软 VC++的界面相似，界面友好，易学易用；在调试程序、软件仿真方面也有很强大的功能。因此，很多开发 51 系列单片机应用的工程师或普通的单片机爱好者都十分喜欢它。学习程序设计语言和某种程序软件，最好的方法是直接操作实践。本章主要通过简单的实例，引导大家学习 Keil Cx51 软件的基本使用方法和调试技巧。

第一节　Keil Cx51 简介

一、Keil Cx51 的特点

Keil Cx51 是 Keil Software 公司出品的 51 系列兼容单片机 C 语言软件开发系统，符合 ANSI 标准，生成的程序代码运行速度极高，所需要的存储器空间极小，完全可以和汇编语言相媲美。

Keil Software 公司目前已经推出了 V7.0 以上版本的 Cx51 编译器，为 8051 单片机软件开发提供了全新的 C 语言环境，同时保留了汇编代码高效、快速的特点。Cx51 已被完全集成到一个功能强大的全新集成开发环境 uVision2 中，其中包括项目(Project)管理器、Cx51 编译器、Ax51 宏汇编器、BL51/Lx51 连接定位器、RTX51 实时操作系统、仿真软件模拟器以及 Monitor51 硬件目标调试器，所有这些功能均可在 uVision2 提供的单一而灵活的开发环境中极为简便地进行操作。

在本书中，将不对 Keil Cx51 和 uVision2 两个术语做严格的区分，一般来说，都是指 uVision2 集成开发环境。

uVision2 提供了强大的项目管理功能，可以十分方便地进行结构化多模块程序设计。uVision2 内部集成源级浏览器(Browser)利用符号数据库中详细的符号信息，使用户可以快速浏览源文件，并优化用户的变量数据存储器。uVision2 内部集成器件数据库(Device Database)储存了多种不同型号单片机的片上资源信息，通过它可以自动设置 Cx51 编译器、Ax51 宏汇编器、BL51/Lx51 连接定位器及调试器。

uVision2 内部集成源程序编辑器，允许用户在编辑源程序文件时(甚至在未经编译和汇编之前)设置程序调试断点，便于在程序调试过程中快速检查和修改程序。uVision2 提供文件查找功能，能对单一文件或全部项目文件进行指定搜索。

uVision2 支持软件模拟仿真(Simulator)和用户目标板调试(Monitor51)两种工作方式，在软件模拟仿真方式下不需要任何 8051 单片机硬件即可完成用户程序仿真调试，极

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

大地提高了用户程序开发效率，在用户目标板调试方式下，利用硬件目标板中的监控程序可以直接调试目标硬件系统，使用户节省购买昂贵硬件仿真器的费用。

二、Cx51 与 ANSI C 的主要区别

Keil Software 公司的针对 51 系列的 Keil Cx51 编译器与 ANS IC 主要有以下几点不同：

① 从头文件来说，51 系列有不同的厂家、不同的系列产品，如 ATMEL 公司有 89C2051、89C51、89C52、89S8252 等系列产品。它们都是基于 51 系列的芯片，不同之处在于内部资源如定时器、中断、I/O 等数量以及功能的不同。为了实现这些功能，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。因此，Keil Cx51 系列头文件集中体现了各系列芯片的不同功能。

② 从数据类型来说，由于 8051 系列器件包含位操作空间和丰富的位操作指令，因此，嵌入式 C 与 ANSI C 相比要多一种位类型，使得它能如同汇编语言一样，灵活地进行位指令操作。

③ 从数据存储类型来说，8051 系列有片内、片外程序存储器，片内、片外数据存储器。片内程序存储器还分直接寻址区和间接寻址区，它分别对应 Code、Data、Xdata、Idata 以及根据 51 系列特点而设定的 Pdata 类型，使用不同存储器将会影响程序执行的效率。在编写 Cx51 程序时，最好指定变量的存储类型，这样有利于提高程序执行效率。

④ 从数据运算操作、程序控制语句以及函数的使用上来讲，它们几乎没有明显的不同。在函数的使用上，由于嵌入式系统的资源有限，它的编译系统不允许太多的程序嵌套。C 语言丰富的库函数给程序开发提供了很大的帮助，但它的库函数和 ANSI C 也有一些不同之处。首先从编译的角度来说，由于 51 系列是 8 位机，扩展 16 位字符不被 Cx51 所支持；其次，ANSI C 所具备的递归特性不被 Cx51 支持。

⑤ Keil Cx51 与标准 ANSI C 从库函数的方面来说有很大的不同。由于部分库函数不适合于嵌入式处理系统，因此它们被排除在 Keil Cx51 之外，如字符屏幕和图形函数。有一些库函数可以继续使用，但这些库函数都是厂家针对硬件特点做出的相应开发，它们与 ANSI C 的构成及用法都有很大不同，如 printf 和 scanf。在 ANSI C 中，这两个函数通常用于屏幕打印和接收字符，而在 Keil Cx51 中，它们则主要用于串行数据的收发。

第二节　Keil Cx51 软件的安装

Keil Cx51 是一个商业的软件，它同时也提供了学习者使用的 Eval 版本，该版本同正式版本一样，但有一定的限制，最终生成的代码不能超过 2KB，但用于学习已经足够了。读者可到 Keil Software 公司的主页下载该软件的最新版本。下面以 Keil Cx51 V7.04 版本为例，介绍如何安装 uVision2 集成开发环境。

① 双击 ek51v704 安装文件，出现如图 4-1 所示的安装画面。

② 点击 Next(下一步)，出现如图 4-2 所示的安装提示画面。

③ 点击 Next(下一步)，出现如图 4-3 所示的安装询问对话框。

④ 选择 Yes，出现选择安装目录对话框，如图 4-4 所示。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

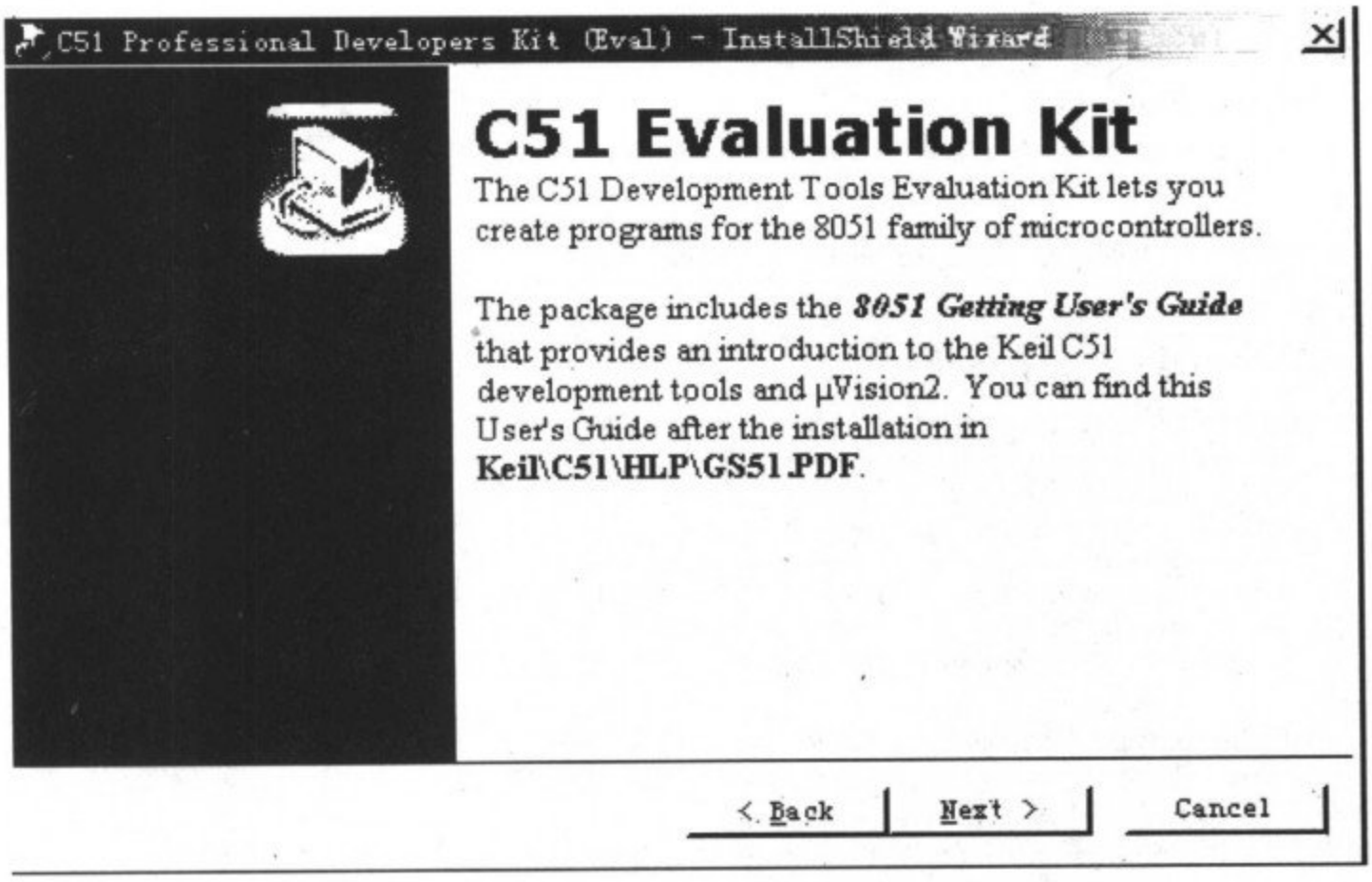


图 4-1　安装画面

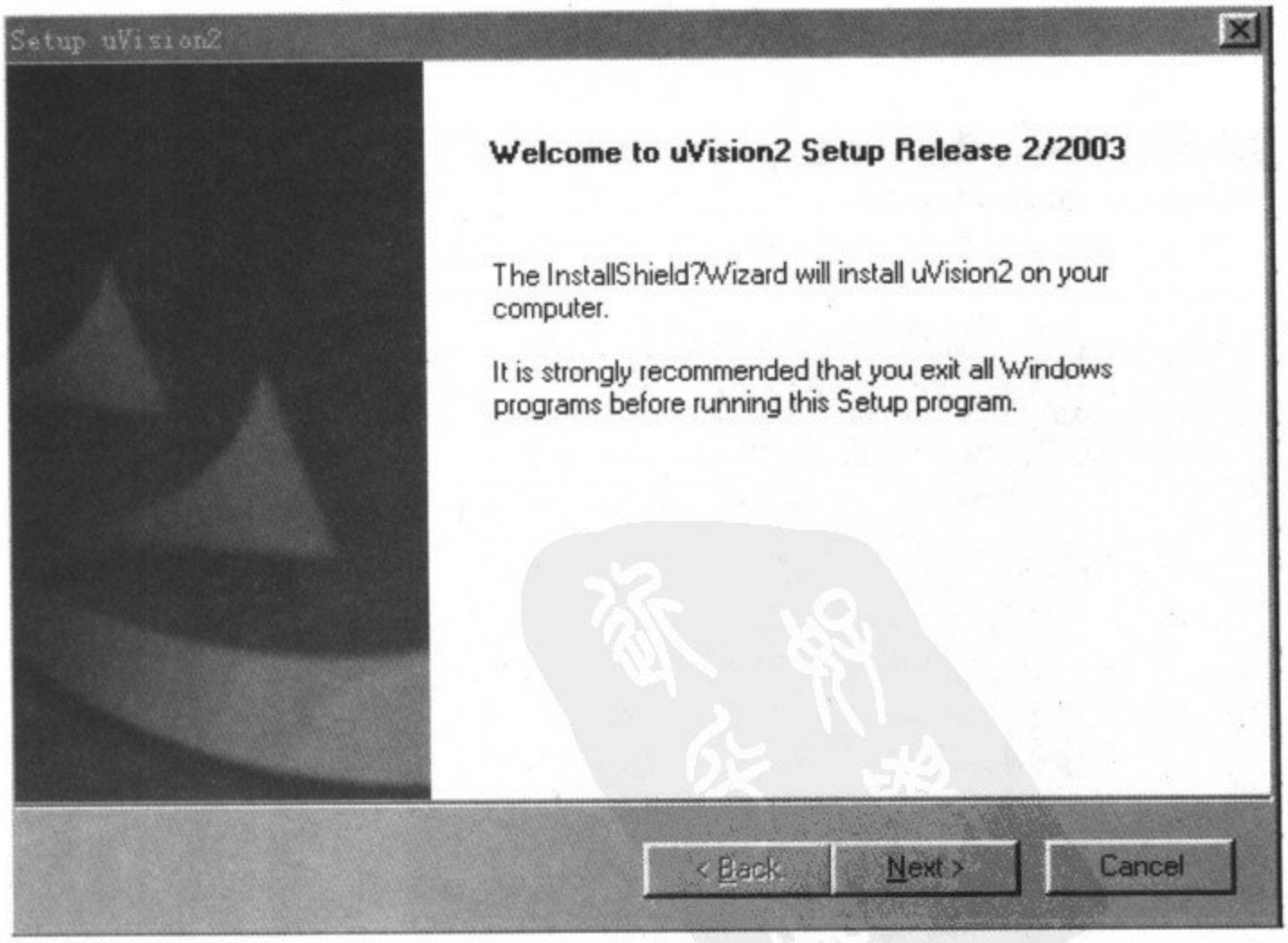


图 4-2　安装提示画面

⑤ 点击 Next(下一步),出现如图 4-5 所示的用户信息对话框。

⑥ 输入用户信息,点击 Next(下一步),程序即自动安装。安装完成后,出现如图4-6所示的安装完成对话框。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

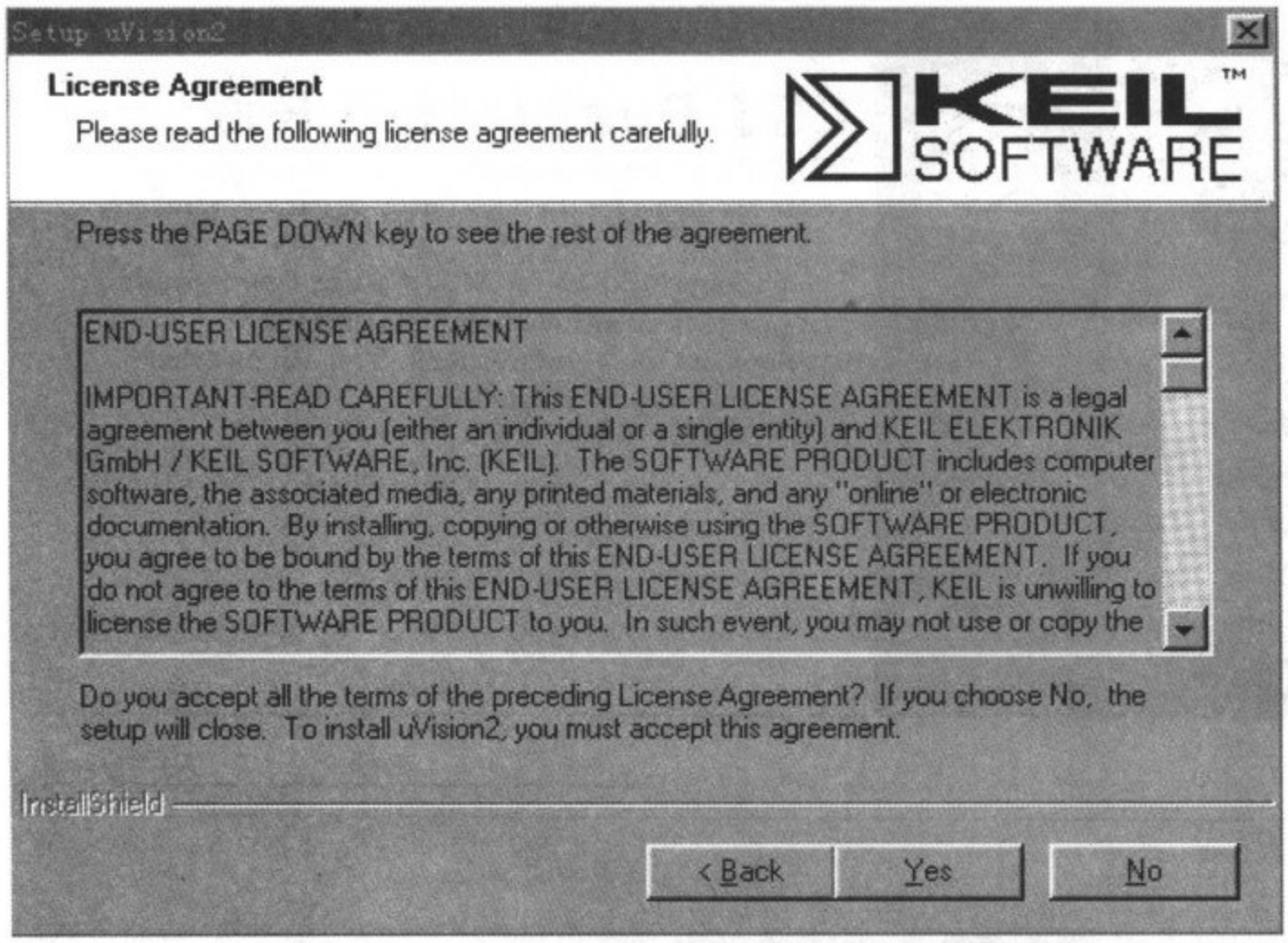


图 4-3　安装询问对话框

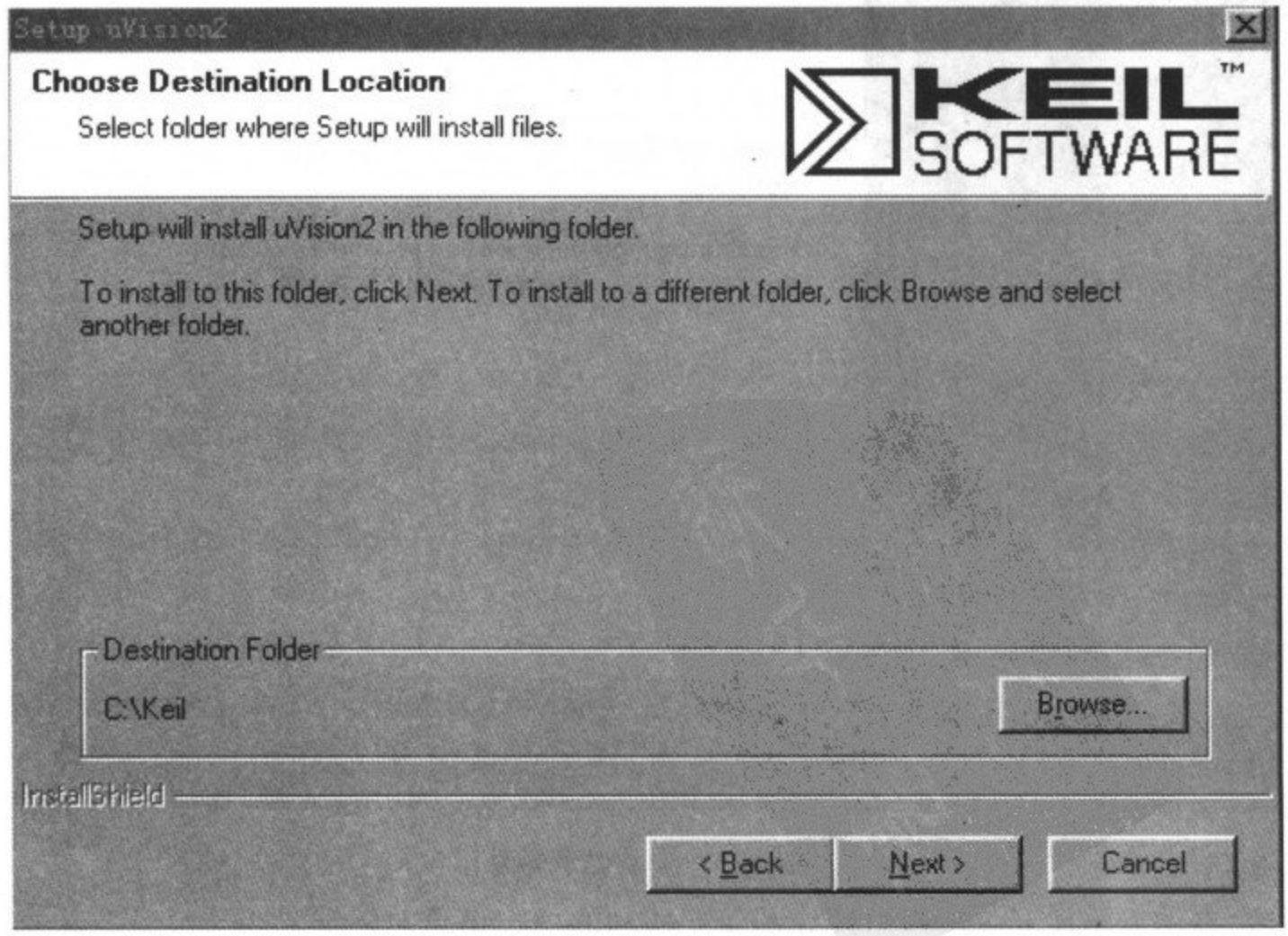


图 4-4　选择安装目录对话框

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

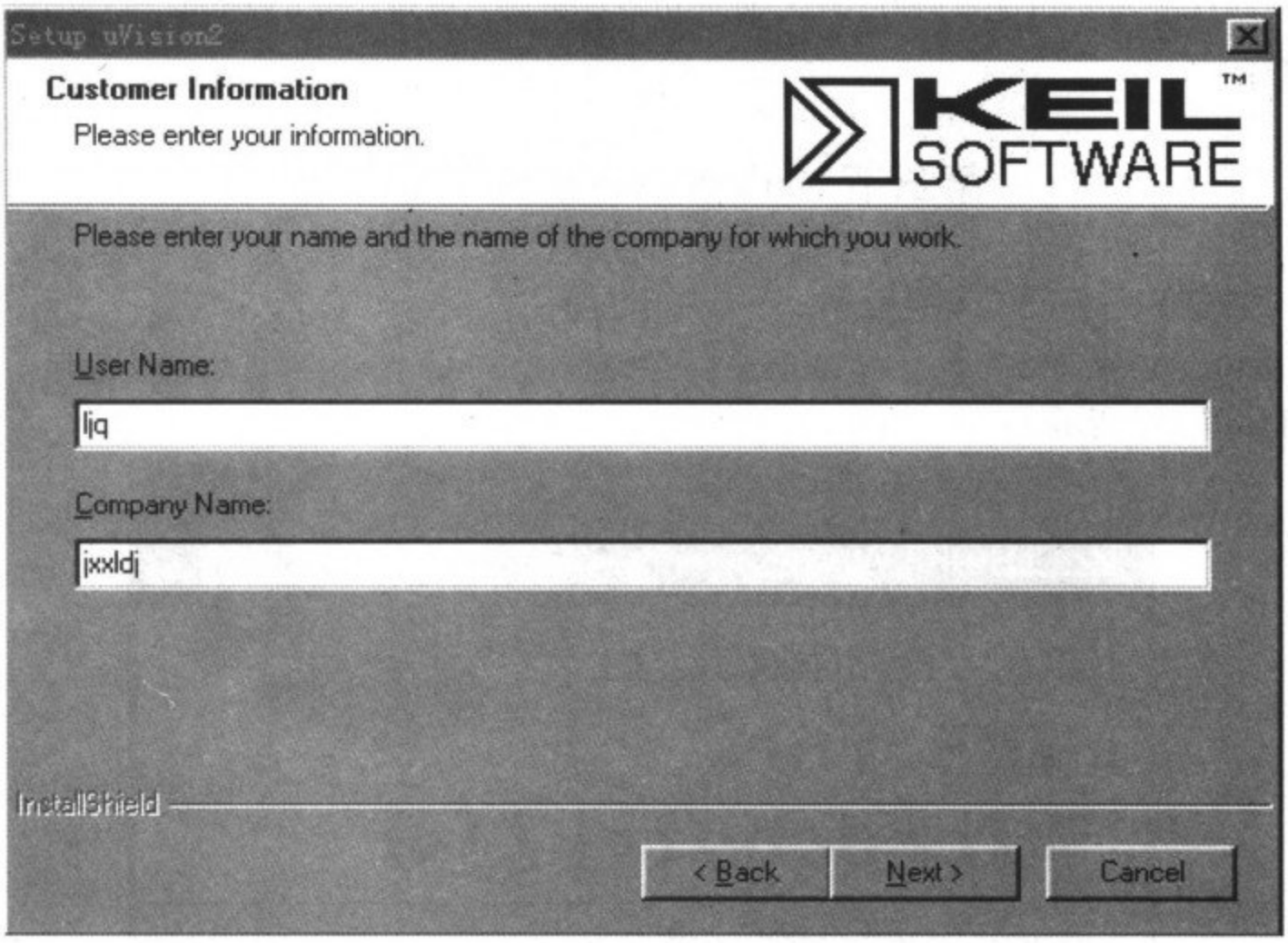


图 4-5　用户信息对话框

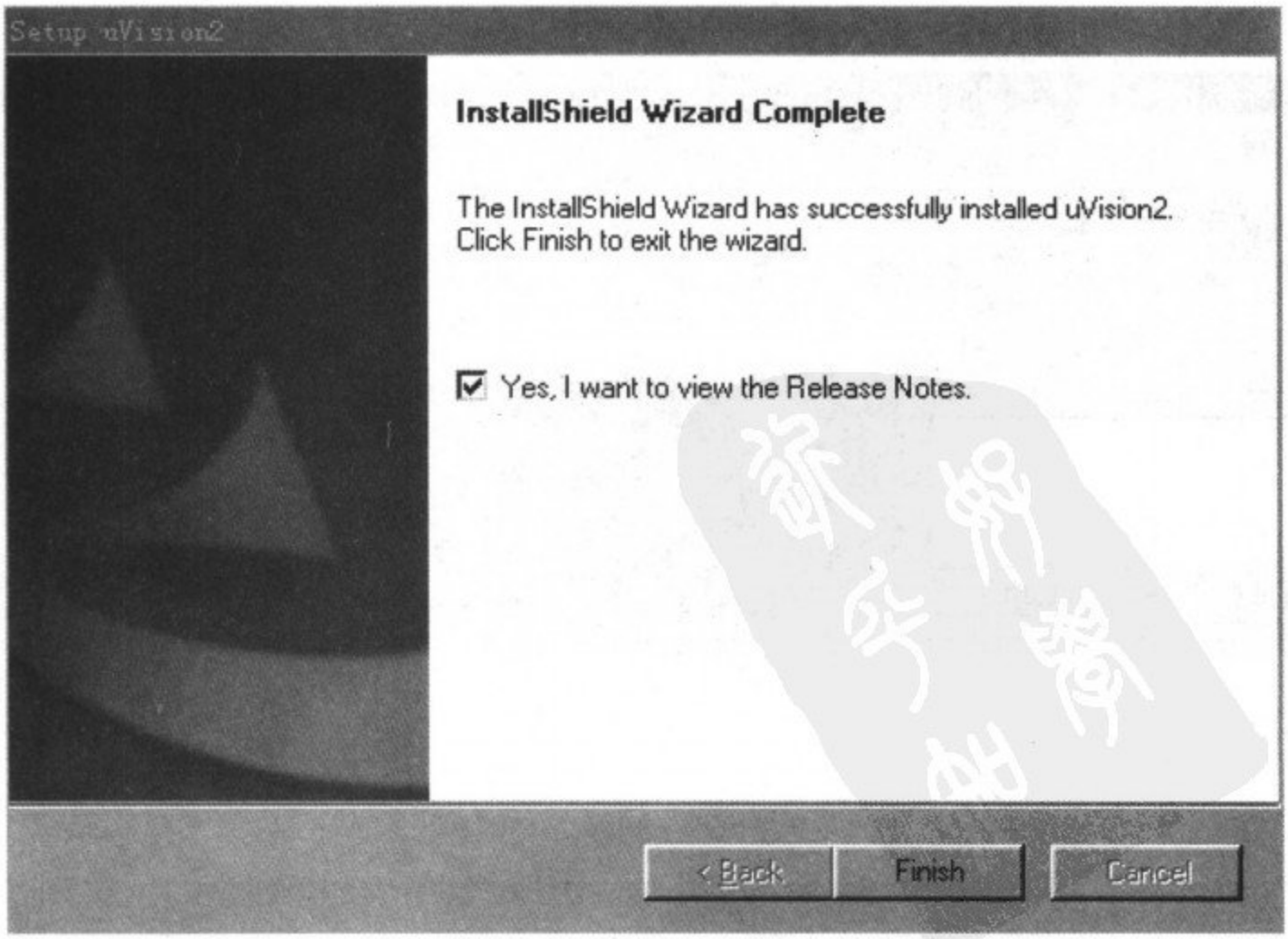


图 4-6　安装完成对话框

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

⑦ 点击 Finish(完成)加以确认,此时可以在桌面上看到 Keil uVision2 软件的快捷图标。

第三节　uVision2 集成开发环境

一、uVision2 软件的启动

uVision2 软件安装好后,在桌面上双击 Keil uVision2 图标,即可启动该软件。启动屏幕如图 4-7 所示。

图 4-7　uVision2 启动屏幕

几秒钟后,出现编辑界面,如图 4-8 所示。

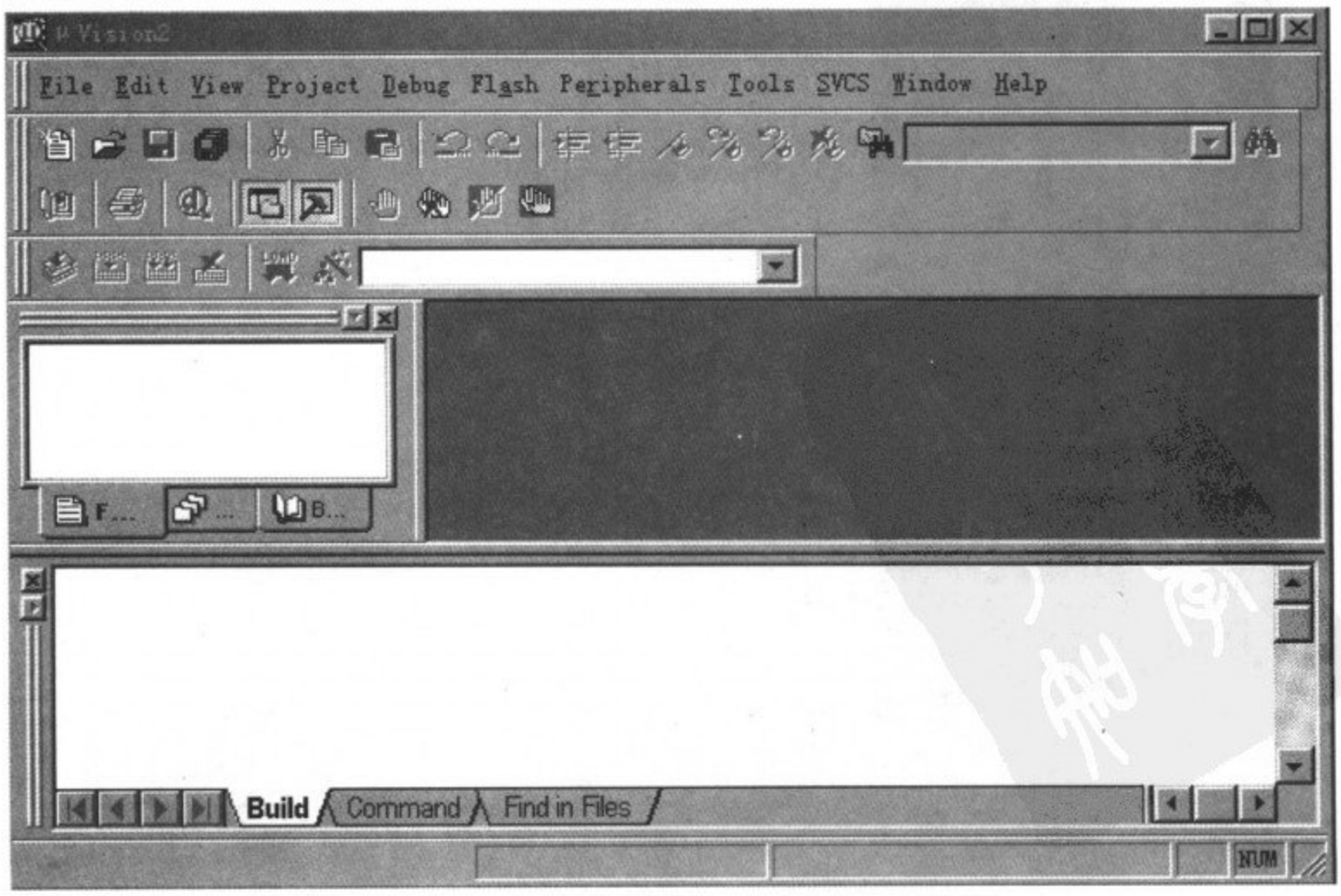


图 4-8　编辑界面

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

二、建立一个新工程

① 点击“Project”菜单，选择下拉式菜单中的“New Project”，弹出文件对话窗口，选择要保存的路径，在“文件名”中输入第一个汇编程序项目名称，这里使用“LED”，如图4-9所示。

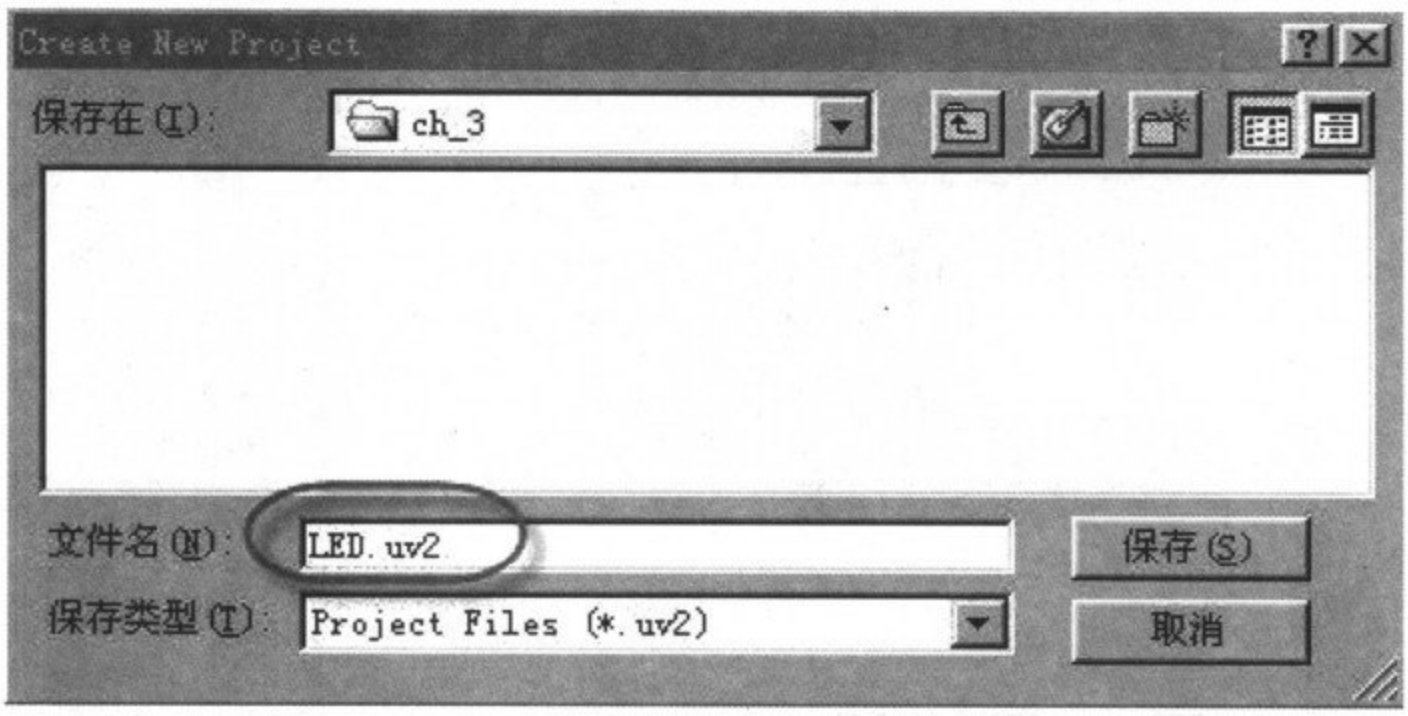


图 4-9　文件对话框窗口

保存后的文件扩展名为 uv2，这是 Keil uVision2 项目文件扩展名，以后可以直接点击此文件以打开先前做的项目。

② 点击“保存”后，这时会弹出一个对话框，要求选择单片机的型号，如图 4-10 所示。

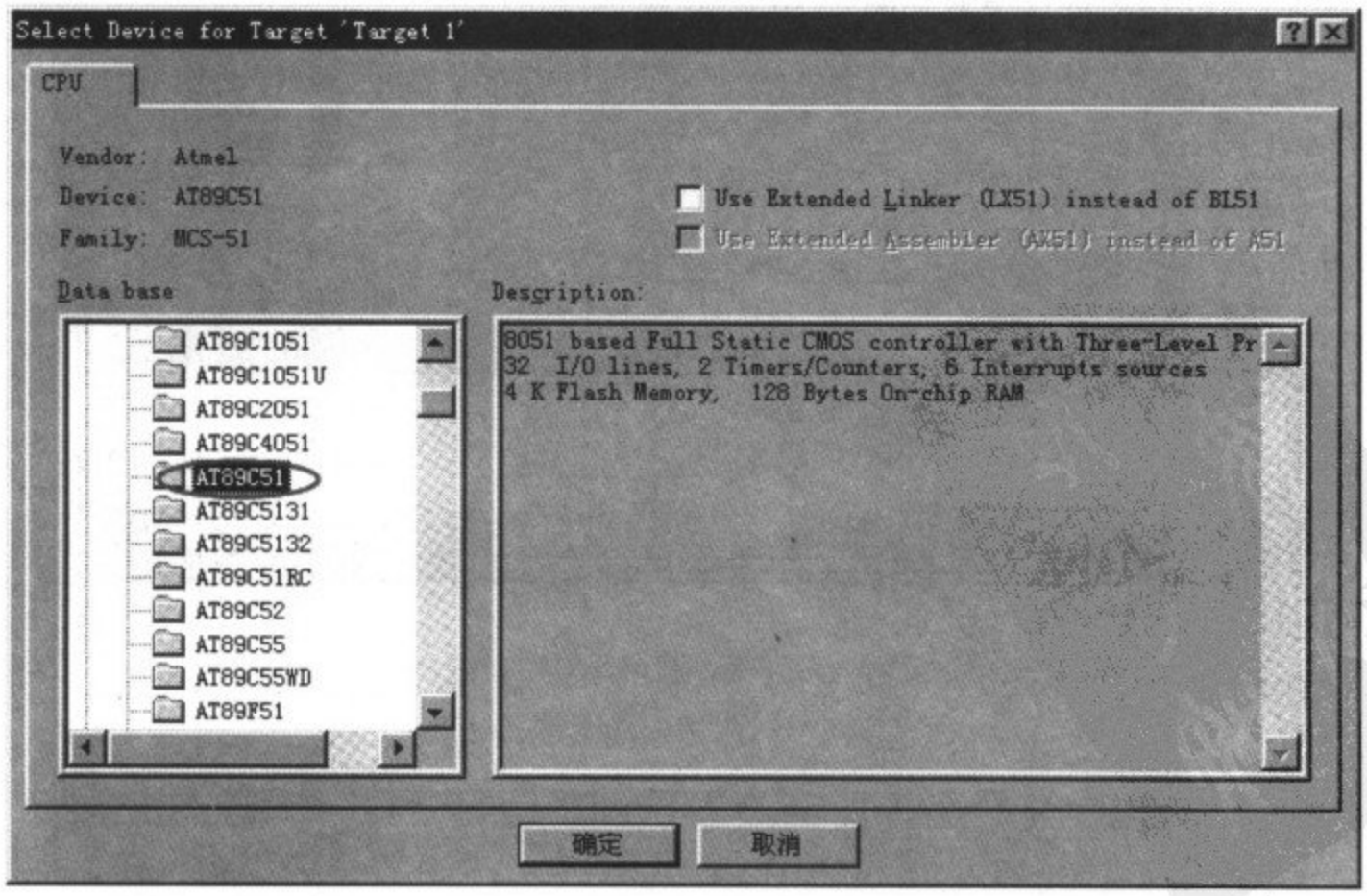


图 4-10　选择单片机窗口

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

用户可以根据所使用的单片机来选择，Keil 几乎支持所有的 51 核的单片机，这里还是以 89C51 来说明，如图 4-10 所示，选择 89C51 后，右边栏是对该单片机的基本说明，然后点击确定。

③ 完成以上步骤后，出现如图 4-11 所示的屏幕。

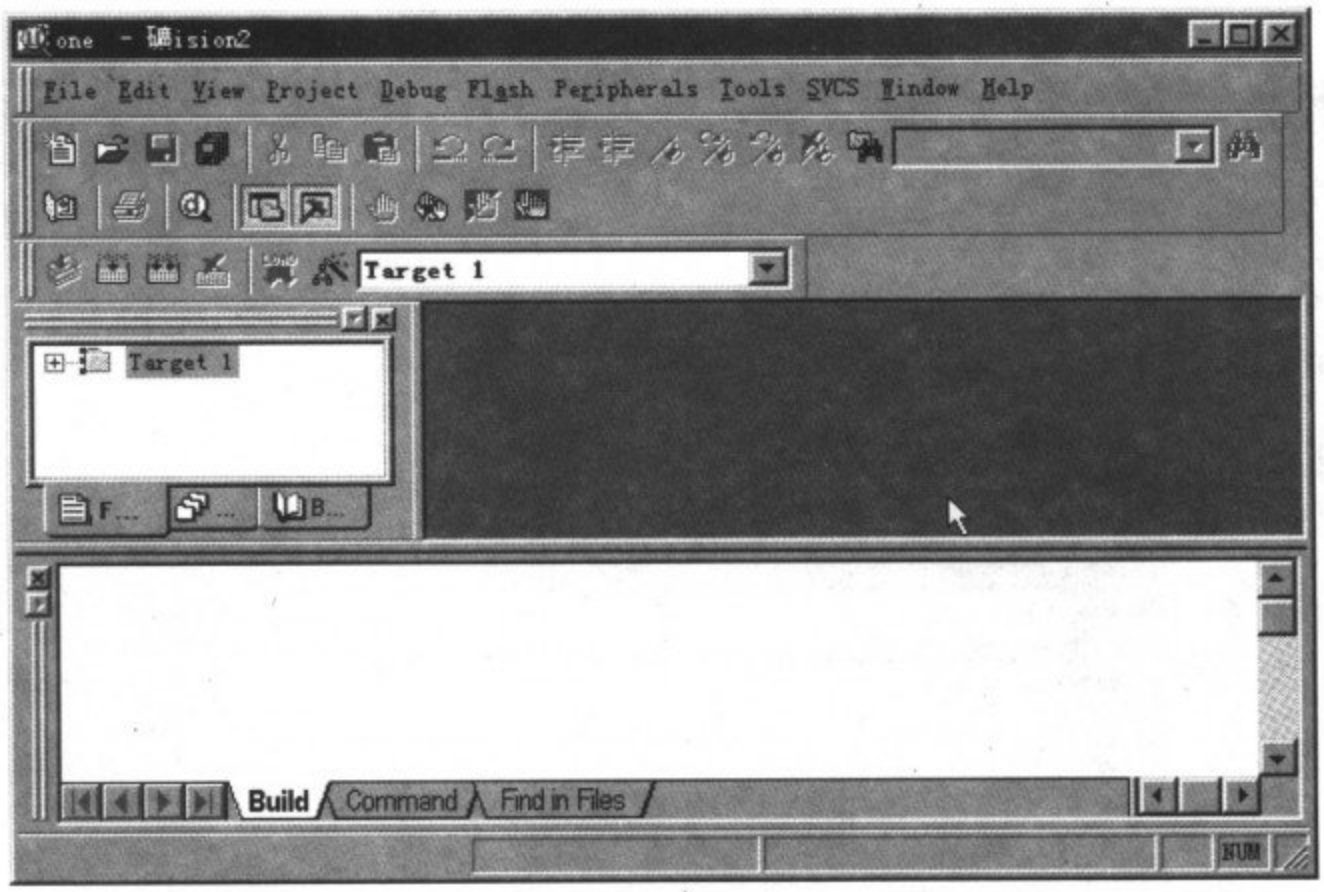


图 4-11　完成以上步骤后的屏幕

④ 现在开始编写第一个程序。在图 4-11 中，单击“File”菜单，再在下拉菜单中单击“New”选项，新建文件后屏幕如图 4-12 所示。

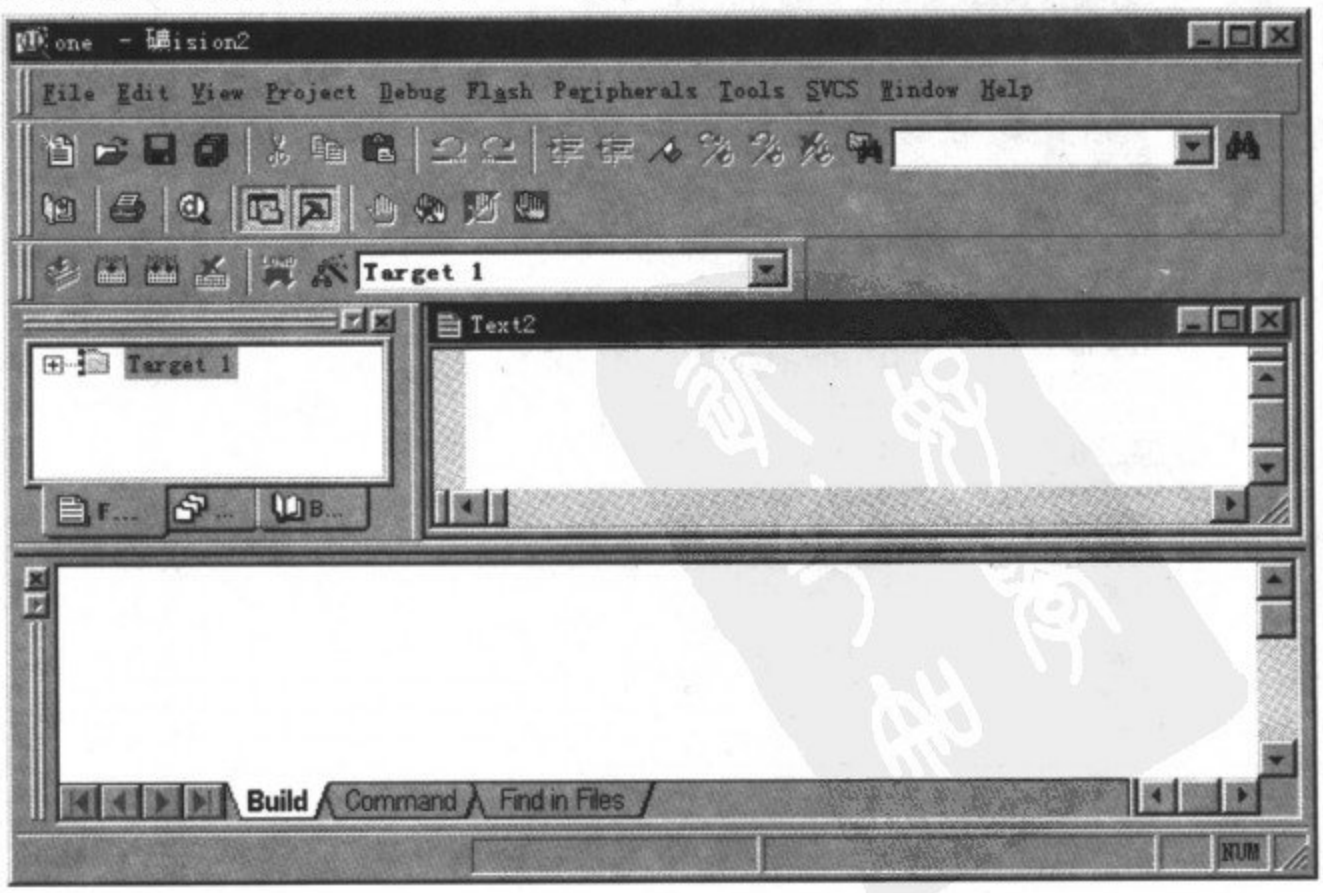


图 4-12　新建文件后的屏幕

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

⑤ 此时光标在编辑窗口里闪烁，这时可以键入用户的应用程序了，但笔者建议首先保存该空白的文件，单击菜单上的“File”，在下拉菜单中选中“Save As”选项单击，屏幕如图 4-13 所示。在“文件名”栏右侧的编辑框中，键入欲使用的文件名，同时，必须键入正确的扩展名。注意，如果用 C 语言编写程序，则扩展名为 .c；如果用汇编语言编写程序，则扩展名必须为 .asm。这里选用文件名为 LED.c，然后，单击“保存”按钮。

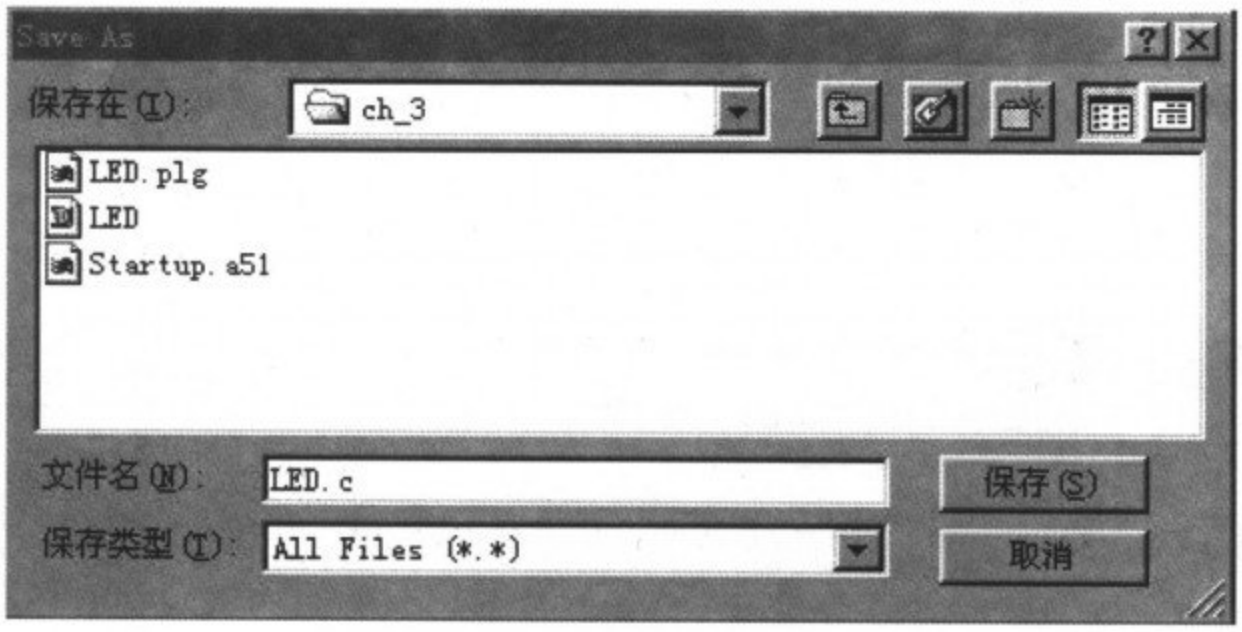


图 4-13　保存文件

⑥ 回到编辑界面后，单击“Target 1”前面的“+”号，然后在“Source Group 1”上单击右键，弹出如图 4-14 所示的菜单。

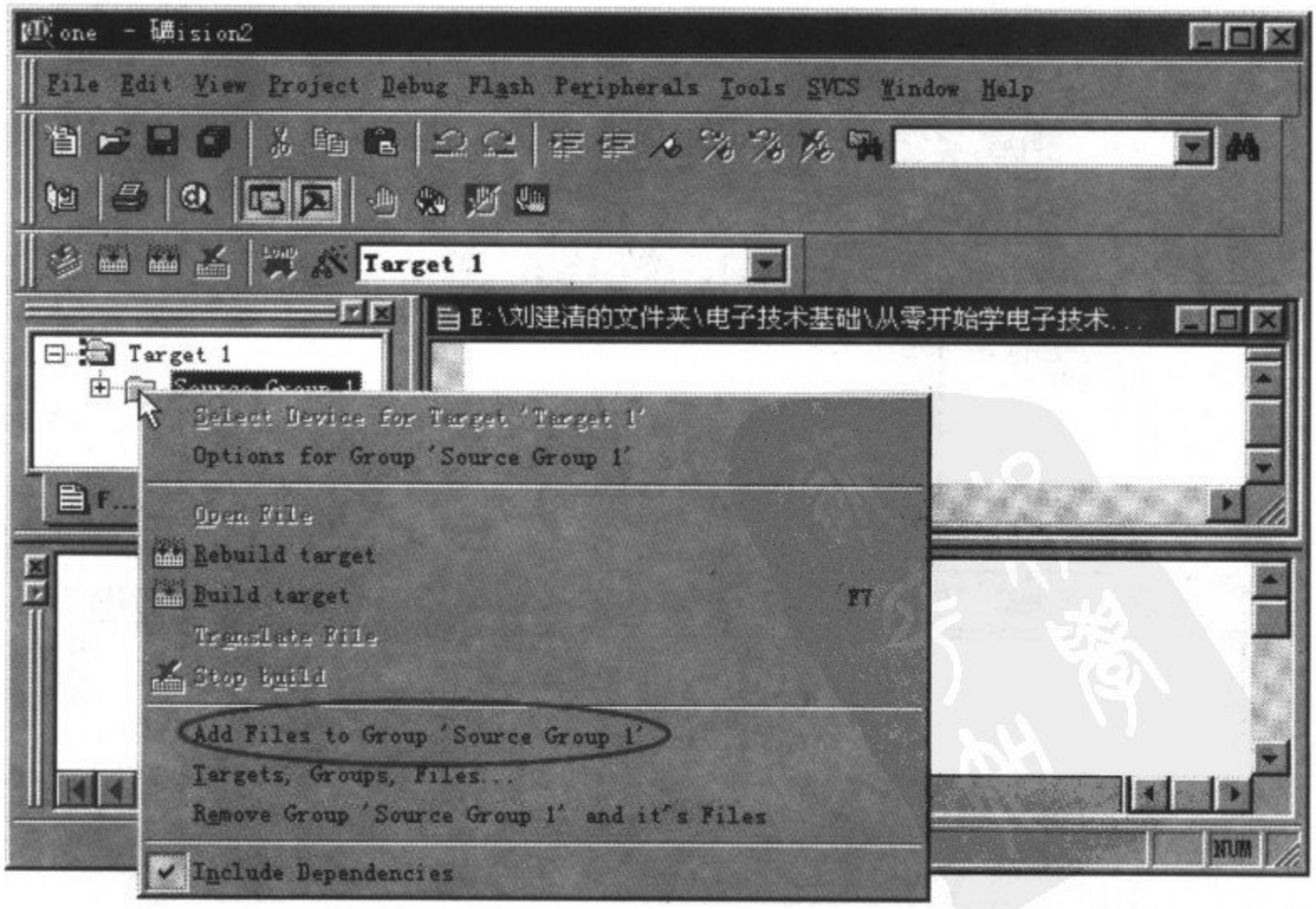


图 4-14　弹出的菜单

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

⑦ 然后单击“Add File to Group ‘Source Group 1’”，出现如图 4-15 所示的增加源文件对话框。

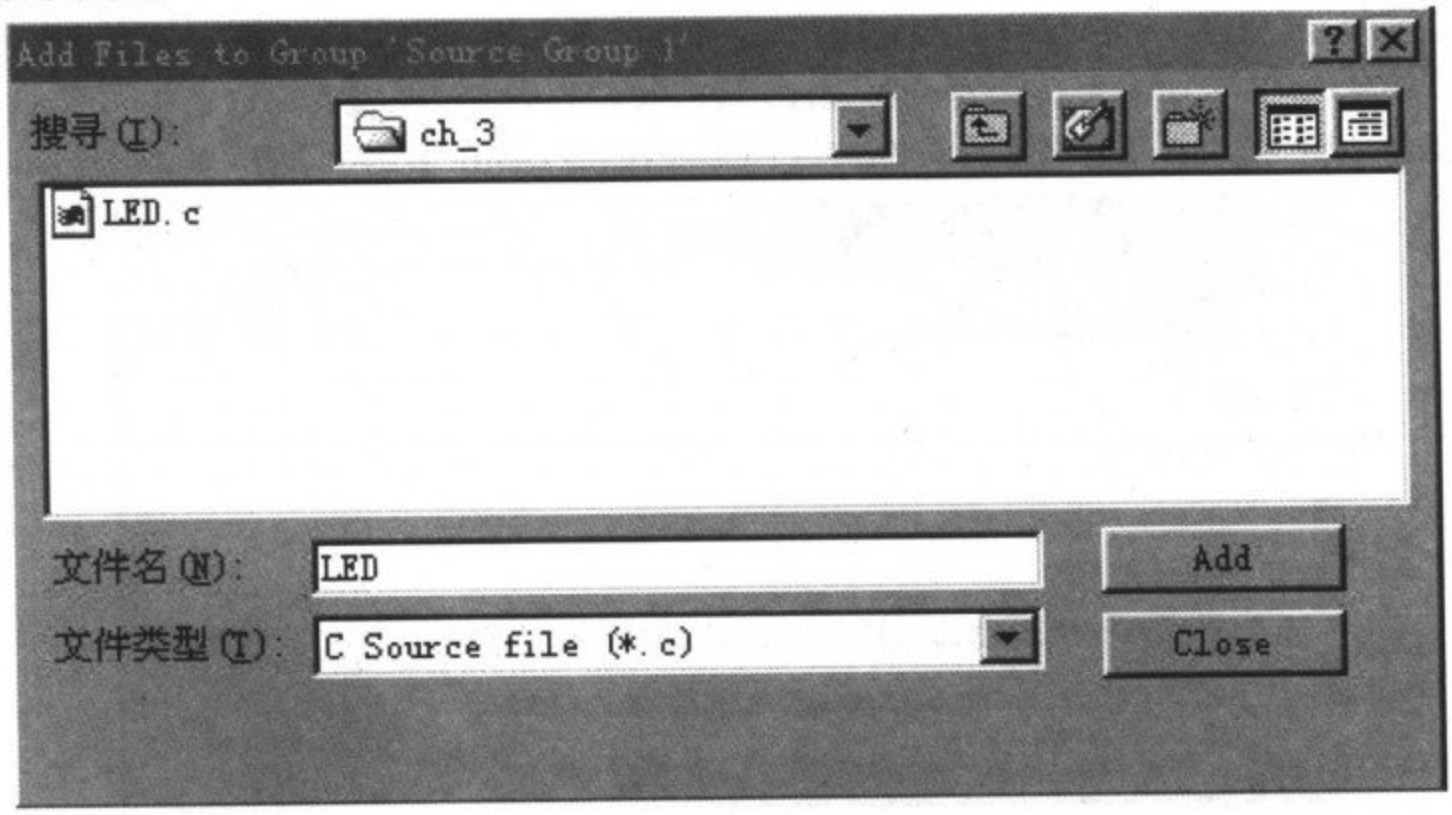


图 4-15　增加源文件对话框

选中“LED.c”，然后单击“Add?”，此时，在“Source Group 1”文件夹中多了一个子项“LED.c”，如图 4-16 所示。子项的数目与所增加的源程序的数目相同。

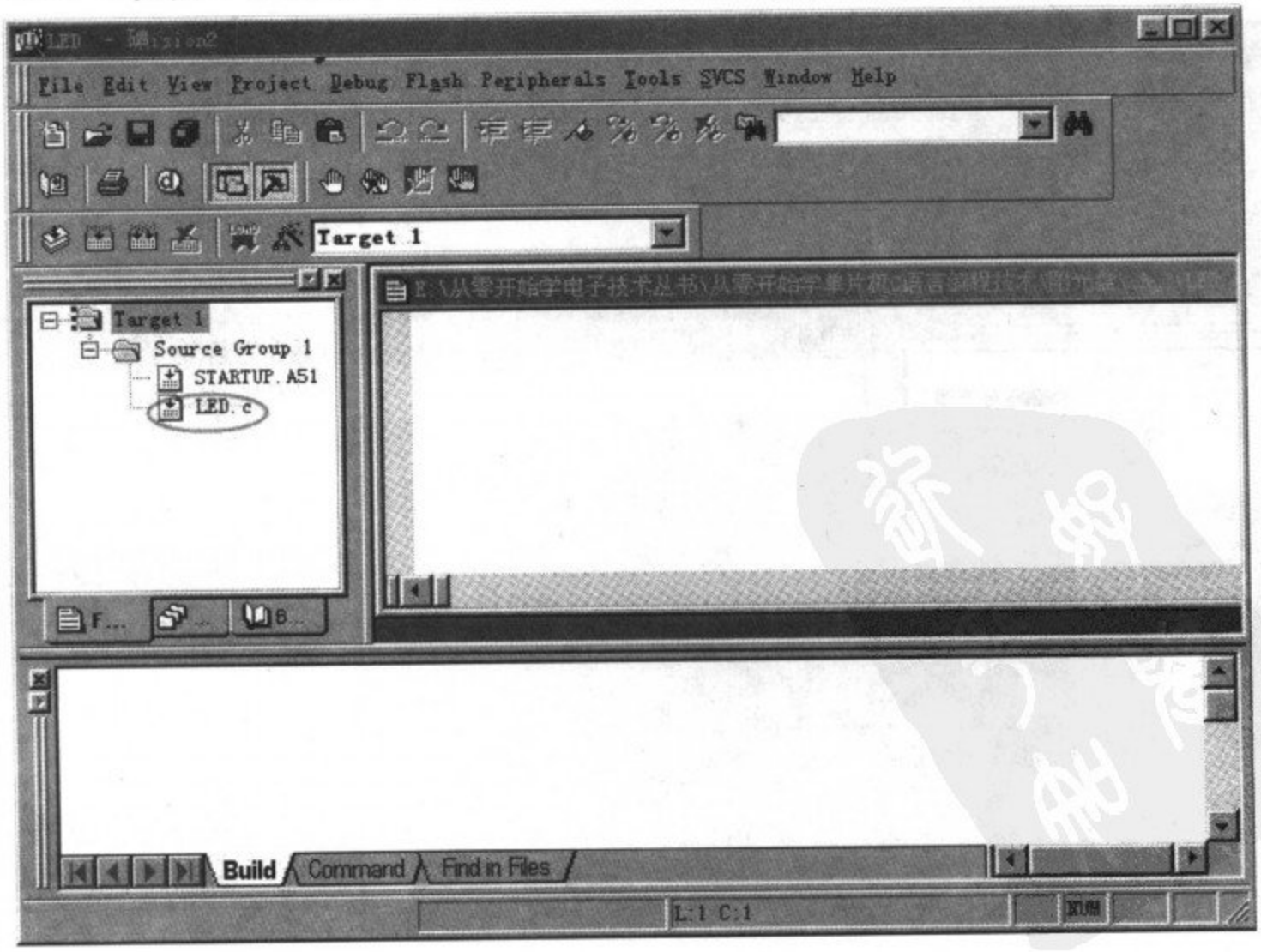


图 4-16　增加文件后的屏幕

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

重点提示 将文件 LED.c 加入“Source Group 1”后，增加源文件对话框并不消失，等待继续加入其他文件，但初学时常会误认为操作没有成功而再次单击“Add”按钮，这时会出现如图 4-17 所示的提示窗口，提示用户所选文件已在列表中，此时应点击“确定”，返回前一对话框，然后点击“Close”即可返回主界面。返回后，点击“Source Group 1”前的加号，会发现 LED.c 文件已在其中。

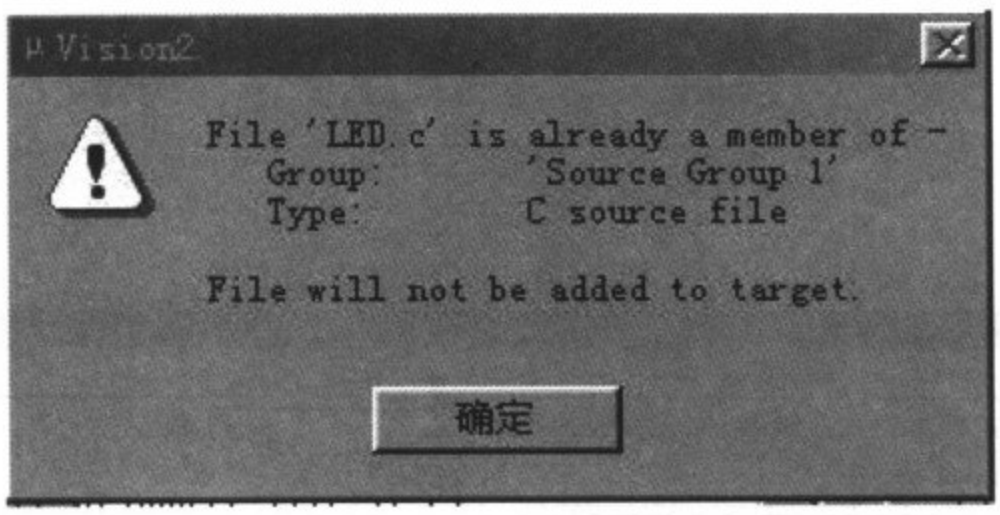


图 4-17 提示窗口

⑧ 现在，在编辑窗口中输入如下的汇编语言源程序（在附赠光盘的 ch_4/LED 文件夹中）：

```
#include<reg51.h>
sbit P10=P1^0;                    //定义位变量
void Delay(unsigned int i)        //延时程序，i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)                 //变量 i 由实际参数传入一个值，因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
}
void main()
{
    for(;;)
    { P10=1;                      //灯灭
    Delay(500);                   // 将实际参数 500 传递给形式参数 i，延时 0.5s
    P10=0;                        //灯亮
    Delay(500);                   //将实际参数 500 传递给形式参数 i，延时 0.5s
    }
}
```

这是一个“灯的闪烁”程序。也就是说，如果把该程序固化到如图 4-18 所示的硬件电路，则该程序可使接在 P1 口 1 脚的发光二极管依次循环点亮。

程序输入完毕后，屏幕如图 4-19 所示。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

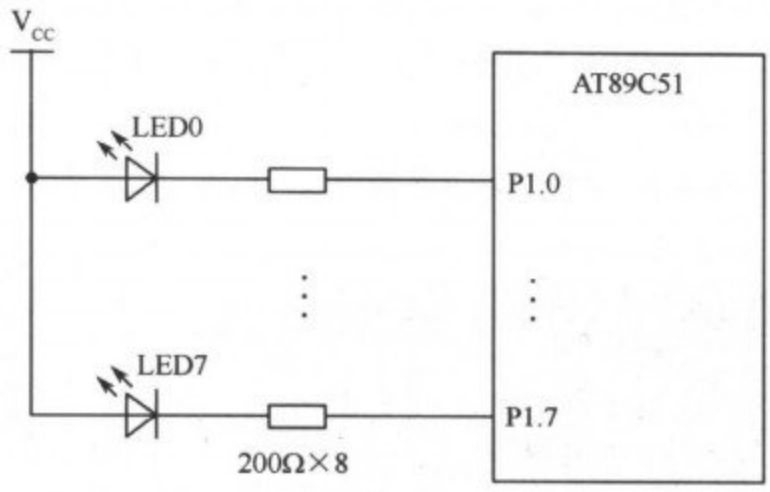


图 4-18　硬件电路

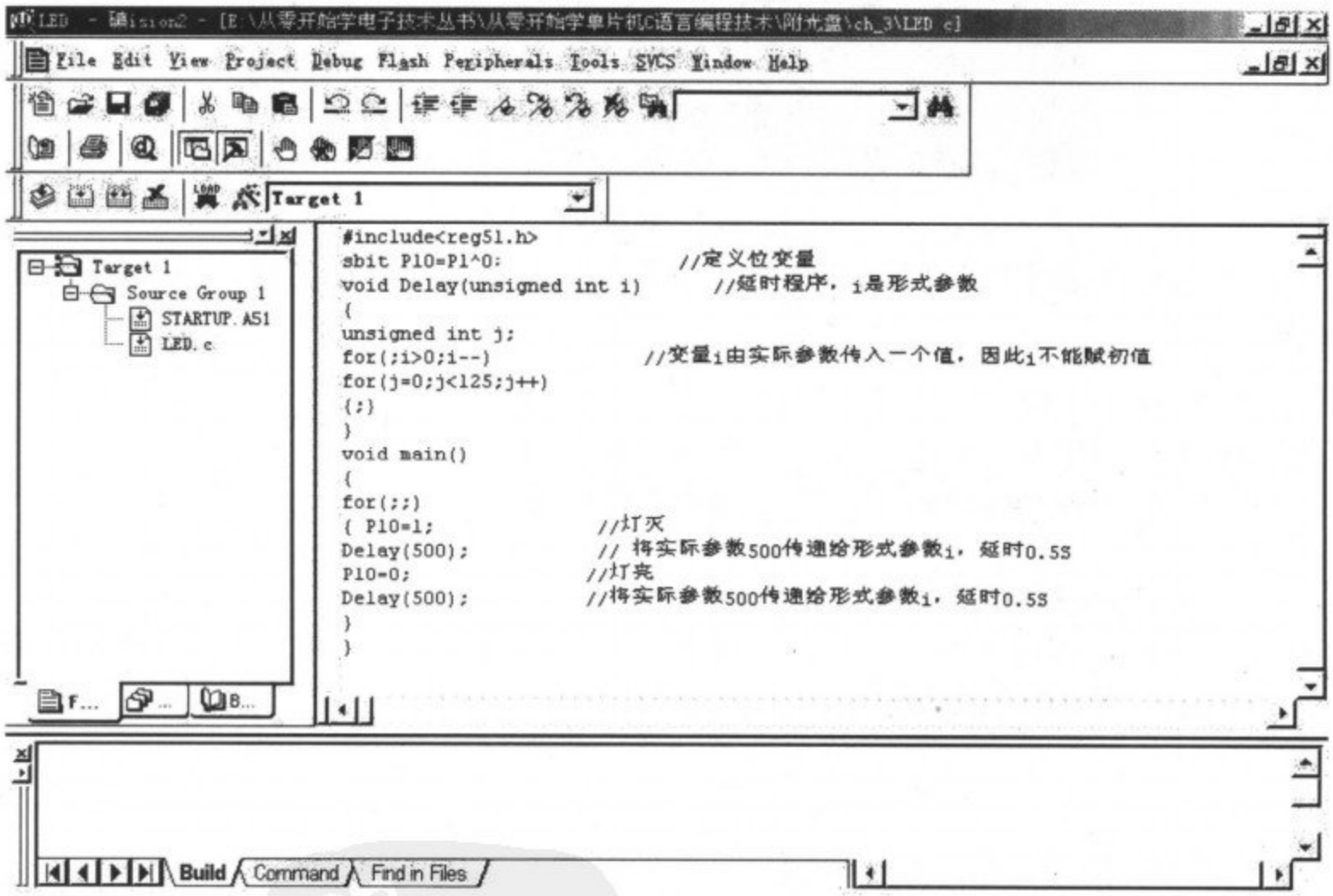


图 4-19　输入程序后的屏幕

三、工程的设置

工程建立好以后，还要对工程作进一步设置，以满足要求。

① 首先点击左边 Project 窗口的 Target 1，然后使用菜单“Project→Option for target ‘target1’”即出现对工程设置的对话框，这个对话框共有 10 个页面，如图 4-20 所示，大部分设置项取默认值就行了。

② Target 页如图 4-20 所示，Xtal 后面的数值是晶振频率值，默认值是所选目标 CPU 的最高可用频率值，该值与最终产生的目标代码无关，仅用于软件模拟调试时显示程序执行时间。正确设置该数值可使显示时间与实际所用时间一致，一般将其设置成与硬件所用晶振频率相同；如果没必要了解程序执行的时间，也可以不设。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

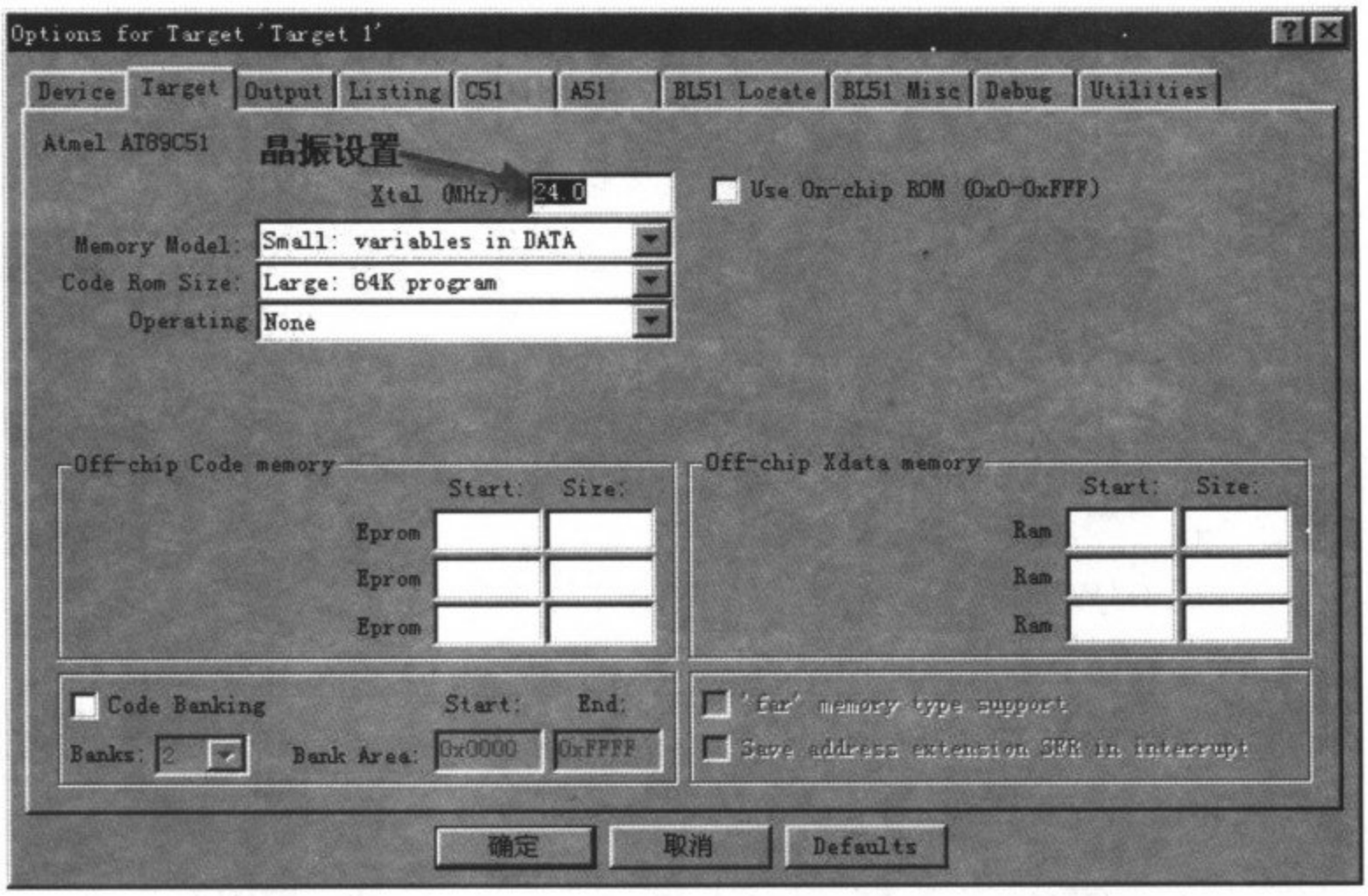


图 4-20　工程设置对话框

Memory Model 用于设置 RAM 使用情况，有三个选择项，即：

Small：所有变量都在单片机的内部 RAM 中。

Compact：可以使用一页(256B)外部扩展 RAM。

Larget：可以使用全部外部的扩展 RAM。

Code Model 用于设置 ROM 空间的使用，同样也有三个选择项：

Small：只用低于 2KB 的程序空间。

Compact：单个函数的代码量不能超过 2KB，整个程序可以使用 64KB 程序空间。

Larget：可用全部 64KB 空间。

这些选择项必须根据所用硬件来决定，对于本例，按默认值设置。

Operating 项是操作系统选择，Keil 提供了两种操作系统：Rtx tiny 和 Rtx full，关于操作系统本书不作介绍，通常不使用任何操作系统，即使用该项的默认值 None。

Off Chip Code memory 用以确定系统扩展 ROM 的地址范围，off Chip xData memory 组用于确定系统扩展 RAM 的地址范围，这些选择项必须根据所用硬件来决定，一般均不需要重新选择，按默认值设置。

③ OutPut 页如图 4-21 所示。

这里面也有多个选择项，其中 Creat Hex File 用于生成可执行代码文件，其格式为 Intel HEX 格式，文件的扩展名为 .HEX，默认情况下该项未被选中，如果要写片做硬件实验，就必须选中该项，这里选择该项。选中该项后，在编译和链接时将产生 *.hex 代码文件，该文件可用编程器去读取并烧到单片机中，再用硬件实验板看结果。

④Listing 页如图 4-22 所示。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

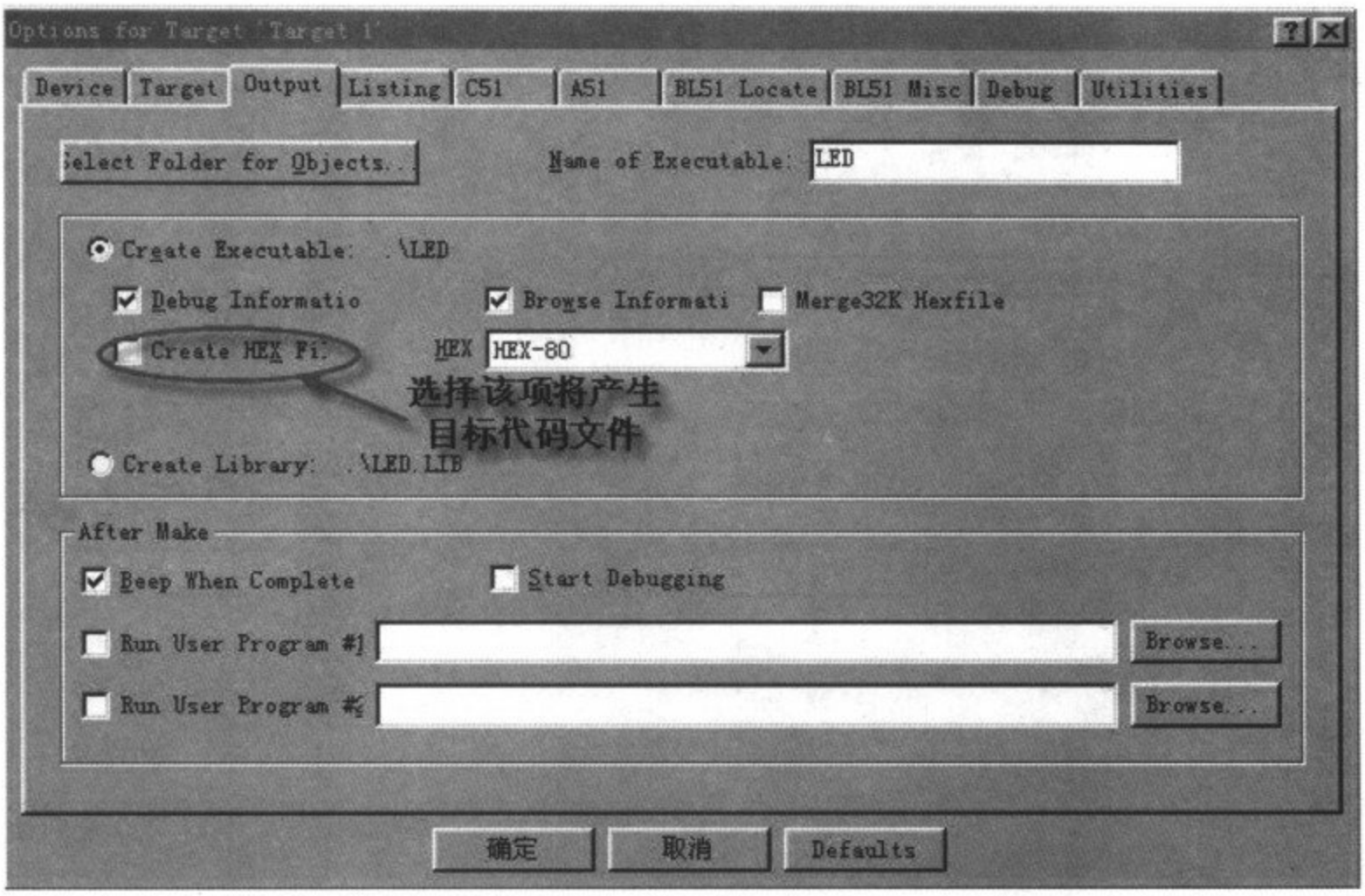


图 4-21　OutPut 页

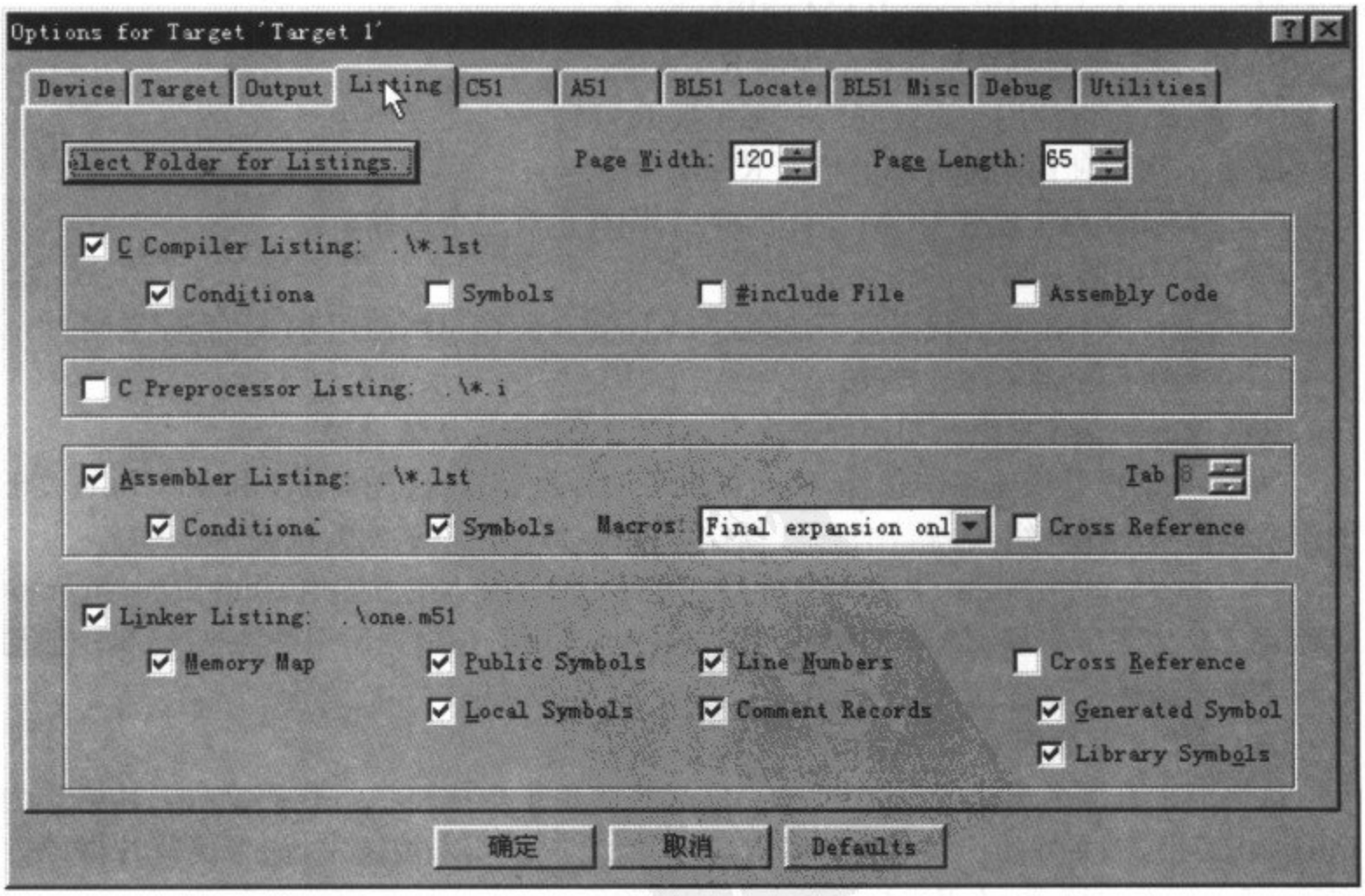


图 4-22　Listing 页

该页用于调整生成的列表文件选项。在汇编或编译完成后将产生 *.lst 的列表文件，在连接完成后也将产生 *.m51 的列表文件，该页用于对列表文件的内容和形式进行

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

细致的调节，一般采用默认设置。

⑤ Debug 页如图 4-23 所示。

该页用于设置调试器，Keil 提供了两种工作模式，即 Use Simulator(软件模拟仿真)和 Use(硬件仿真)，Use Simulator 是将 Keil 设置成软件模拟仿真模式，在此模式下不需要实际的目标硬件就可以模拟 MCS-51 单片机的很多功能，在准备硬件之前就可以测试用户的应用程序，这是很有用的。Use 选项是高级 GDI 驱动，运用此功能，高级用户可以把 Keil C51 嵌入到自己的系统中，从而实现在目标硬件上调试程序。如果没有相应的硬件调试器，应选择 Use Simulator。

⑥ 工程设置对话框中的其他各页面与 C51 编译选项、A51 的汇编选项、BL51 连接器的连接选项等用法有关，这里均取默认值，不做任何修改。设置完成后，单击确定按钮进行确认。

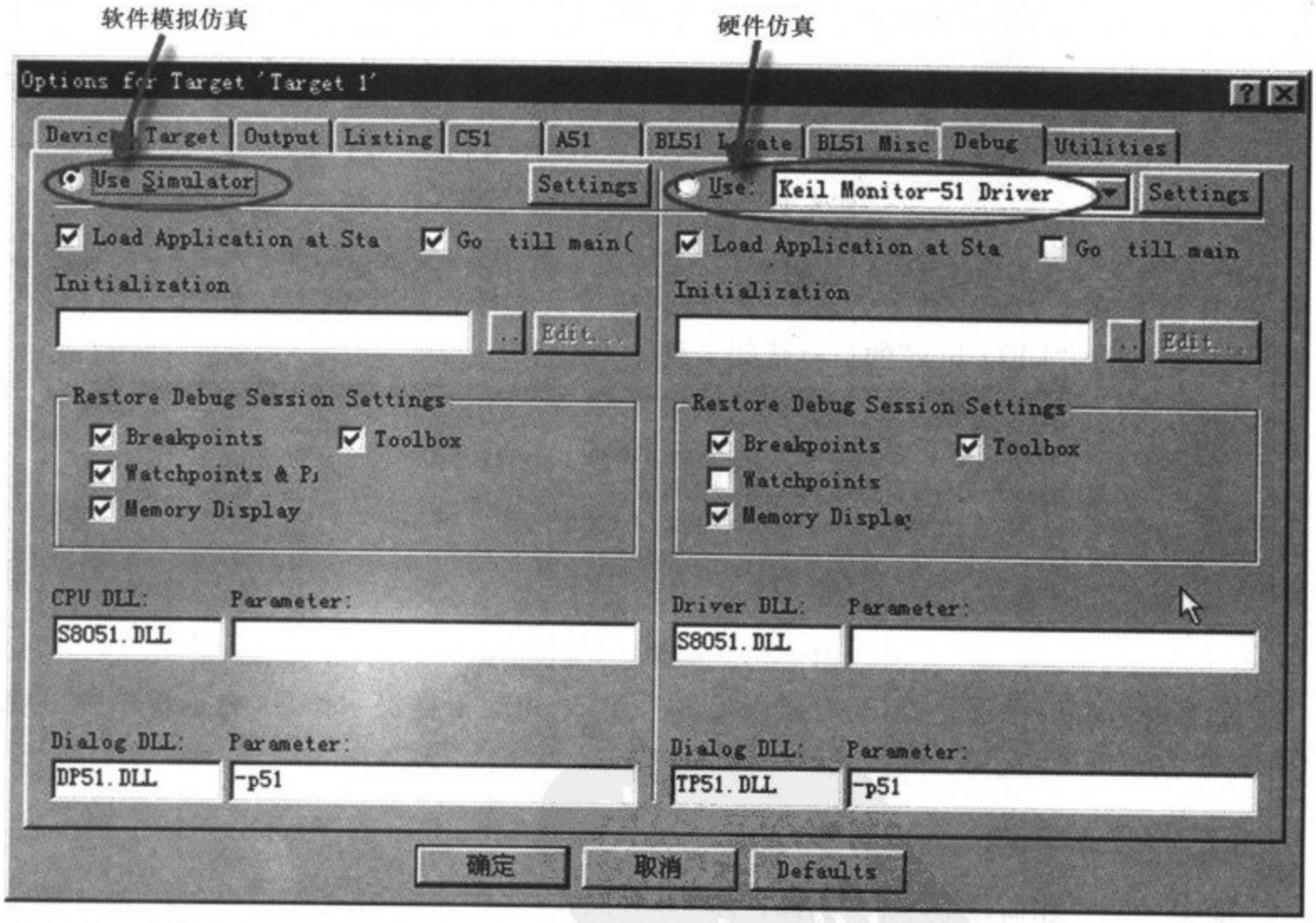


图 4-23　Debug 页

四、程序的编译和链接

汇编程序文件加到了项目中后，就可以编译运行了。在图 4-24 所示图中，1、2、3 都是和编译有关的按钮，不同的是，1 是用于编译按钮，不对文件进行链接；2 是编译链接按钮，用于对当前工程进行链接，如果当前文件已修改，软件会对该文件进行编译，然后再链接以产生目标代码；3 是重新编译按钮，每点击一次均会再次编译链接一次，不管程序是否有改动，确保最终产生的目标代码是最新的。在 3 右边的是停止编译按钮，只有点击了前三个中的任一个，停止按钮才会生效。这几个按钮均可以在“project”菜单中找到。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

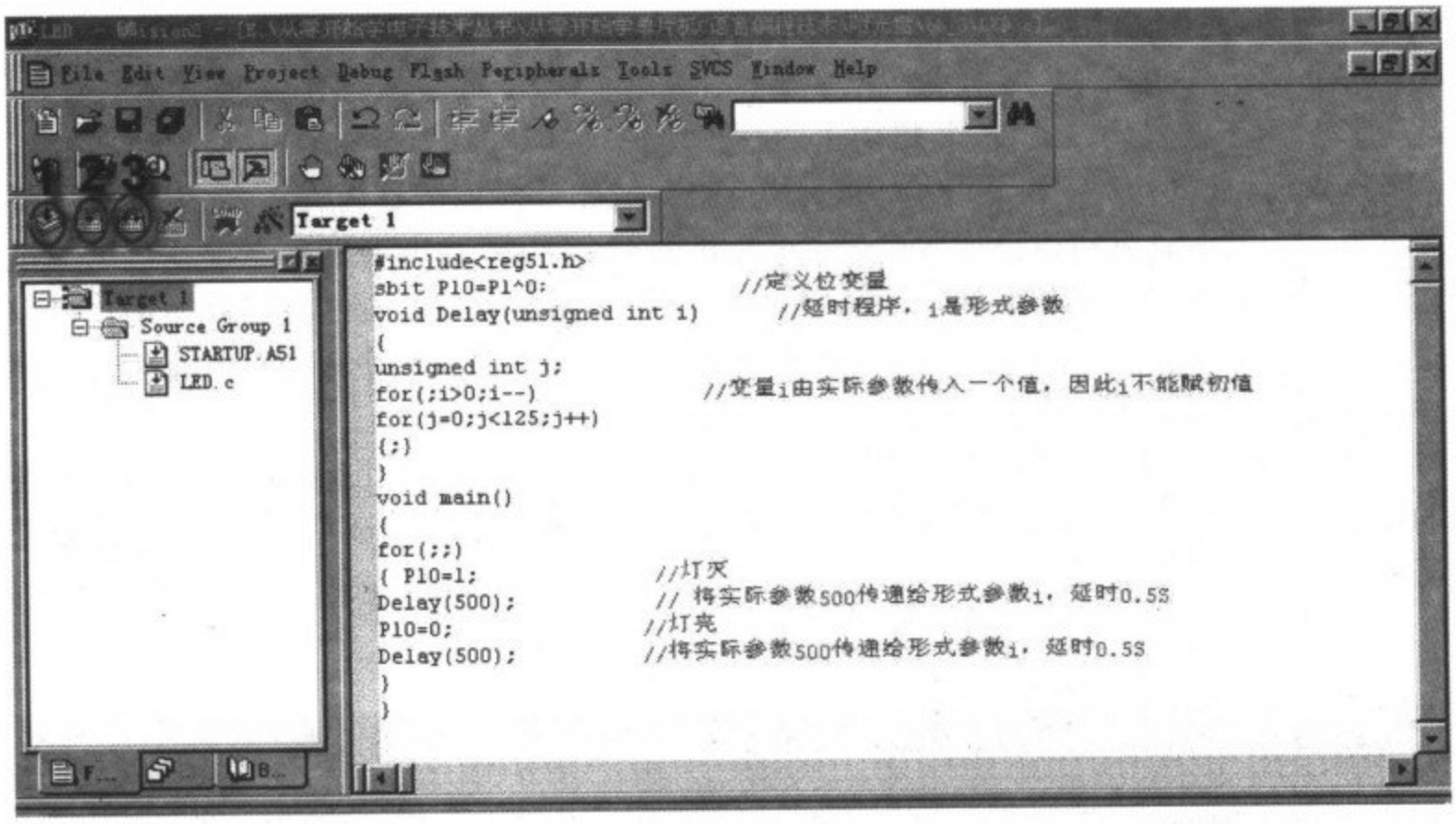


图 4-24　编辑按钮和菜单命令

这个项目只有一个文件，按 1、2、3 中的任一个都可以编译。这里，为了产生目标代码，选择按 2 或 3。在下面的“Build”窗口中可以看到编译后的有关信息，如图 4-25 所示，提示获得了名为“LED. hex”的目标代码文件。

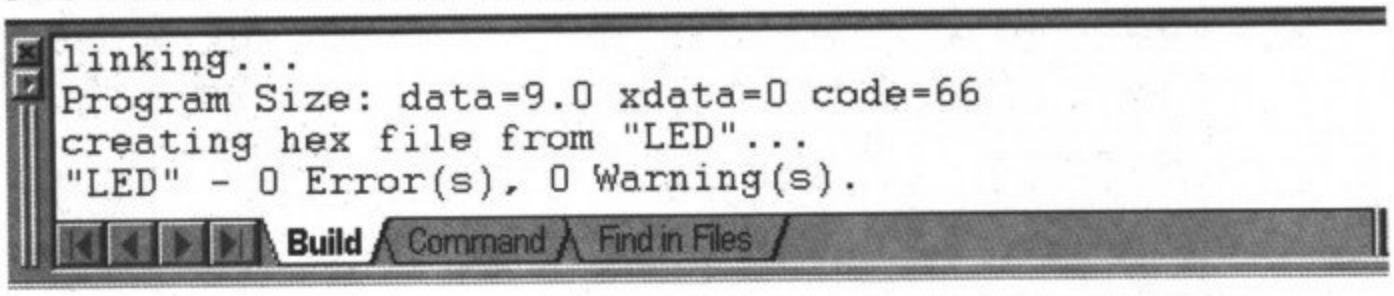


图 4-25　编译后的有关信息

如果源程序有语法错误，会有错误报告出现，用户应根据提示信息，更正程序中出现的错误，重新编译，直至正确为止。

五、程序的调试

前面学习了如何建立工程、编译，并获得目标代码，但是做到这一步仅仅代表源程序没有语法错误，至于源程序中存在的其他错误，必须通过调试才能发现并解决。事实上，除了极简单的程序外，绝大部分的程序都要通过反复调试才能得到正确的结果，因此，调试是软件开发中一个重要的环节，在进行调试之前，要选择软件模拟仿真方式，即将工程设置窗口 Debug 页中的 Use Simulator 选中。

① 对工程成功地进行汇编、连接以后，使用菜单 Debug→Start/Slop Debug Session（或按 Ctrl+F5 键）即可进入调试状态，如图 4-26 所示。

重点提示　Keil 内建了一个仿真 CPU 来模拟执行程序，该仿真 CPU 功能强大，可以在没有硬件和仿真机的情况下进行程序的调试。不过，模拟与真实的硬件执行程序还是有区别的，其中最明显的就是时序。软件模拟是不可能和真实的硬件具有相同的时序

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

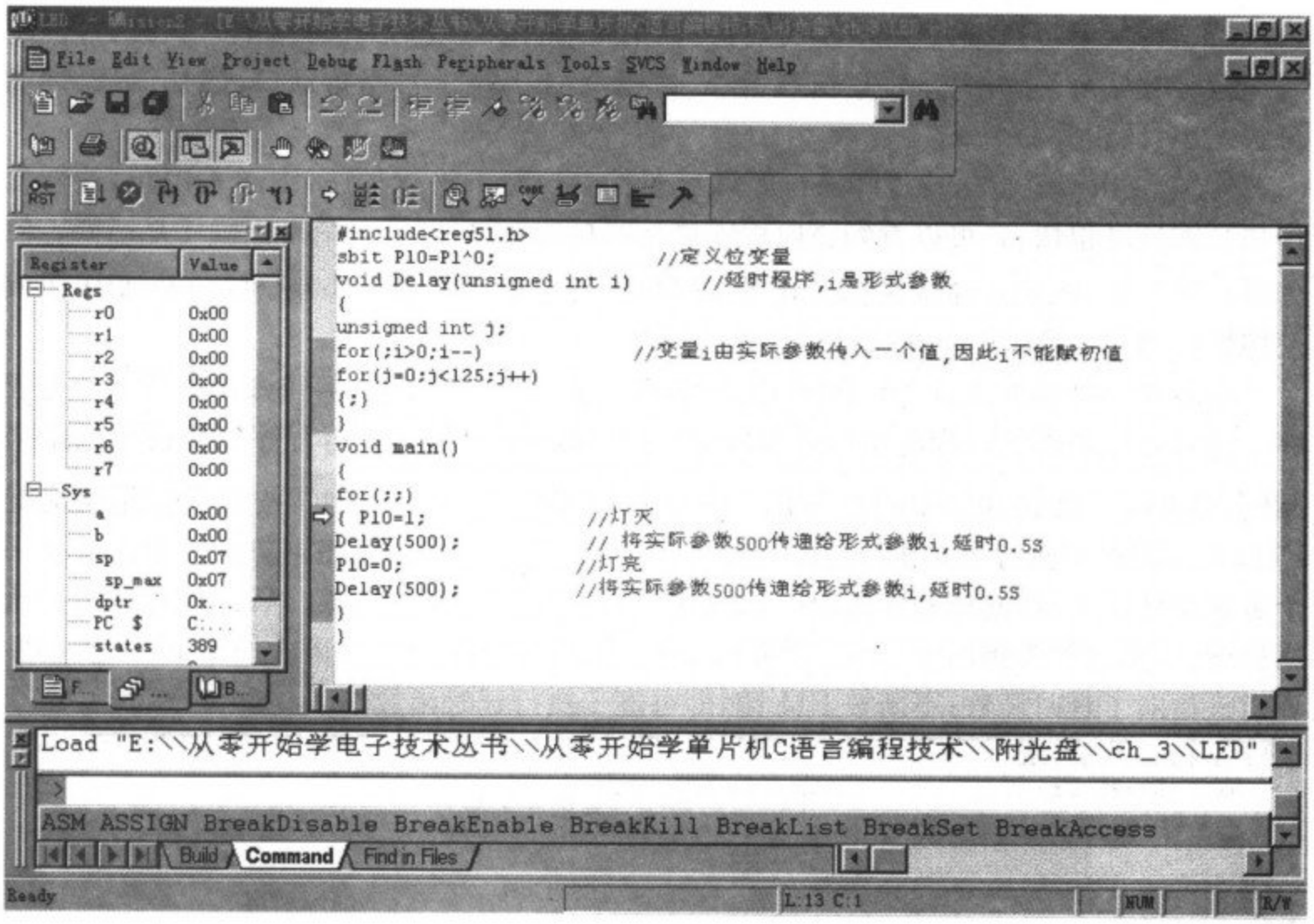


图 4-26　程序调试状态

的，具体的表现就是程序执行的速度和用户使用的计算机有关，计算机性能越好，运行速度越快。

② 进入调试状态后，界面与编辑状态相比有明显的变化，Debug 菜单项中原来不能用的命令现在已经可以使用了，工具栏会多出一个用于运行和调试的工具条，如图 4-27 所示。

Debug 菜单上的大部分命令可以在此找到对应的快捷按钮，从左到右依次是复位、全

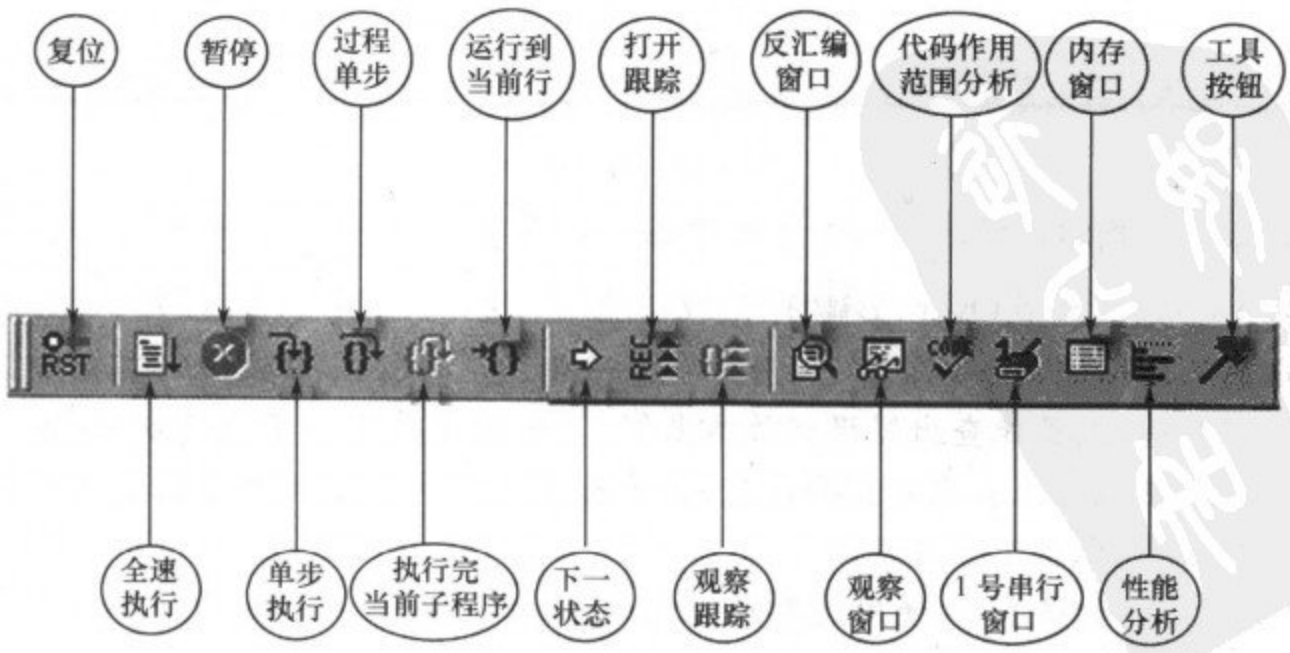


图 4-27　运行和调试工具条

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

速执行、暂停、单步执行、过程单步、执行完当前子程序、运行到当前行、下一状态、打开跟踪、观察跟踪、反汇编窗口、观察窗口、代码作用范围分析、1 号串行窗口、内存窗口、性能分析、工具按钮等命令。

③ 全速执行是指一行程序执行完以后紧接着执行下一行程序，中间不停止，这样程序执行的速度很快，并可以看到该段程序执行的总体效果，即最终结果正确还是错误。但如果程序有错，则难以确认错误出现在哪些程序行。使用菜单 Debug→Go 或 按钮或使用快捷键 F5，可以用全速执行形式执行程序。

单步执行是每次执行一行程序，执行完该行程序以后即停止，等待命令执行下一行程序。此时可以现察该行程序执行完以后得到的结果，是否与写该行程序想要得到的结果相同，借此可以找到程序中问题所在。使用菜单 Debug→Step 或 按钮或使用快捷键 F11，可以用单步执行形式执行程序。按下 F11 键，可以看到源程序窗口的左边出现了一个黄色调试箭头，指向源程序的第一行，如图 4-28 所示。每按一次 F11，即执行该箭头所指程序行，然后箭头指向下一行。当箭头指向“Delay(500) ”行时，再次按下 F11 键，会发现箭头指向了延时函数；不断按 F11 键，即可逐步执行延时函数。

```
#include<reg51.h>
sbit P10=P1^0;                    //定义位变量
void Delay(unsigned int i)        //延时程序,i是形式参数
{
unsigned int j;
for(;i>0;i--)                     //变量i由实际参数传入一个值,因此i不能赋初值
for(j=0;j<125;j++)
{;}
}
void main()
{
for(;;)
{ P10=1;                          //灯灭
Delay(500);                       // 将实际参数500传递给形式参数i,延时0.5S
P10=0;                            //灯亮
Delay(500);                       //将实际参数500传递给形式参数i,延时0.5S
}
}
```

图 4-28　单步执行

过程单步是指将汇编语言中的子程序或 C 语言中的函数作为一个语句来全速执行。使用菜单 Debug→Step Over 或 按钮或功能键 F10，可以用过程单步形式执行命令。

方法技巧　通过单步执行程序，可以找出一些问题的所在，但是仅依靠单步执行来查错有时是困难的，或虽能查出错误但效率很低，为此必须辅之以其他的方法，如在本例中的延时函数需要执行很多次才能执行完，如果用反复按 F11 键来执行延时函数显然不合适。为此，可以采取以下一些方法：

• 在进入该子程序后，使用菜单 Debug→Step Out of Current Function(单步执行到该函数外)或 按钮，即全速执行完调试光标所在的函数，并指向主程序中的下一行程序。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

• 用鼠标在延时函数的最后一行点一下，把光标定位于该行，然后用菜单 Debug→Run to Cursor Line(执行到光标所在行)或 按钮或 Ctrl+F10 键，即可全速执行完黄色箭头与光标之间的程序行。

• 在开始调试时，按过程单步快捷键 F10，程序将单步执行，执行到延时函数时，按下 F10 键，调试光标不进入延时函数的内部，而是全速执行完该延时函数。

灵活应用以上几种方法，可以大大提高查错的效率。

④ 程序调试时，一些程序行必须满足一定的条件才能被执行到(如程序中某变量达到一定的值、按键被按下、串口接收到数据、有中断产生等)，这些条件往往是异步发生或难以预先设定的，这类问题使用单步执行的方法是很难调试的，这时就要使用程序调试中的另一种非常重要的方法——断点设置。断点设置的方法有多种，常用的是在某一程序行设置断点，设置好断点后可以全速执行程序，一旦执行到该程序行即停下，可在此时观察有关变量值，以确定问题所在。在程序行设置/移除断点的方法是将光标定位于需要设置断点的程序行，使用菜单 Debug→Insert/Remove Breakpoint 设置或移除断点(也可以用鼠标在该行双击实现同样的功能)；Debug→Enable/Disable Breakpoint 是开启或暂停光标所在行的断点功能；Debug→Disable All Breakpoint 是暂停所有断点；Debug→Kill All Breakpoint 是清除所有的断点设置。这些功能也可以用工具条上的快捷按钮进行设置。

⑤ 程序调试完成后，可用 Keil 软件仿真程序运行的结果。

按外部设备菜单 Peripherals→I/O Ports→Port 1 就打开 Port1 I/O 观察窗口了，如图 4-29 所示。

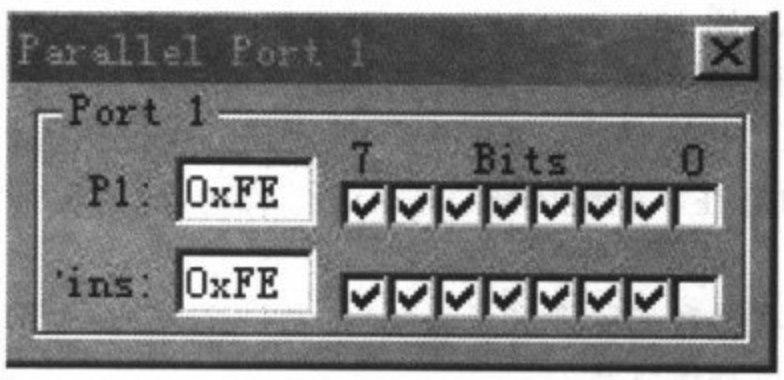


图 4-29 Port 1 调试窗口

图 4-29 中，凡框内打“√”者为高电平，未打“√”者为低电平。按 F5 键全速执行，会发现窗口中 P1.0 中的“√”在不停地闪动，相当于 P1.0 脚外接的灯在不停地闪烁。

六、用实验板进行仿真实验

下面用在线仿真器 MON 51 和下载型实验板进行硬件仿真实验。在线仿真器 MON 51 是完全依托 Keil Cx51 软件强大的功能来实现仿真的，所以必须配合 Keil C51 软件才能工作。具体方法如下：

① 将 MON51 硬件仿真器和 PC 机连接好。取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置，给实验板通电。

② 启动 Keil Cx51 软件，建立工程，输入前面介绍的 LED.c 程序，具体方法在前面已有介绍。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

③ 工程建立好以后，要对工程进行设置，以满足硬件仿真的要求。

点击菜单“Project→Option for? target? ‘target1’”，出现对工程设置的对话框，在Target 页中，Xtal 后面的数值是晶振频率值，输入仿真器的工作频率(11.0592MHz)。在Debug 页中，Keil 提供了两种工作模式，即 Use Simulator(软件模拟仿真)和 UseKeil Monitor-51 Driver(硬件仿真)，在这里，选择 Use，即将 Keil 配置为硬件仿真。单击 Debug 页中的 Settings，出现如图 4-30 所示的设置串口对话框。

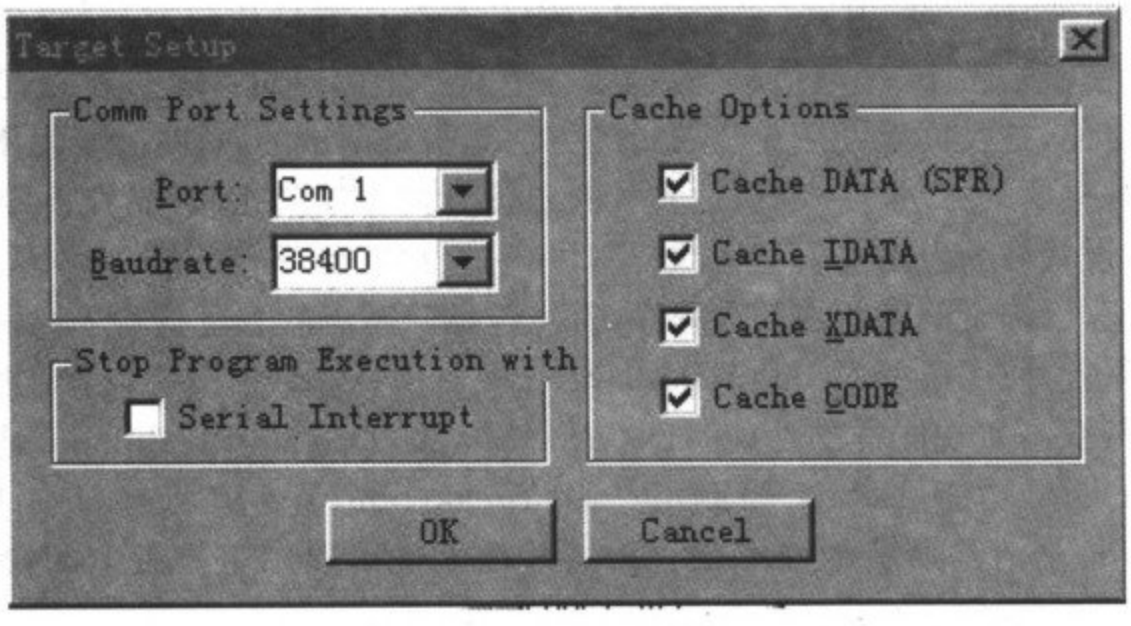


图 4-30　设置串口对话框

设置选项选择您要使用的串行口，必须和实际相符合。如果被仿真的目标板使用12MHz 或者是 11.0592MHz 晶振时，波特率选择 38400，如果被仿真的目标板使用6MHz 晶振时，波特率选择 18400。这里波特率选择 38400。

④ 按编译按钮进行编译，编译成功后，可以进行硬件仿真了。

⑤ 按 Ctrl+F5 可以进入仿真，这时再按 F5 键进入全速执行状态。这时会看到实验开发板 P1.0 脚外接的灯在不停地闪烁，表示运行成功，这时可以查看相关的变量和参数，非常方便。

注意事项　① 仿真器和实验开发板连接时，请注意方向，否则会烧毁 MON 51 仿真器。② 请在断电后再拔插通信线。

第四节　常用窗口介绍

为帮助用户分析和调试程序，Keil 提供了以下几个重要的调试窗口，下面分别进行介绍。

一、变量观察窗口

在调试状态下单击菜单 View→Watch & Call Stack Window 选项或单击按钮，即可打开或关闭该窗口(若原先没有打开该窗口，执行该命令就可以打开该窗口；若原先已经打开了该窗口，执行该命令将关闭该窗口)。打开后的窗口如图 4-31 所示。

此窗口又包括 4 个小窗口(分 4 页显示)，分别是 Locals、Watch＃1，Watch＃2 和 Call Stack。可以在 Locals 窗口中观察局部变量的值，也可以在 Watch＃1、Watch＃2 观察窗

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

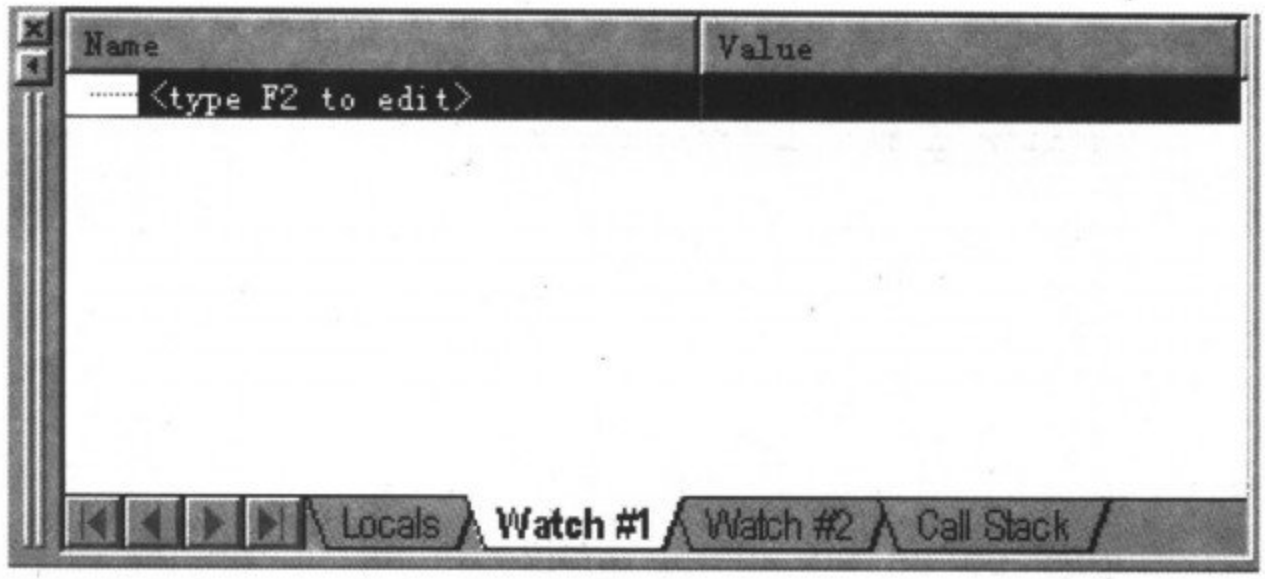


图 4-31　变量观察窗口

口中输入被调试的变量名，系统会自动在 Value 栏内显示该变量的值，而 Call Stack 观察窗口主要给出了一些调用子程序时的基本信息。

如果想观察寄存器 A 中的值，可以在 Watch＃1 窗口中选中 type F2 to edit，然后按 F2 键，输入“A”即可。再用同样的方法输入变量 P10，修改后的窗口如图 4-32 所示。

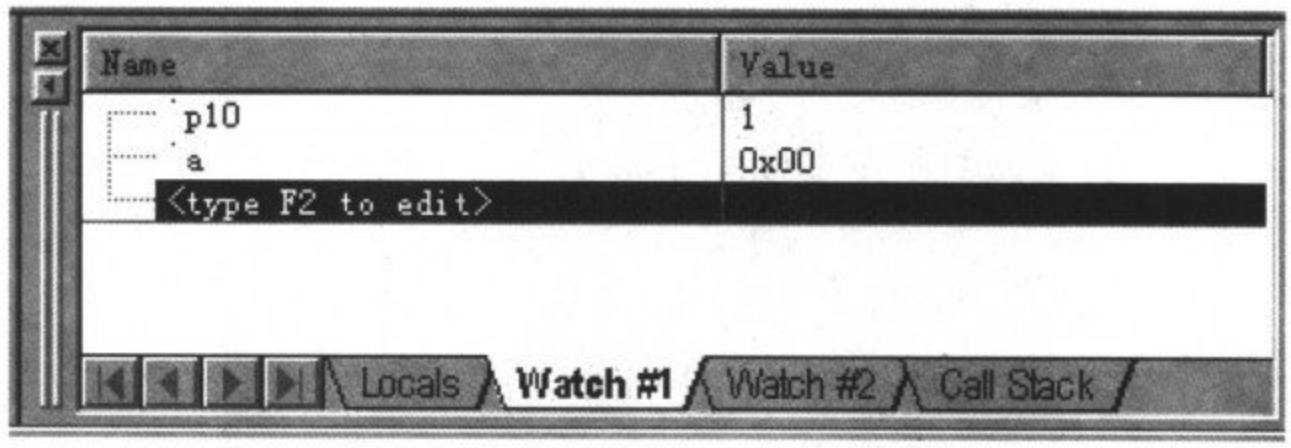


图 4-32　修改后的 Watch＃1 窗口

此时，单击全速执行按钮，会发现 Watch＃1 窗口中寄存器 A 的值不断地变化，同时，P10 脚的值会在 0 和 1 之间不断转换。

方法技巧　一般情况下，用户仅在单步执行时才对变量值的变化感兴趣，全速执行时，变量的值是不变的，只有在程序停下来之后，才会将这些值的最新变化反映出来。但是，在一些特殊场合，用户也可能需要在全速执行时观察变量的变化，此时可以点击 View→>Periodic Window Update(周期更新窗口)，确认该项处于被选中状态，即可在全速执行时动态地观察有关值的变化。但是，选中该项，将会使程序模拟执行的速度变慢。

二、存储器观察窗口

在调试状态下单击菜单 View→Memory Window 或按钮，即可打开或关闭该窗口。此窗口也同样包括 4 小窗口，分别是 Memory＃1、Memory＃2、Memory＃3、Memory＃4，通过这些窗口可以观察不同存储区不同单元，DATA 是可直接寻址的片内数据存储区，XDATA 是外部数据存储区，IDATA 是间接寻址的片内数据存储区，CODE 是程序存储区。可以在存储区观察窗口的 Address 栏内输入相应的字母(D、X、I、C)来观

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

察不同的存储单元，如输入“C:0x00”，则系统会给出从 00H 单元开始的程序存储器(ROM)及其相应的值，即查看程序的二进制代码，如图 4-33 所示。

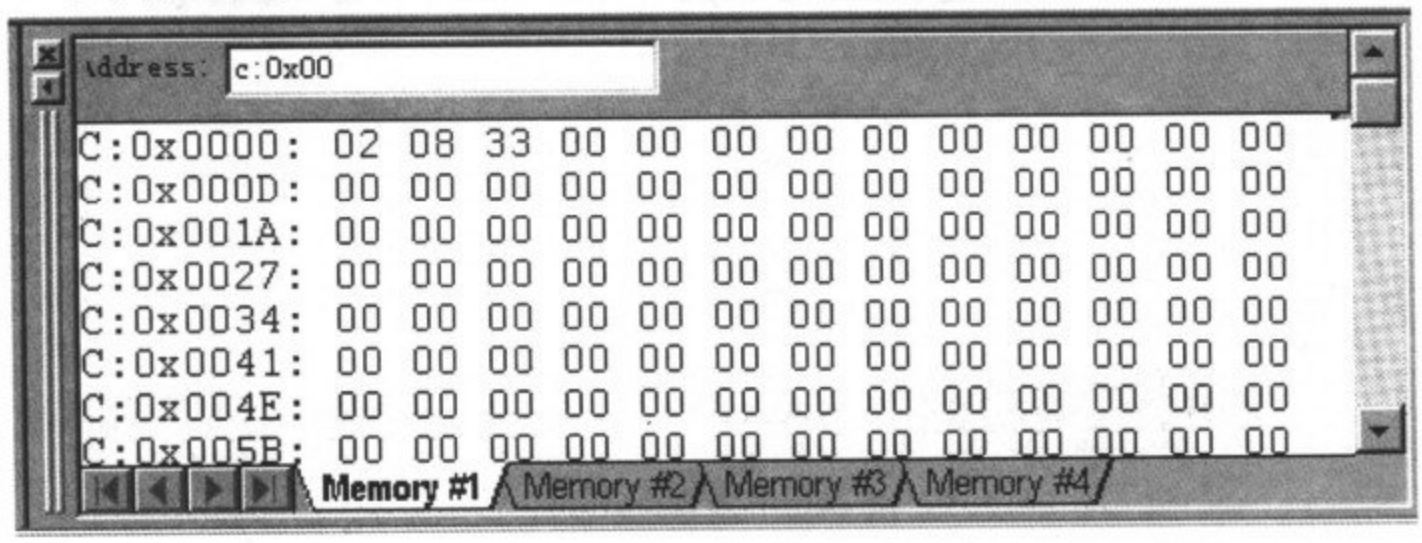


图 4-33　00H 单元开始的程序存储器及其相应的值

该窗口的显示值可以以各种形式显示，如十进制、十六进制、字符型等，改变显示方式的方法是点击鼠标右键，弹出如图 4-34 所示的快捷菜单。

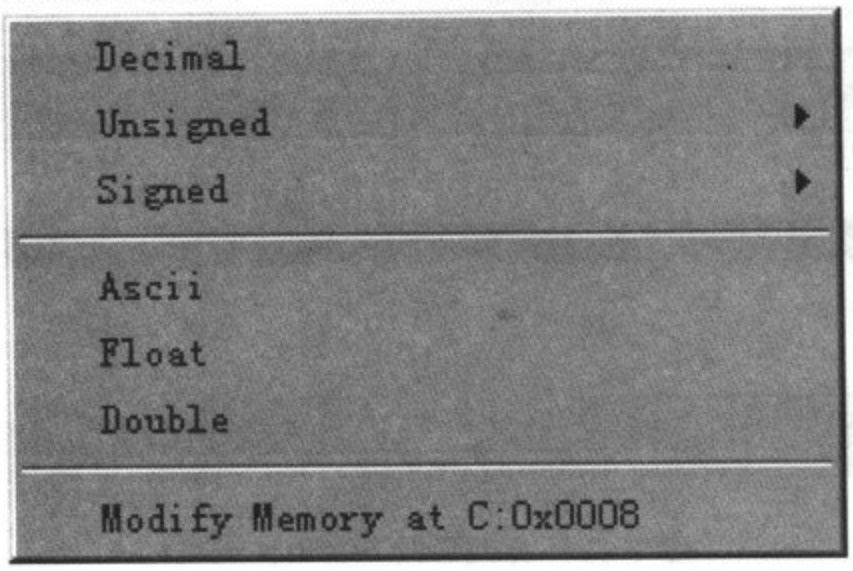


图 4-34　快捷菜单

该菜单用分隔条分成三部分，其中第一部分与第二部分的三个选项为同一级别。选中第一部分的任一选项，内容将以整数形式显示；而选中第二部分的 ASCII 项，则将以字符形式显示，选中 Float 项，将以相邻 4 字节组成的浮点数形式显示，选中 Double 项，则将相邻 8 字节组成双精度形式显示。第一部分又有多个选择项，其中 Decimal 项是一个开关，如果选中该项，则窗口中的值将以十进制的形式显示；否则，按默认的十六进制方式显示。Unsigned 和 Signed 后分别有三个选项：Char、Int、long 分别代表以单字节方式显示、将相邻双字节组成整型数方式显示、将相邻 4 字节组成长整型方式显示，而 Unsigned 和 Signed 则分别代表无符号形式和有符号形式，究竟从哪一个单元开始的相邻单元，则与用户的设置有关。以整型为例，如果输入的是“I:0”，那么 00H 和 01H 单元的内容将会组成一个整型数，而如果输入的是“I:1”，01H 和 02H 单元的内容会组成一个整型数，依此类推。默认以无符号单字节方式显示，一般采用默认方式即可。第三部分的 Modify Memory at X:xx 用于更改鼠标处的内存单元值，选中该项即出现对话框，可以在对话框内输入要修改的内容。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

三、寄存器观察窗口

在调试状态下单击菜单 View→Project Window 选项或工具栏上的按钮，即可打开或关闭如图 4-35 所示的寄存器观察窗口。

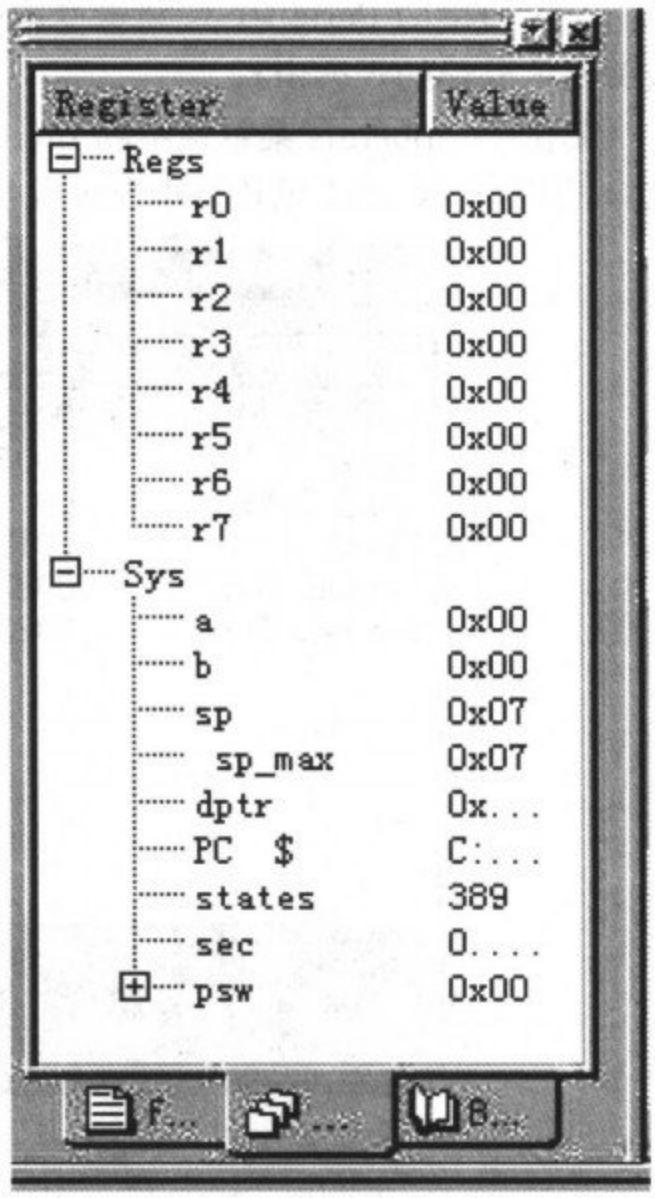


图 4-35　寄存器观察窗口

寄存器窗口包括两组：通用寄存器组 Regs 和系统特殊寄存器组 Sys。通用寄存器组包括 r0～r7 共 8 个寄存器，而系统寄存器组包括寄存器 a、b、sp、pc、dptr、psw 和 sec(能够观察每条指令执行时间)等共 10 个寄存器。这些寄存器是程序中经常使用的和控制程序运行中至关重要的。通过观察这些寄存器的变化，将更加有利于用户分析程序。

每当程序中执行到对某寄存器的操作时，该寄存器会以反色(蓝底白字)显示，用鼠标单击然后按下 F2 键，即可修改该值。

方法技巧　利用寄存器窗口可以观察程序执行的时间，下面观察一下 LED.c 中延时程序的延时时间是否满足要求，即是否确实延时 0.5s。展开工程窗口 Regs 页中的 Sys 目录树，其中的 Sec 项记录了从程序开始执行到当前程序流逝的秒数。点击 RST 按钮以复位程序，Sec 的值回零，按下 F10 键，程序窗口中的黄色箭头指向 Delay(500)行，此时，记录下 Sec 值为 0.00042318，然后再按 F10 键执行完该段程序，再次查看 Sec 的值为 0.55057075，两者相减大约是 0.55 s，所以延时时间大致是正确的。读者可以试着将延时程序中的 unsigned int 改为 unsigned char，试试看时间是否仍正确。注意，使用这一功能的前提是在项目设置中正确设置晶振的数值。

四、串口调试观察窗口

在调试状态下单击菜单 View→Serial Window＃1 或 Serial Window＃2 选项，即可打开或关闭该窗口。该窗口提供了一个调试串口的界面，串口发送或接收的数据都可以在该窗口输出或输入。

五、反汇编观察窗口

在调试状态下单击菜单 View→Disassembly Window 选项，即可打开或关闭该窗口。通过该窗口，可以看出每条 C 语言程序的汇编代码，这对于分析一些 C 程序是很有帮助的。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

六、外围设备观察窗口

外围设备观察窗口通常包括中断系统观察窗口、I/O 口观察窗口、串口属性观察窗口和定时/计数器观察窗口 4 个基本组成部分。这些外围设备观察窗口可以在调试状态下单击主窗口中的 Peripherals 菜单栏选项，在弹出的下拉菜单中选择相应的选项即可打开或关闭该外设的观察窗口。

单击 Peripherals 菜单栏中的 Interrupt 选项，将弹出如图 4-36 所示的中断系统观察窗口，用于显示 8051 单片机中断系统状态。

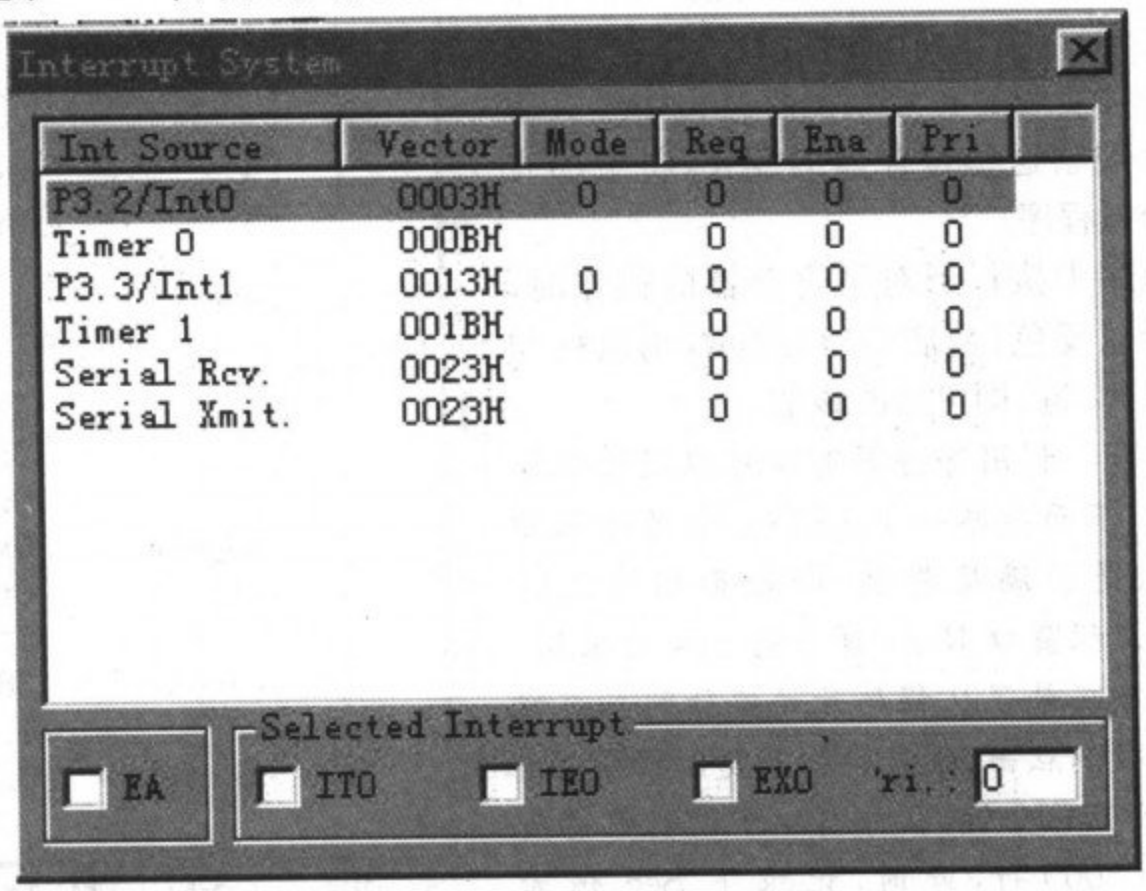


图 4-36　中断系统观察窗口

选中不同的中断源，窗口中“Selected Interrupt”栏将出现与之相对应的中断允许和中断标志位的复选框。通过对这些状态位的置位和复位操作(选中或不选中)，很容易实现对单片机中断系统的仿真。

对于具有多个中断源的单片机如 80C52 等，除了如上所述几个基本中断源之外，还可以对其他中断源如监视定时器(Watchdog Timer)等进行模拟仿真。

Peripherals 菜单中的“I/O-Ports”选项用于仿真 8051 单片机的并行 I/O 接口 Port0～Port3，选中 Port1 后将弹出如图 4-37 所示窗口。

其中“P1”栏显示 8051 单片机 P1 口锁存器状态，“Pins”栏显示 P1 口各个引脚的状态，仿真时它们各位的状态可根据需要进行修改。

Peripherals 菜单中“Serial”选项用于仿真 8051 单片机的串行口，单击该选项，将弹出如图 4-38 所示窗口。

窗口中“Mode”栏用于选择串行口的工作方式，单击其中的箭头，很容易选择 8 位移位寄存器、8 位/9 位可变波特率 UART、9 位固定波特率 UART 等不同工作方式。选定工作方式后，相应特殊工作寄存器 SCON 和 SBUF 的控制字也显示在窗口中。通过对特殊控制位 SM2、REN、TB8、RB8、TI 和 RI 复选框的置位和复位操作(选中或不选中)，很

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

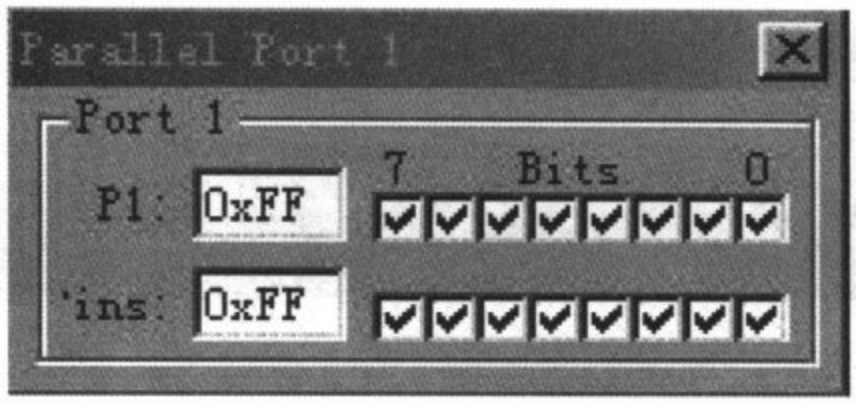


图 4-37　P1 口观察窗口

容易实现对 8051 单片机内部串行口的仿真。“Baud rate”栏用于显示串行口的工作波特率，SMOD 位置位时将使波特率加倍。“IRQ”栏用于显示串行口的发送和接收中断标志。

Peripherals 菜单中的“Timer”选项用于仿真 8051 单片机内部定时/计数器，选中其中 Timer0 后将弹出如图 4-39 所示窗口。

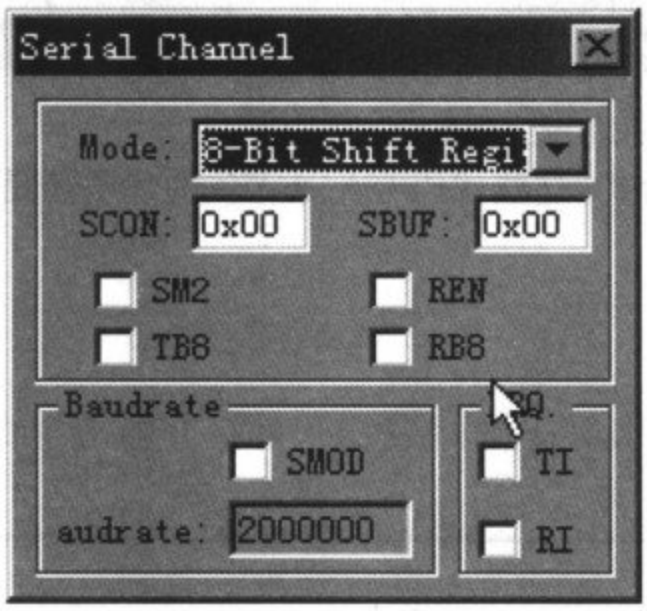


图 4-38　串行属性观察窗口

图 4-39　定时/计数器 0 观察窗口

窗口中“Mode”栏用于选择工作方式，可选择定时器或计数器方式，单击其中的箭头，很容易实现选择。选定工作方式后相应特殊工作寄存器 TCON 和 TMOD 的控制字也显示在窗口中，TH0 和 TL0 用于显示计数初值，T0 Pin 和 TF0 复选框用于显示 T0 引脚和定时/计数器的溢出状态。窗口的“Control”栏用于显示和控制定时/计数器的工作状态(Run 或 Stop)，TR0、GATE 和 INT0＃复选框是启动控制位，通过对这些状态位的置位和复位操作(选中或不选中)，很容易实现对 8051 单片机内部定时/计数器仿真。

七、代码作用范围分析窗口

在编写的程序中，有些代码可能永远不会被执行到(这是无效的代码)，也有一些代码必须在满足一定条件后才能被执行到，借助于代码作用范围分析工具，可以快速地了解代码的执行情况。进入调试后，全速执行，然后按停止按钮，停下来后，使用调试工具条上的 CODE 按钮，可打开代码作用范围分析对话框，如图 4-40 所示。

里面有各个模块代码执行情况的更详细的分析。同时，在源程序的左列有三种颜色：灰、淡灰和绿。其中，淡灰所指的行并不是可执行代码，如变量或函数定义、注释行等，灰色行是可执行但从未执行过的代码，而绿色则是已执行过的程序行。如果发现全速执行

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

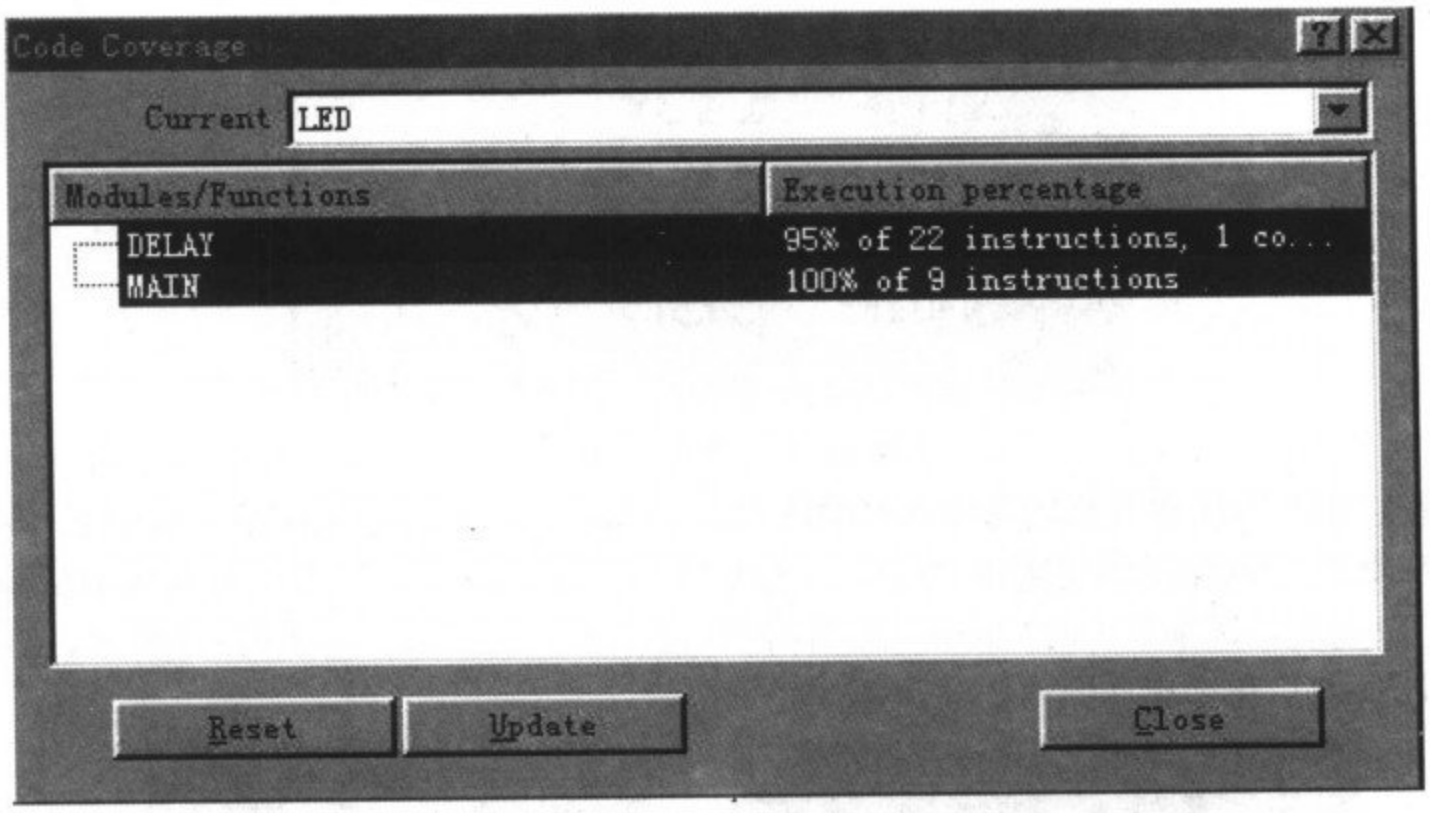


图 4-40　代码作用范围分析窗口

后有一些未被执行到的代码，那么就要仔细分析，这些代码究竟是无效的代码还是因为条件没有满足而没有被执行到。

八、输出窗口

点击菜单 View→Out Windows 或按工具栏上的 按钮，可打开输出窗口，如图4-41所示。

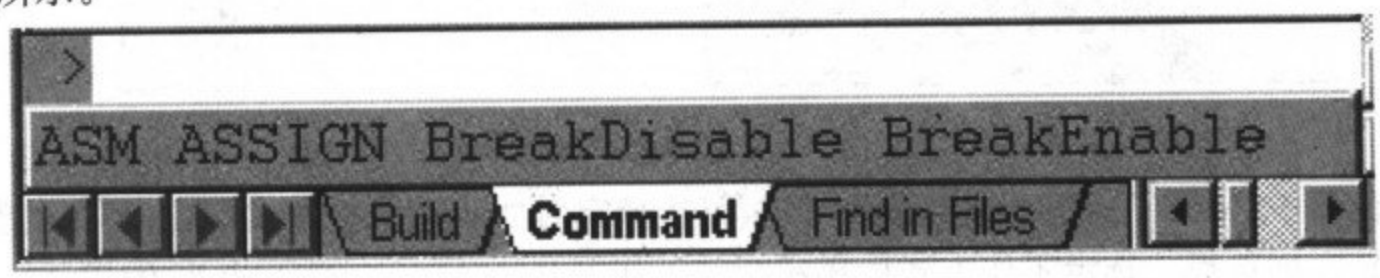


图 4-41　输出窗口

进入调试程序后，输出窗口自动进入到 Command 页，该页用于输入调试命令和输出调试信息。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第五章　Cx51 数据与运算

Keil Cx51 是一种专用 8051 单片机设计的 C 编译器，支持符合 ANSI 标准的 C 语言进行程序设计。同时，针对 8051 单片机自身特点作了一些特殊扩展。C 语言语法简洁、紧凑，使用方便、灵活，在单片机开发中得到了广泛的应用。本章主要介绍 C 语言的基本语法结构：数据类型、常量与变量，以及运算符和表达式。

第一节　标识符和关键字

一、标识符

标识符是用来标识源程序中某个对象的名字的，这些对象可以是语句、数据类型、函数、变量、数组等等。

1. 标识符命名规则

标识符的命名应符合以下规则：

①有效字符只能由字母、数字和下划线组成，且以字母或下划线开头。

②有效长度随系统而异，但至少前 8 个字符有效，如果超长，则超长部分被舍弃。

③C 语言的关键字不能用作变量名。

2. 注意事项

标识符的第一个字符必须是字母或下划线，如“1timer”是错误的，编译时便会有错误提示。有些编译系统专用的标识符是以下划线开头，所以一般不要以下划线开头命名标识符。

标识符在命名时，应当简单、含义清晰，尽量为每个标识符取一个有意义的名字，这样有助于阅读理解程序。比如 NumberOfStudents，这个变量一看就明白是为“学生的人数”定义的。而如果定义为 aaa、bbb 之类，就容易混淆。当然，有一些约定俗成的用法，如字母 i、j 等，常用来作循环流程中的计数变量。再者，有意义的名字并不是指一定要把变量所要代表的意思用英文句子写出，一般可以用简写，如 NumOfStudent，它同样意义明了，但更简短。此外，也可以使用拼音。

C 语言是对字母大小写敏感的一种高级语言，如果要定义一个定时器 1，可以写做“timer1”，如果程序中有“TIMER1”，那么这两个是完全不同定义的标识符。

在 C51 编译器中，只支持标识符的前 32 位为有效标识，一般情况下也足够用了。

二、关键字

关键字则是编程语言保留的特殊标识符，它们具有固定名称和含义，在程序编写中不允许标识符与关键字相同。与其他语言相比，C 语言的关键字是比较少的，ANSI C 标准规定了 32 个关键字，如表 5-1 所列。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

表 5-1　ANSI C 标准的关键字

关键字	用　途	说　明
auto	存储种类说明	用以说明局部变量，缺省值为自动变量
break	程序语句	退出最内层循环
case	程序语句	Switch 语句中的选择项
char	数据类型说明	单字节整型数或字符型数据
const	存储类型说明	在程序执行过程中不可更改的常量值
continue	程序语句	转向下一次循环
default	程序语句	Switch 语句中的失败选择项
do	程序语句	构成 do…while 循环结构
double	数据类型说明	双精度浮点数
else	程序语句	构成 if…else 选择结构
enum	数据类型说明	枚举
extern	存储种类说明	在其他程序模块中说明了的全局变量
float	数据类型说明	单精度浮点数
for	程序语句	构成 for 循环结构
goto	程序语句	构成 goto 转移结构
if	程序语句	构成 if…else 选择结构
int	数据类型说明	基本整型数
long	数据类型说明	长整型数
register	存储种类说明	使用 CPU 内部寄存的变量
return	程序语句	函数返回
short	数据类型说明	短整型数
signed	数据类型说明	有符号数，二进制数据的最高位为符号位
sizeof	运算符	计算表达式或数据类型的字节数
static	存储种类说明	静态变量
struct	数据类型说明	结构类型数据
swicth	程序语句	构成 switch 选择结构
typedef	数据类型说明	重新进行数据类型定义
union	数据类型说明	联合类型数据
unsigned	数据类型说明	无符号数数据
void	数据类型说明	无类型数据
volatile	数据类型说明	该变量在程序执行中可被隐含地改变
while	程序语句	构成 while 和 do…while 循环结构

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

Keil Cx51 中的关键字除了有 ANSI C 标准的 32 个关键字外，还根据 51 单片机的特点扩展了相关的关键字，如表 5-2 所列。

表 5-2　Keil Cx51 编译器扩展的关键字

关键字	用　途	说　明
bit	位标量声明	声明一个位标量或位类型的函数
sbit	位标量声明	声明一个可位寻址变量
Sfr	特殊功能寄存器声明	声明一个特殊功能寄存器
Sfr16	特殊功能寄存器声明	声明一个 16 位的特殊功能寄存器
data	存储器类型说明	直接寻址的内部数据存储器
bdata	存储器类型说明	可位寻址的内部数据存储器
idata	存储器类型说明	间接寻址的内部数据存储器
pdata	存储器类型说明	分页寻址的外部数据存储器
xdata	存储器类型说明	外部数据存储器
code	存储器类型说明	程序存储器
interrupt	中断函数说明	定义一个中断函数
reentrant	再入函数说明	定义一个再入函数
using	寄存器组定义	定义芯片的工作寄存器

其实，在 Keil Cx51 的文本编辑器中编写 C 程序，系统可以把保留字以不同颜色显示，缺省颜色为天蓝色。

第二节　数据类型

任何程序设计都离不开对于数据的处理，一个程序如果没有数据，它就无法工作。C 语言的数据类型可分为基本类型、构造类型、指针类型和空类型四类，具体分类情况如图 5-1 所示。

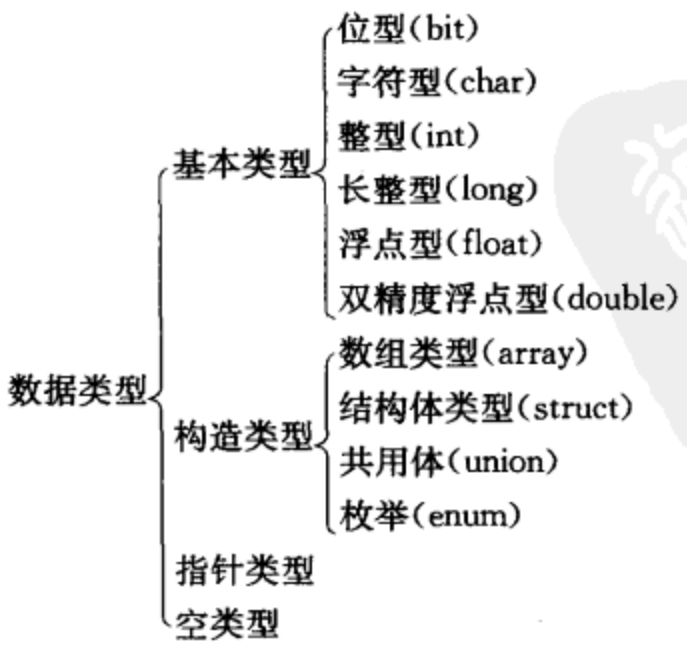


图 5-1　数据类型分类

在标准 C 语言中基本的数据类型为 char、int、short、long、float 和 double，而在 Cx51 编译器中，int 和 short 相同，float 和 double 相同。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

一、数据类型简介

Keil uVision2 Cx51 编译器所支持的数据类型如表 5-3 所列。

表 5-3　Keil uVision2 Cx51 编译器所支持的数据类型

数据类型	名称	长　度	值　域
unsigned char	无符号字符型	单字节	0～255
signed char	有符号字符型	单字节	－128～＋127
unsigned int	无符号整型	双字节	0～65535
signed int	有符号整型	双字节	－32768～＋32767
unsigned long	无符号长整型	4 字节	0～4294967295
signed long	有符号长整型	4 字节	－2147483648～＋2147483647
float	浮点型	4 字节	±1.175494E－38～±3.402823E＋38
*	指针型	1～3 字节	对象的地址
bit	位类型	位	0 或 1
sfr	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	可寻址位	位	0 或 1

1. char 字符类型

char 类型的长度是一个字节，通常用于定义处理字符数据的变量或常量。分无符号字符类型 unsigned char 和有符号字符类型 signed char，默认值为 signed char 类型。unsigned char 类型用字节中所有的位来表示数值，可以表达的数值范围是 0～255。signed char 类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，负数用补码表示。所能表示的数值范围是－128～＋127。unsigned char 常用于处理 ASCII 字符以及小于或等于 255 的整型数。

2. int 整型

int 整型长度为两个字节，用于存放一个双字节数据。分有符号整型数 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。unsigned int 表示的数值范围是 0～65535。

3. long 长整型

long 长整型长度为 4 个字节，用于存放一个 4 字节数据。分有符号长整型 signed long 和无符号长整型 unsigned long，默认值为 signed long 类型。signed long 表示的数值范围是－2147483648～＋2147483647，字节中最高位表示数据的符号，“0”表示正数，“1”表示负数。unsigned long 表示的数值范围是 0～4294967295。

4. 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。

5. * 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

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

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 Cx51 中，它的长度一般为 1 个～3 个字节。

6. bit 位类型

bit 位类型是 Cx51 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。

7. sfr 特殊功能寄存器

sfr 也是 Cx51 扩充的数据类型，占用一个内存单元，值域为 0～255。利用它可以访问 51 单片机内部的所有特殊功能寄存器。

定义方法如下：sfr 特殊功能寄存器名＝地址常数

例如：sfr P1＝0x90；这一句定义 P1 为 P1 端口在片内的寄存器，在后面的语句中用以用 P1 ＝ 0xff(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。

注意事项 在关键字 sfr 后面必须跟一个标识符作为寄存器名，名字可任意选取，但应符合一般习惯。等号后面必须是常数，不允许有带运算符的表达式，而且该常数必须在特殊功能寄存器的地址范围之内(0x80～0xff)。

8. sfr16 16 位特殊功能寄存器

在新一代的 8051 单片机中，特殊功能寄存器经常组合成 16 位来使用。采用关键字 sfrl6 可以定义这种 16 位的特殊功能寄存器。sfr16 也是 Cx51 扩充的数据类型，占用两个内存单元，值域为 0～65535。

例如，对于 8052 单片机的定时器 T2，可采用如下的方法来定义：

```
sfrl6 T2＝0xCC;  //定义 TIMER2，其地址为 T2L＝0xCC，T2H＝0xCD
```

这里 T2 为特殊功能寄存器名，等号后面是它的低字节地址，其高字节地址必须在物理上直接位于低字节之后。这种定义方法适用于所有新一代的 8051 单片机中新增加的特殊功能寄存器。

9. sbit 可寻址位

在 8051 单片机应用系统中，经常需要访问特殊功能寄存器中的某些位，Keil Cx51 编译器为此提供了一个扩充关键字 sbit，利用它定义可位寻址对象。定义方法有如下三种。

(1)sbit 位变量名＝位地址

这种方法将位的绝对地址赋给位变量，位地址必须位于 0x80～0xff 之间。例如：

```
sbit OV＝0xD2;
sbit CY＝0xD7;
```

(2)sbit 位变量名＝特殊功能寄存器名^位置

当可寻址位位于特殊功能寄存器中时可采用这种方法，“位位置”是 0～7 之间的一个常数。例如：

```
sfr PSW＝0xD0;
sbit OV＝PSW^2;
sbit CY＝PSW^7;
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

(3)sbit 位变量名＝字节地址^位位置

这种方法以一个常数(字节地址)作为基地址,该常数必须在 0x80～0xff 之间。“位位置”是 0～7 之间的一个常数。例如:

```
sbit OV=0xD0^2;
sbit CY=0xD0^7;
```

sbit 是一个独立的关键字,不要将它与关键字 bit 相混淆。关键字 bit 是 Keil Cx51 编译器的一种扩充数据类型,用来定义一个普通位变量,它的值是二进制数的 0 或 1。

二、实验演练

现在来写个小程序,看看 unsigned char 和 unsigned int 用于延时的不同效果,说明它们的长度是不同的。

实验:用下载型实验板做以下实验:在 Keil Cx51 中,先将以下源程序中的变量 i、j 改为 int 整符数据,观察 P1.0 脚外接灯的延时效果。再将程序中的变量 i、j 改为 char 字符型数据,观察 P1.0 脚外接灯的延时效果。

实验源程序如下:

```
#include<reg51.h>
sbit P10=P1^0;                    //定义位变量
void Delay(unsigned int i)        //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)                 //变量 i 由实际参数传入一个值,因此 i 不能赋初
                                    值
    for(j=0;j<125;j++)
    {;}
}
void main()
{
    for(;;)
    { P10=! P10;                  //将 P10 取反
    Delay(1000);                  // 将实际参数 1000 传递给形式参数 i,
    }
}
```

主函数中,有一行是 P10＝! P10,在 P10 前有一个“!”,“!”是 C 语言的一个运算符,意义是取反,即该符号后面的那个变量的值取反。所谓取反只是对变量的值而言的,并不会自动改变变量本身,可以认为 C 编译器在处理“! P10”时,将 P10 的值给了一个临时变量,然后对这个临时变量取反,而不是直接对 P10 取反,因此取反完毕还要使用赋值符号(＝)将取反后的值再赋给 P10,这样,如果原来 P10 是低电平(LED 亮),那么取反后,P10 就是高电平(LED 灭);反之,如果 P10 是高电平,取反后,P10 就是低电平,这条指令被反

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

复地执行，接在 P10 上的灯就会不断地“亮”、“灭”。该条指令会被反复执行的关键就在于 main 中有一行无限循环语句：for(;;)。

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 LED1.c。对程序进行编译、链接和调试，产生 LED1.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target'，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 键进行全速执行。观察实验板上 P1.0 脚外接灯闪烁的快慢。

实验中会发现，若将程序中的变量 i、j 改为 char 字符型数据，则灯的闪烁速度明显加快，即延时时间变短了许多。

该实验程序在本书附赠光盘的 ch_5/led1 文件夹中。

尽管这个程序实际意义不大，但它可以帮助理解 int 和 char 数据类型的使用方法。

这里必须说明的是，当定义一个变量为特定的数据类型时，在程序使用该变量时不应使它的值超过数据类型的值域。如本例中的变量 i、j 若定义为 unsigned char，则其值不能超出 0～255 的值。

第三节　常　　量

先看一段程序代码片段。省略号表示可能会有的其他操作。

```
int a = 3;
.....
a = 100;
```

程序中，a 是变量。一开始初始化为 3。后来出于需要，又将它赋值为 100。a 的值也就从 3 变成了 100。

程序中，3 和 100 就是一种常量。像这种直接在代码写出大小的量，称为立即数，也称为常数，而常数是常量的一种。一般来说，常数或代表固定不变值的名字称为常量。

一、常量的数据类型

常量是在程序运行过程中不能改变值的量，常量的数据类型有整型、浮点型、字符型、字符串型和位类型。

1. 整型常量

整型常量可以用十进制、八进制和十六进制表示。至于二进制形式，虽然它是计算机中最终的表示方法，但它太长，所以 C 语言不提供用二进制表达常数的方法。

用十进制表示，是最常用也是最直观的。如：7356，－90，等等。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

八进制用数字 0 开头(注意不是字母 O),例如 010、016 等。

十六进制以数字 0+小写字母 x 或大写字母 X 开头),例如 0x10、0Xf 等。注意,十六进制数只能用合法的十六进制数字表示,字母 a、b、c、d、e、f 也可以大写。

另外,长整型常量的表示方法是在数字后面加字母 L,如 104L、034L 等。

2. 浮点型常量

浮点型常量可分为十进制和指数表示形式。十进制由数字和小数点组成,如 0.888、3345.345、0.0 等。指数表示形式为:[±]数字[. 数字]e[±]数字,[]中的内容为可选项,根据具体情况可有可无,但其余部分必须有,如 125e3、7e9、-3.0e-3。

3. 字符型常量

字符型常量是单引号内的字符,如'a','d'等,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。常用转义字符如表 5-4 所列。

表 5-4　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	0AH/10
\r	回车符(CR)	0DH/13
\t	水平制表符	09H/9
\b	退格符	08H/8
\f	换页符	0CH/12
\'	单引号	27H/39
\"	双引号	22H/34
\\	反斜杠	5CH/92

4. 字符串型常量

字符串型常量由双引号内的字符组成,如"test"、"OK"等。当引号内没有字符时,为空字符串。在 C 语言中,系统在每个字符串的最后自动加入一个字符'/0'作为字符串的结束标志。请注意字符常量和字符串常量的区别,例如'\101、'Z'是字符常量,在内存中占一个字节,而"ABC\n"、"Z"是字符串常量,前者占五个字节、后者占两个字节的存储空间,其中一个字节用来存放'\0',两个连续的双引号""也是一个字符串常量,称作"空串",但要占一个字节的存储空间来存放'\0'。

5. 位类型

位类型的值是一个二进制。

二、用宏表示常数

假如要写一个有关圆的计算程序,那么 π(3.14159)值会被频繁地用到。由于 π 值不可能改变,所以应该将它当成一个常量对待。为了避免一遍遍地写 3.14159 这一长串的数而发生错误,可以采用宏。宏不仅可以用来代替常数值,还可以用来代替表达式,甚至是代码段。下面介绍其代替常数值的功能。

宏的语法为:

#define 宏名称 宏值

比如要代替前面说到的 π 值,应为:

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

#define PAI 3.14159

注意，宏定义不是C语言严格意义上的语句，所以其行末不用加分号结束。

有了上面的语句，程序中凡是要用到3.14159的地方都可以使用PAI这个宏来取代。作为一种建议和大家的习惯，宏名称经常使用全部大写的字母。

对宏的处理，在编译过程中称为“预处理”。也就是说在正式编译前，编译器必须先将代码出现的宏，用其相应的宏值替换，这个过程有点类似在文字处理软件中的查找替换。完成预处理后，所有原来的“PAI”都成了立即数3.14159。

三、常量定义

常量定义的格式为：

const 数据类型 常量名 = 常量值；

常量定义必须以const开始；另外，常量必须在定义的同时，完成赋值。

const float PAI = 3.14159；

const的作用就是指明这个量(PAI)是常量，而非变量。

常量必须一开始就指定一个值，然后，在以后的代码中，不允许改变PAI的值，比如：

如果一个常量是整型，可以省略指出数据类型，如：

const k = 100；

相当于

const int k = 100；

反过来说，如果不指定数据类型，则编译器会将它当成整型。比如：

const k = 1.234；

虽然想让k等于一个实型数，然而，最终k的值其实是1。因为编译器把它当成整型常量了。

因此建议在定义变量时，明确指出类型，不管它是整型或其他类型。

第四节 变 量

在程序运行过程中，其值可以被改变的量称为变量。变量有两个要素，一是变量名，每个变量都必须有一个名字——变量名，变量命名遵循标识符命名规则；二是变量值，在程序运行过程中，变量值存储在内存中，在程序中，通过变量名来引用变量的值。

一、变量的初始化

int a；

声明了一个整型变量a，但这变量的值是多少？a等于0吗？还是等于100？都不知道。“不知道”的意思是：a有值，但这个值的大小是随机的，所以无法确定。无法确定一

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

个变量值是常有的事。但有些时候，出于某种需要，需要事先给一个变量赋初值。那么，如果为一个变量赋值呢？

为变量赋初值一般用“=”进行赋值。需要说明的是，这里的“=”和数学中的“等号”并不相同，这里不是等同关系，而是赋值操作。

例如：

```
int a = 0;
```

其作用是将 0 赋予 a，让 a 的值初始化为 0。

定义多个变量时也一样。

例如：

```
int a = 0,b= 1;
```

当然也可以根据需要，只给部分变量初始化。

例如：

```
int a = 0,b;
```

或：

```
int a,b=1;
```

也可以在定义以后赋值。

例如：

```
int a;
a=100;
```

二、变量的定义及存储模式

在 C 语言中，要求对所有用到的变量，必须先定义、后使用；在定义变量的同时，也可以对变量进行初始化。

定义一个变量的格式如下：

[存储种类] 数据类型 [存储器类型] 变量名表

在定义格式中，除了数据类型和变量名表是必要的，其他都是可选项。

存储种类有四种：自动(auto)、外部(extern)、静态(static)和寄存器(register)，缺省类型为自动(auto)。

变量的数据类型和前面介绍的各种数据类型的定义是一样的。需要说明的是，编程时，为了书写方便，经常使用简化形式来定义变量的数据类型，其方法是在源程序的开头使用 #difine 语句。

例如：

```
#define uchar unsigned char
#define uint unsigned int
```

这样，在以后编程时，就可以使用 uchar 代替 unsigned char，用 uint 代替 unsigned int

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

来定义变量。

说明了一个变量的数据类型后，还可选择说明该变量的存储器类型。存储器类型的说明就是指定该变量在 Cx51 硬件系统中所使用的存储区域，并在编译时准确地定位。表 5-5 中是 Keil uVision2 所能认别的存储器类型。

表 5-5　存储器类型

存储器类型	说　明
data	直接访问内部数据存储器(128 字节)，访问速度最快
bdata	可位寻址内部数据存储器(16 字节)，允许位与字节混合访问
idata	间接访问内部数据存储器(256 字节)，允许访问全部片内地址
pdata	分页访问外部数据存储器(256 字节)，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

如果省略存储器类型，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。需要指出的是，变量的存储种类与存储器类型是完全无关的。

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，对这种变量的访问速度最快，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。

COMPACT 存储模式中，变量被定义在分页寻址的片外数据存储器中，每一页片外数据存储器的长度为 256 字节，这时，对变量的访问通过寄存器间接寻址(MOVX @Ri)进行的，堆栈位于片内数据存储器中。采用这种编译模式时，变量的高 8 位地址由 P2 口决定，低 8 位地址由 R0 或 R1 的内容决定。

LARGE 存储模式中，所有函数和过程的变量、局部数据段都定位在 8051 系统的外部数据区，最多可有 64KB，使用 DPTR 数据指针访问数据。这种访问数据的效率是不高的。

重点提示　C 语言中还有一个用以重新定义数据类型的语句，这个语句就是 typedef，例如以下语句：

```
typedef int integer;
integer a,b;
```

这两句在编译时，其实是先把 integer 定义为 int，在以后的语句中遇到 integer 就用 int 置换，integer 就等于 int，所以 a、b 也就被定义为 int。typedef 不能直接用来定义变量，它只是对已有的数据类型作一个名字上的置换，并不产生一个新的数据类型。下面两句就是一个错误的例子：

```
typedef int integer;
integer = 100;
```

使用 typedef 可以方便程序的移植和简化较长的数据类型定义。用 typedef 还可以定义结构类型，其语法是

typedef 已有的数据类型　新的数据类型名

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

三、变量的种类

变量主要分为局部变量、全局变量两种。

1. 局部变量

局部变量的有效范围只有在它所声明的函数内部有效，局部变量被声明在一个函数之中，例如：

```
main()
{                          有效范围
    int i=10;      /* 局部变量 i,其有效范围只有 main 函数内 */
    i=i+1;
        ⋮
}
void delay(void)
{                          有效范围
int i=11;/* 局部变量 i,基有效范围只有在 delay 函数内 */
i=i+1;
}
```

main()中的变量 i 与 delay()中的变量 i 为两个不相干的变量。

2. 全局变量

全局变量的有效范围在整个文件内(*.c)，文件中可能包含很多的函数，而全局变量在该文件内的任何函数均有效，但在不同的文件中无效，全局变量被声明在所有函数之外。例如：

```
  char delay;全局变量声明
main(  )                          变量 delay 的有效范围
  {
    delay=10;
  }
void f1(void)
  {
    delay=delay+1;
  }
void f2(void)
  {
    delay=delay*10;
  }
void f2(void)
  {
    delay=delay+10;
  }
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

四、实验演练

前面介绍了常量与变量，下面写个流水灯的程序来实验一下。这个程序简单易懂，可以说明常量与变量的基本使用用法。

```
#include<reg51.h>
void Delay(unsigned int i)              //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)                       //变量 i 由实际参数传入一个值,因此 i 不能赋
                                          初值
    for(j=0;j<125;j++)
    {;}
}
void main()
{
    for(;;)
    { P1=0xff;                          //P1 口灯全灭
    Delay(500);                         // 延时
    P1=0xfe;                            //点亮 P1.0 灯
    Delay(500);                         // 延时
    P1=0xfd;                            //点亮 P1.1 灯
    Delay(500);                         // 延时
    P1=0xfb;                            //点亮 P1.2 灯
    Delay(500);                         // 延时
    P1=0xf7;                            //点亮 P1.3 灯
    Delay(500);                         //延时
    P1=0xef;                            //点亮 P1.4 灯
    Delay(500);                         // 延时
    P1=0xdf;                            //点亮 P1.5 灯
    Delay(500);                         // 延时
    P1=0xbf;                            //点亮 P1.6 灯
    Delay(500);                         // 延时
    P1=0x7f;                            //点亮 P1.7 灯
    Delay(500);                         // 延时
    }
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 LED2.c。对程序进行编译、链接和调试，

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

产生 LED2. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target ',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 键进行全速执行。观察实验板上 P1 口外接灯的工作情况。

实验中会发现,P1 口外接的 8 只灯依次点亮,呈流水灯闪亮。

该实验程序在本书附赠光盘的 ch_5/led2 文件夹中。

读者还可以通过更改 P1 口的数据,来改变 P1 的输出状态,使 P1 口的外接灯呈不同的花样进行显示。

第五节　运算符和表达式

运算符就是完成某种特定运算的符号。运算符按其表达式中与运算符的关系可分为单目运算符、双目运算符和三目运算符。单目就是指需要有一个运算对象,双目就要求有两个运算对象,三目则要三个运算对象。表达式则是由运算及运算对象所组成的具有特定含义的式子。C 语言是一种表达式语言,表达式后面加“;”号就构成了一个表达式语句。

一、赋值运算符及其表达式

对于“=”这个符号,大家一定不会陌生,在 C 语言中,它的功能是给变量赋值,称之为赋值运算符。它的作用就是把数据赋给变量。如“x=10;”,利用赋值运算符将一个变量与一个表达式连接起来的式子为赋值表达式,在表达式后面加";"便构成了赋值语句。使用“=”的赋值语句格式如下:

变量=表达式;

例如:

```
a=0xFF;          //将常数十六进制数 FF 赋予变量 a
b=c=33;          //同时赋值给变量 b,c
d=e;             //将变量 e 的值赋予变量 d
f=a+b;           //将变量 a+b 的值赋予变量 f
```

由上面的例子可以知道,赋值语句的意义就是先计算出“=”右边的表达式的值,然后将得到的值赋给左边的变量。而且右边的表达式可以是一个赋值表达式。

注意事项　初学者在编写调试程序时,容易忽略了分号“;”,或在程序中加入了全角符号,运算符打错、漏掉,这样会造成程序不能被正常地编译。

二、算术运算符及其表达式

Cx51 有以下几种算术运算符,如表 5-6 所列。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

表 5-6　算术运算符的功能

算术运算符号	功　能	算术运算符号	功　能
+	加法	++	自增 1
−	减法	−−	自减 1
*	乘法	%	求余
/	除法		

除法运算符和一般的算术运算规则有所不同，如是两浮点数相除，其结果为浮点数，如 10.0/20.0 所得值为 0.5，而两个整数相除时，所得值就是整数，如 7/3，所得值为 2。

求余运算符的对象只能是整型，在%运算符左侧的运算数为被除数，右侧的运算数为除数，运算结果是两数相除后所得的余数。

自增和自减运算符的作用是使变量自动加 1 或减 1。自增和自减符号放在变量之前和之后是不同的。

++i，−−i：在使用 i 之前，先使 i 值加(减)1。

i++，i−−：在使用 i 之后，再使 i 值加(减)1。

粗略地看，++i 和 i++的作用都相当于 i=i+1，但++i 和 i++的不同之处在于++i 先执行 i=i+1，再使用 i 的值；而 i++则是先使用 i 的值，再执行 i=i+1。

例如：若 i=5，则执行 y=++i 时，先使 i 加 1，即 i=i+1=6，再引用结果，即 y=6。运算结果为 i=6，y=6。

再如：若 i=5，则执行 y=i++时，先引用 i 值，即 y=5，再使 i 加 1，即 i=i+1=6。运算结果为 i=6，y=5。

重点提示　C 语言规定了算术运算符的优先级和结合性。

优先级指当运算对象两侧都有运算符时，执行运算的先后次序。按运算符优先级别高低顺序执行运算。

结合性指当一个运算对象两侧的运算符的优先级别相同时的运算顺序。

各种运算符的优先级和结合性见附录一。

三、关系运算符及其表达式

关系运算符用来比较变量的值或常数的值，并将结果返回给变量。C 语言有 6 种关系运算符，如表 5-7 所列。

表 5-7　关系运算符

关系运算符	功　能	关系运算符	功　能
>	大于	<=	小于等于
>=	大于等于	==	等于
<	小于	!=	不等于

前四个具有相同的优先级，后两个也具有相同的优先级，但是前四个的优先级要高于后两个。

当两个表达式用关系运算符连接起来时，这时就是关系表达式。关系表达式通常是用来判别某个条件是否满足。要注意的是，用关系运算符的运算结果只有 0 和 1 两种，也就是逻辑的真与假，当指定的条件满足时结果为 1，不满足时结果为 0。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

例如：unsigned char i=10,j=20;

k=(i>j)　结果：k=0
k=(i==j) 结果：k=0
k=(i<j)　结果：k=1

四、逻辑运算符及其表达式

关系运算符所能反映的是两个表达式之间的大小等于关系，逻辑运算符则是用于求条件式的逻辑值。C语言有三种逻辑运算符，如表5-8所列。

表5-8　逻辑运算符

逻辑运算符	功　能
&&	逻辑与
||	逻辑或
!	逻辑非

用逻辑运算符将关系表达式或逻辑量连接起来就是逻辑表达式。逻辑表达式的一般形式为：

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。

进行逻辑与运算时，首先对条件式1进行判断，如果为真(非0值)，则继续对条件式2进行判断，当结果为真时，逻辑运算的结果为真(值为1)，如果结果不为真时，逻辑运算的结果为假(0值)。如果在判断条件式1时就不为真的话，就不用再判断条件式2了，而直接给出运算结果为假。

逻辑或，是指只要两个运算条件中有一个为真时，运算结果就为真，只有当条件式都不为真时，逻辑运算结果才为假。

逻辑非则是把逻辑运算结果值取反，如果条件式的值为真，进行逻辑非运算后则结果变为假，条件式运算值为假时，逻辑结果为真。

同样，逻辑运算符也有优先级，从高到低依次是：！(逻辑非)→&&(逻辑与)→||(逻辑或)。可见，逻辑非的优先级最高。

例如：

```
char a,b,c
char i=4,j=2,k=3
a=(i>j)&&(j>k)    结果：(i>j)为真，(j>k)为假，所以 a=0
b=(i>j)||(j>k)    结果：(i>j)为真，(j>k)为假，所以 b=1
c=!(i>j)          结果：(i>j)为真，所以 c=0
```

五、位运算符及其表达式

学过汇编的朋友都知道汇编对位的处理能力是很强的，C语言也能对运算对象进行

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

按位操作，从而使C语言也能具有一定的对硬件直接进行操作的能力。位运算符的作用是按位对变量进行运算，但是并不改变参与运算的变量的值。如果要求按位改变变量的值，则要利用相应的赋值运算。还有就是位运算符是不能用来对浮点型数据进行操作的。C51中共有6种位运算符，如表5-9所列。

表5-9　位运算符

位运算符	功　能	位运算符	功　能
&	按位与	~	按位取反
|	按位或	>>	右移
^	按位异或	<<	左移

位运算一般的表达形式如下：

变量1 位运算符 变量2

位运算符也有优先级，从高到低依次是：~(按位取反)→<<(左移)→>>(右移)→&(按位与)→^(按位异或)→|(按位或)。

表5-10是位逻辑运算符的真值表，a表示变量1，b表示变量2。

表5-10　位逻辑运算符的真值表

a	b	~a	~b	a&b	a|b	a^b
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0

位运算符中的移位操作比较复杂，左移(<<)运算符是用来将变量1的二进制位值向左移动由变量2所指定的位数。

例如：

a=Ox8f(即二进制数10001111)，进行左移运算a<<2，就是将a的全部二进制位值一起向左移动2位，其左端移出的位值被丢弃，并在其右端补以相应位数的“0”。因此，移位的结果是a=Ox3c(即二进制数00111100)。

右移(>>)运算符是用来将变量1的二进制位值向右移动由变量2指定的位数。进行右移运算时，如果变量1属于无符号类型数据，则总是在其左端补“0”；如果变量1属于有符号类型数据，则在其左端补入原来数据的符号位(即保持原来的符号不变)，其右端的移出位被丢弃。

对于a=Ox8f，如果a是无符号数，则执行a>>2之后结果为a=Ox23(即二进制数00100011)；如果a是有符号数，则执行a>>2之后结果为a=Oxe3(即二进制数11100011)；

六、复合赋值运算符及其表达式

复合赋值运算符就是在赋值运算符“=”的前面加上其他运算符。表5-11是C语言中的复合赋值运算符。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

表 5-11　复合赋值运算符

+=	加法赋值	>>=	右移位赋值
-=	减法赋值	&=	逻辑与赋值
*=	乘法赋值	|=	逻辑或赋值
/=	除法赋值	^=	逻辑异或赋值
%=	取模赋值	~=	逻辑非赋值
<<=	左移位赋值		

复合运算的一般形式为：

变量　复合赋值运算符　表达式

其含义就是变量与表达式先进行运算符所要求的运算，再把运算结果赋值给参与运算的变量。其实这是C语言中一种简化程序的方法，凡是二目运算都可以用复合赋值运算符去简化表达。例如：

a+=1 等价于 a=a+1

b/=a+2 等价于 b=b/(a+2)

很明显，采用复合赋值运算符会降低程序的可读性，但这样却可以使程序代码简单化，并能提高编译的效率。对于C语言初学者，在编程时最好还是根据自己的理解和习惯去使用程序表达的方式，不要一味追求程序代码的短小。

七、其他运算符及其表达式

1. 逗号运算符

在VB中，“Dim a,b,c”的逗号就是把多个变量定义为同一类型的变量，在C语言中也一样，如“int a,b,c”。

但是在C语言中，逗号还是一种特殊的运算符，也就是逗号运算符，可以用它将两个或多个表达式连接起来，形成逗号表达式。逗号表达式的一般形式为：

表达式1,表达式2,表达式3……表达式n

在程序运行时，是从左到右计算出各个表达式的值，而整个用逗号运算符组成的表达式的值等于最右边表达式的值，就是“表达式n”的值。需要注意的是，并不是在程序的任何位置出现的逗号，都可以认为是逗号运算符。如函数中的参数、同类型变量的定义中的逗号，只是用来间隔之用，而不是逗号运算符。

2. 条件运算符

C语言中有一个三目运算符，它就是“?:”，条件运算符，它要求有三个运算对象。它可以把三个表达式连接构成一个条件表达式。条件表达式的一般形式如下：

逻辑表达式? 表达式1 : 表达式2

条件运算符的作用简单来说就是根据逻辑表达式的值选择使用表达式的值。当逻辑表达式的值为真时(非0值)时，整个表达式的值为表达式1的值；当逻辑表达式的值为假(值为0)时，整个表达式的值为表达式2的值。要注意的是，条件表达式中逻辑表达式的类型可以与表达式1和表达式2的类型不一样。下面是一个逻辑表达式的例子。

如有a=1,b=2,要求取a、b两数中较小的值放入min变量中，可以这样写程序：

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
if (a<b)
min = a;
else
min = b; //这一段的意思是当 a<b 时,min 的值为 a 的值,否则为 b 的值。
```

用条件运算符去构成条件表达式就变得十分简单明了。

```
min = (a<b)? a : b
```

很明显,它的结果和含意都和上面的一段程序是一样的,但是代码却比上一段程序少很多,编译的效率也相对要高,但有着和复合赋值表达式一样的缺点,就是可读性较差。在实际应用时可以根据自己的习惯使用。

3. 指针和地址运算符

指针是C语言中一个十分重要的概念,也是学习C语言的一个难点。有关指针将会在后续章节中进行详述。在这里,先来了解C语言中提供的两个专门用于指针和地址的运算符:

* 取内容
& 取地址

取内容和地址的一般形式分别为:

```
变量 = * 指针变量
指针变量 = & 目标变量
```

取内容运算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。要注意的是,指针变量中只能存放地址(也就是指针型数据),一般情况下不要将非指针类型的数据赋值给一个指针变量。

4. sizeof 运算符

sizeof 是用来求数据类型、变量或是表达式的字节数的一个运算符,但它并不像"="之类运算符那样在程序执行后才能计算出结果,它是直接在编译时产生结果的。它的语法如下:

```
sizeof (数据类型)
sizeof (表达式)
```

下面是两句应用例句,程序大家可以试着编写一下。

```
printf("char 是多少个字节? %bd 字节\n",sizeof(char));
printf("long 是多少个字节? %bd 字节\n",sizeof(long));
```

结果是:

```
char 是多少个字节? 1 字节
long 是多少个字节? 4 字节
```

5. 强制类型转换运算符

两个不同数据类型的数在相互赋值时会出现不对的值,如下面的一段小程序:

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
void main(void)
{
unsigned char a;
unsigned int b;
b=100*4;
a=b;
while(1);
}
```

这段小程序并没有什么实际的应用意义，如果仔细研究，会发现 a 的值是不会等于 100 * 4 的。因为 a 和 b 一个是 char 类型，一个是 int 类型，从以前的学习可知，char 只占一个字节值，最大只能是 255。但编译时为何不出错呢？先来看看这程序的运行情况，如图 5-2 所示。

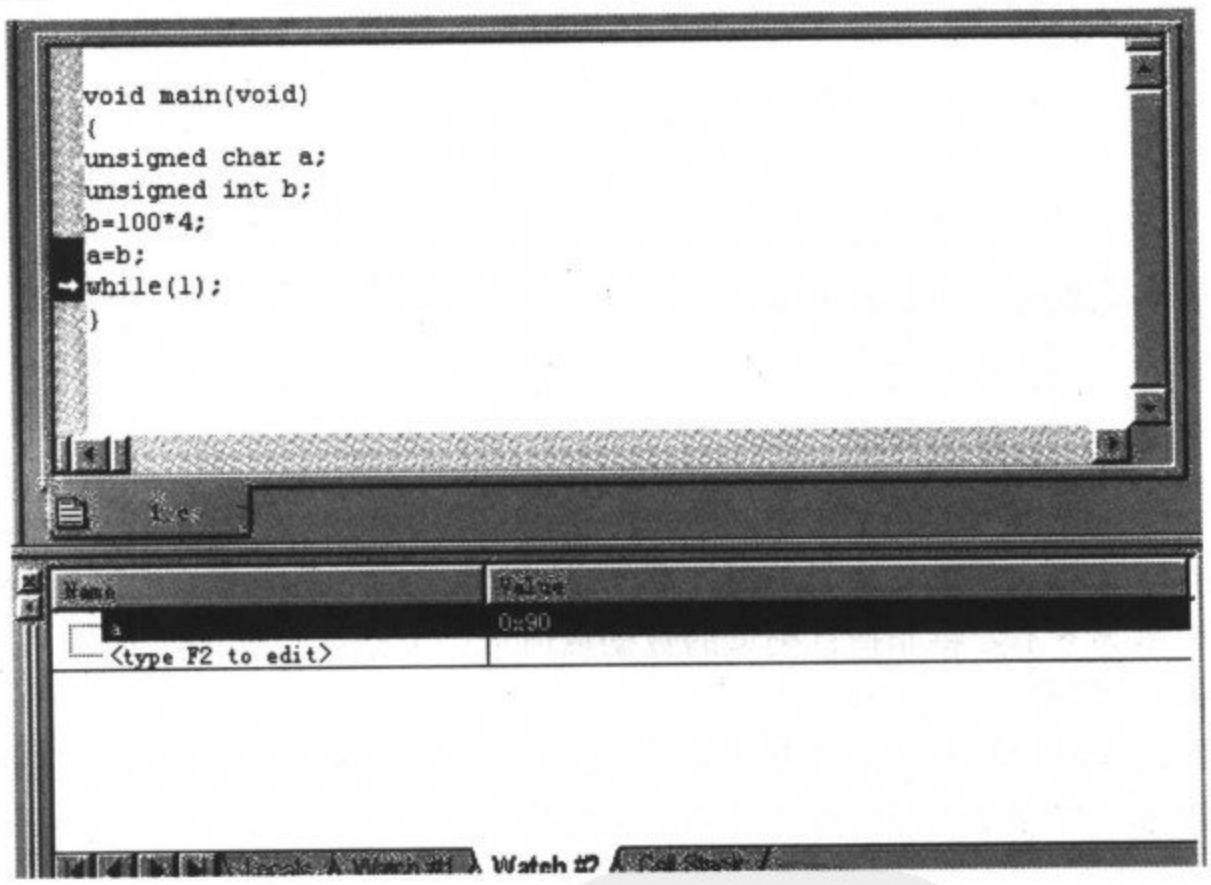


图 5-2　程序运行和变量观察窗口

由 b=100 * 4 就可以得知 b=0x190，这时可以在 Keil 的变量观察窗口中查看 a 的值。对于 watch 窗口，前面已学习过，在这个窗口中，可以在 watch 页中输入所要查看的变量名，对它的值进行查看。做法是按图中的 watch#1(或 watch#2)，然后光标移到图中的“type F2 edit”按 F2 键，这样就可以输入变量名了。在这里可以查看到 a 的值为 0x90，也就是 b 的低 8 位。这是因为执行了数据类型的隐式转换。

在 C 语言程序中进行算术运算时，需要注意数据类型的转换。有两种数据类型转换方式，即隐式转换和显式转换。隐式转换是在对程序进行编译时由编译器自动处理的。隐式转换遵循以下规则：

①所有 char 型的操作数转换成 int 型。

②用运算符连接的两个操作数如果具有不同的数据类型，按以下次序进行转换：如果一个操作数是 float 类型，则另一个操作数也转换成 float 类型；如果一个操作数是 long

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

类型，则另一个操作数也转换成 long 类型；如果一个操作数是 unsigned 类型，则另一个操作数也转换成 unsigned 类型。

③在对变量赋值时发生的隐式转换，将赋值号“＝”右边的表达式类型转换成赋值号左边变量的类型。例如，把整型数赋值给字符型变量，则整型数的高 8 位将丧失；把浮点数赋值给整型变量，则小数部分将丧失。在 C 语言中，只有基本数据类型（即 char、int、long 和 float）可以进行隐式转换，其余的数据类型不能进行隐式转换。例如，不能把一个整型数利用隐式转换赋值给一个指针变量，在这种情况下就必须利用强制类型转换运算符来进行显式转换。强制类型转换运算符的一般使用形式为：

（类型）＝表达式；

显式类型转换在给指针变量赋值时特别有用。例如，预先在 8051 单片机的片外数据存储器（xdata）中定义了一个字符型指针变量 px，如果想给这个指针变量赋一初值 OxB000，可以写成：px＝(char xdata ＊)0xB000。这种方法特别适合于用标识符来存取绝对地址。

八、实验演练

1. 输出和输入函数

在进行实验之前，先介绍一下 C 语言的输出和输入函数。

（1）输出函数 printf

printf 函数的全称是格式化输出函数，即按指定格式完成输出功能。它可以向屏幕输出字符、字符串、数值以及空行等，在中文操作系统下还可以输出汉字。输出的内容可以是执行结果，也可以是提示语。

printf 函数的一般调用格式如下：

printf(格式控制串，输出项表)

如果在 printf 函数之后加上“；”，就构成了输出语句。

例如：

```
printf("a＝%d,b＝%d",a,b);
```

其中，printf 是函数名，在圆括号中用双引号括进来的字符串，如"a＝%d,b＝%d"称为格式控制串，a、b 是输出项表中的输出项，它们都是 printf 函数的参数。

格式字符串的主要作用是为各输出项提供格式转换说明，将要输出的数据转换为指定的格式输出。它总是由“%”符号开始，紧跟其后的是格式描述符，如表 5-12 所列。

表 5-12　格式转换

式字符	说　明
%c	输出一个字符
%d	输出带符号十进制整数
%f	输出实型数字
%o	输出无符号八进制数
%s	输出字符串
%x	输出无符号十六进制数
%%	输出百分号

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

格式字符串的最后常跟转义字符，转义字符以反斜线开头，一些无法像一般字符那样可明确地写出来，如键盘上回车、退格之类的专用符号，在C语言程序中可用转义字符表示并输出，如\n表示回车换行等。

输出项表中的各输出项要用逗号隔开，输出项可以是合法的常量、变量或表达式。格式转换说明的个数要与输出项的个数相同，使用的格式描述符也要与它们一一对应且类型匹配。

(2)输入函数 scanf

scanf 函数是C语言提供的标准输入函数，它的作用是在终端设备上输入数据。scanf 函数的语法格式如下：

scanf("格式控制串"，变量地址列表)；

这里的"格式控制串"与 printf 函数中的"格式控制串"是一致的，主要用于对所输入数据的数据类型的控制。

"变量地址列表"，是在变量名前面加上一个"&"符号，表示该变量在计算机内存中的地址。

2. 串行调试软件

为了能够在电脑端看到单片机发出的数据，必须借助串口调试软件 comdebug。串口调试软件无需安装，可以直接在当前位置运行这个软件。软件界面如图 5-3 所示。

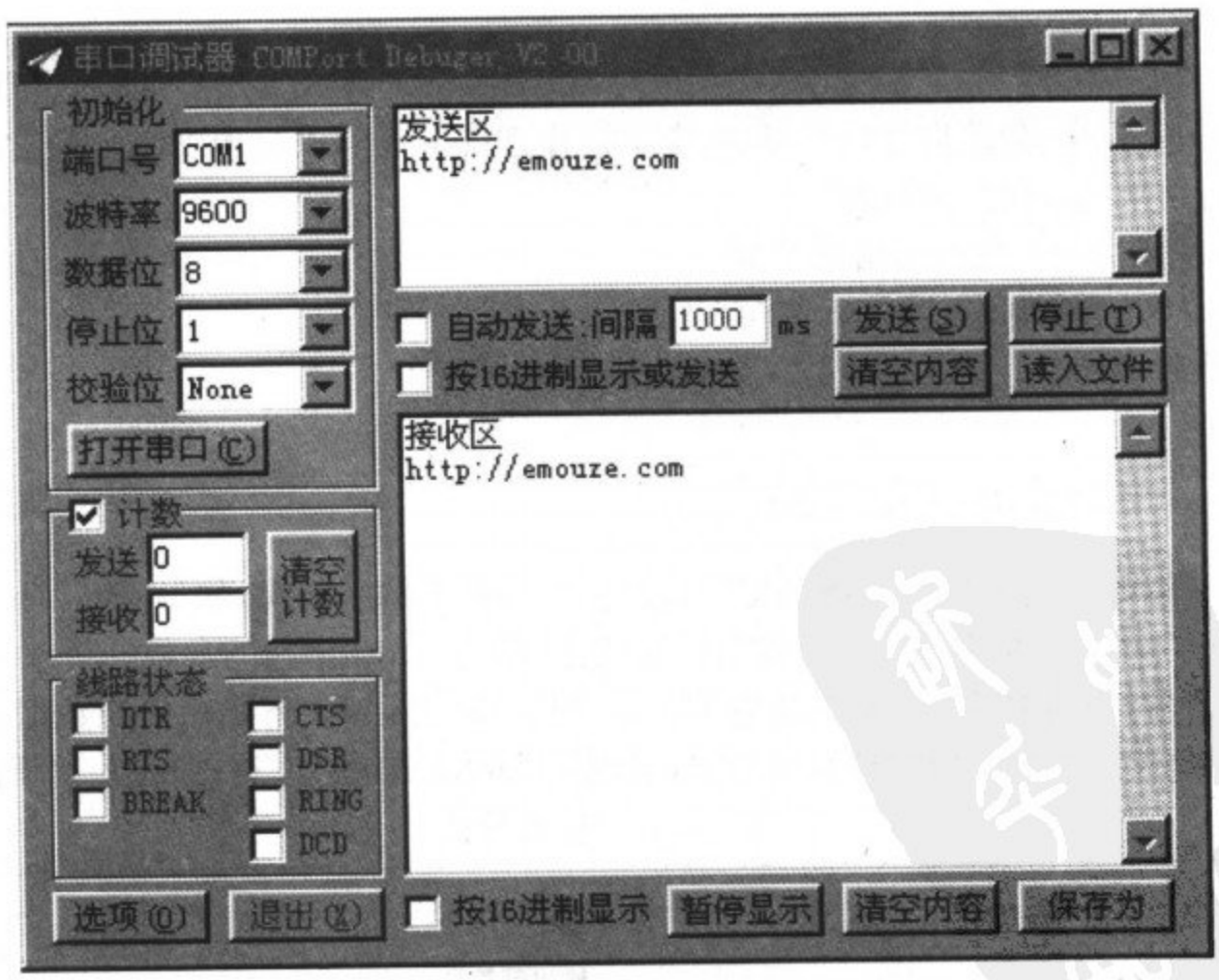


图 5-3　串口调试软件界面

3. 实验

实验 1：利用 AT89C51 实验板，验证关系运算符的使用，实验程序如下(该程序在附赠光盘中的 ch_5/guanxi 文件夹中)：

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
#include <AT89X51.h>
#include <stdio.h>                          //调用输入输出函数时,要求源文
                                              件中包含该头文件

void main(void)
{
  int x,y;
  SCON= 0x40;                               //串口方式 1
  REN=1;                                    //允许接收
  TMOD= 0x20;                               //定时器 1 定时方式 2
  TH1= 0xE6;                                //12MHz 1200 波特率
  TL1= 0xE6;
  TI= 1;
  TR1= 1;                                   //启动定时器
  while(1)
    {
    printf("您好! 我是 computer! \n");      //显示
    printf("请输入两个 int,X 和 Y\n");      //显示
    scanf("%d%d",&x,&y);                    //输入
    if (x<y)
      printf("x<y\n");                      //当 X 小于 Y 时
    else  if(x==y)                          //当 X 不小于 Y 时再做判断
      printf("x=y\n");                      //当 X 等于 Y 时
    else
      printf("x>y\n");                      //当 X 大于 Y 时
    }
}
```

该程序中,计数器的计数初值 X 可通过下列公式计算出:

$$X=256-\frac{2^{smod}\times f_{osc}}{384\times 波特率}$$

这里,SMOD=0,晶振 f_{osc} 为 12MHz,波特率为 1200b/s

所以 $$X=256-\frac{1\times 12\times 10^6}{384\times 1200}=230=0xE6$$

实验步骤如下:

① 打开 Keil 软件,输入上面的程序,保存为 guanxi.c。对程序进行编译、链接,产生 guanxi.hex 目标文件。

②取下 AT89C51 实验板上的 CPU,插入 RF-810 编辑器,对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接,将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件,将波特率调整为 1200,串口选择为 COM1,并

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

打开串口。

⑤打开 AT89C51 实验板电源，串口调试软件显示如图 5-4 所示，要求您输入数据。

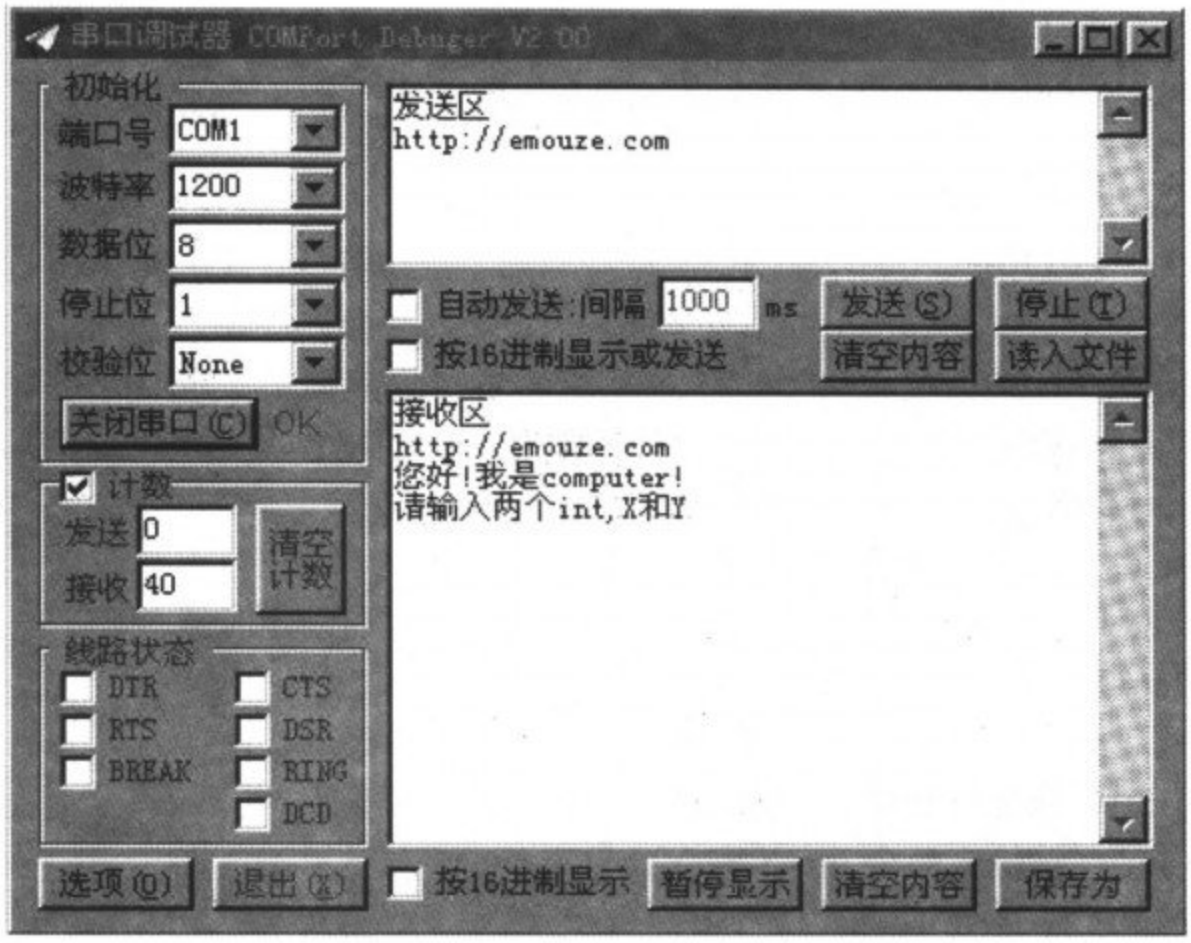


图 5-4　输入状态

⑥在串口调试软件的发送区输入第一个数据(设为 10)，按一下键盘上的回车键，以使 scanf 函数确认有数据输入，再按一下调试软件中的“发送”按钮，此时在调试软件的接收区会出现数据 10，如图 5-5 所示。

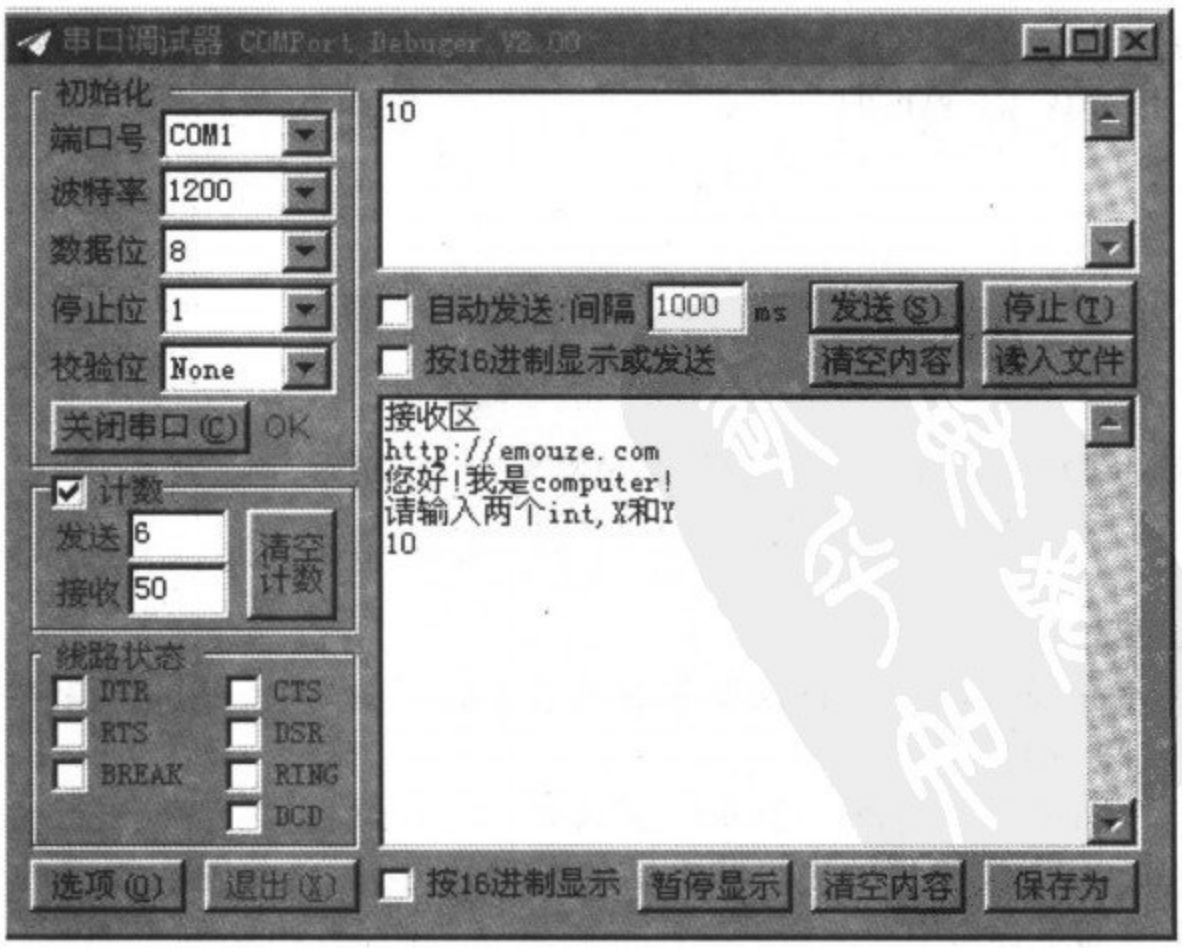


图 5-5　输入第一个数据 10

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

⑦将在串口调试软件的发送区的第一个数据删除，再输入第二个数据(设为 20)，按一下键盘上的回车键，以使 scanf 函数确认有数据输入，再按一下调试软件中的“发送”按钮，此时，在调试软件的接收区会出现数据 20，并给出判断结果，并处于等待输入状态，如图 5-6 所示。

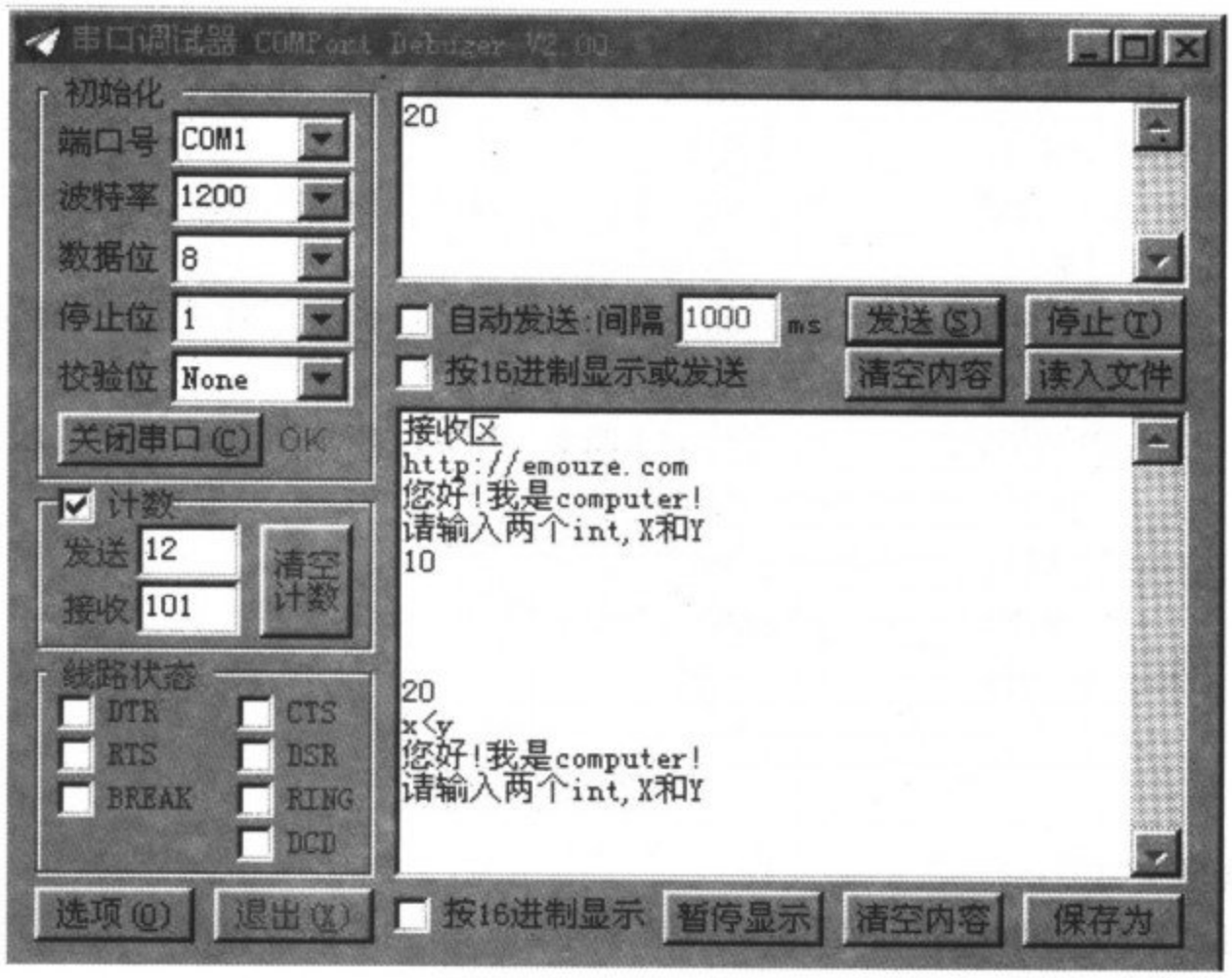


图 5-6　输出判断结果

实验 2：用下载型实验板做位运算符实验，要求 8 只 LED 灯先全部点亮，然后依次熄灭，再重复以上过程。实验程序如下：

```
#include <reg51.h>
void Delay(unsigned int i)      //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)               //变量 i 由实际参数传入一个值,因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
}
void main(void)
{
    while(1)
    {
    P1=0x0;                     //点亮 8 只 LED
    Delay(1000);                //延时
    P1= P1|0x01;                //熄灭 LED1
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    Delay(1000);              //延时
    P1= P1|0x3;               //熄灭 LED1、LED2
    Delay(1000);              //延时
    P1= P1|0x7;               //熄灭 LED1、LED2、LED3
    Delay(1000);              //延时
    P1= P1|0xF;               //熄灭 LED1、LED2、LED3、LED4
    Delay(1000);              //延时
    P1= P1|0x1F;              //熄灭 LED1、LED2、LED3、LED4、LED5
    Delay(1000);              //延时
    P1= P1|0x3F;              //熄灭 LED1、LED2、LED3、LED4、LED5、LED6
    Delay(1000);              //延时
    P1= P1|0x7F;              //熄灭 LED1、LED2、LED3、LED4、LED5、LED6、
                                LED7
    Delay(1000);              //延时
    P1=P1|0xFF;               //熄灭全部 LED
    Delay(1000);              //延时
    }
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 wei. c。对程序进行编译、链接和调试，产生 wei. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target'，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 键进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_5/wei 文件夹中。

实验 3：用下载型实验板做以下实验，8 只 LED 灯只有一只灯未点亮，且未点亮的灯向右逐位移动。实验程序如下：

```
#include <reg51. h>
void Delay(unsigned int i)    //延时程序，i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)             //变量 i 由实际参数传入一个值，因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
}
void main()
{
    unsigned char a,k;
    k=0x80;
    for(a=0;a<8;a++)
      {
        P1=k;
        Delay(1000);          //延时
        k=k>>1;
      }
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 youyi. c。对程序进行编译、链接和调试，产生 youyi. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 键进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_5/youyi 文件夹中。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第六章　Cx51 的基本语句

C 语言是一种结构化的程序设计语言，提供了相当丰富的程序控制语句。这些语句主要包括表达式语句、复合语句、选择语句和循环语句等。可以说语句是组成程序的灵魂。学习和掌握语句的用法是 C 语言学习中的重点。

第一节　表达式语句和复合语句

一、表达式语句

表达式语句是最基本的一种语句。不同的程序设计语言都会有不一样的表达式语句，如 VB 语言，就是在表达式后面加入回车构成 VB 的表达式语句；而在 51 单片机的 C 语言中，则是加入分号“;”构成表达式语句。

例如：

```
a=b*10;
i++;
```

这些都是合法的表达式语句。一些初学者往往在编写调试程序时忽略了分号“;”，造成程序无法被正常地编译。另外，在程序中加入了全角符号、运算符打错漏掉，也会造成程序不能被正常编译。

在 C 语言中有一个特殊的表达式语句，称为空语句，它仅仅是由一个分号“;”组成。有时候为了使语法正确，就要求有一个语句，但这个语句又没有实际的运行效果，那么这时就要有一个空语句。空语句通常会用以下两种用法：

①while，for 构成的循环语句后面加一个分号，形成一个不执行其他操作的空循环体。用它可以写等待事件发生的程序。例如，下面这段程序是读取 8051 单片机串行口数据的函数，其中就用了一个空语句 while(! RI);来等待单片机串行口接收结束。

```
read_com()              //函数定义
{
char a;                 //变量定义
while(! RI);            //等待 8051 串行口接收数据
a=SUBF;                 //读串行口内容
RI=0;                   //清除串行口接收标志
return(a);              //返回
}
```

需要注意的是，“;”号作为空语句使用时，要与语句中有效组成部分的分号相区分，如

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

for (;i<8;i++),第一个分号也应该算是空语句,它会使 i 赋值为 0(如果程序前有 i 值,则 i 的初值为 i 的当前值),for (;i<8;i++)就相当于 for (i=0;i<8;i++),书写时尽量用后面的写法,这样能使人更容易读明白。

②在程序中为有关语句提供标号,标记程序执行的位置,使相关语句能跳转到要执行的位置。这会用在 goto 语句中。

例如:

```
repeat:;
……
goto repeat;
```

二、复合语句

在 C 语言中,一对花括号“{}”不仅可用作函数体的开头和结尾标志,也可作为复合语句的开头和结尾的标志,复合语句也称为“语句块”,其形式如下:

{语句 1;语句;……;语句 n;}

复合语句之间用“{}”分隔,而它内部的各条语句还是需要以分号“;”结束。复合语句是允许嵌套的,也是就是在“{}”中的“{}”也是复合语句。复合语句在程序运行时,“{}”中的各行单语句是依顺序执行的。在 C 语言中,可以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。要注意的是,在复合语句中所定义的变量,称为局部变量。所谓局部变量就是指它的有效范围只在复合语句中,而函数也算是复合语句,所以函数内定义的变量有效范围也只在函数内部。

三、实验演练

实验:用 AT89C51 实验板做以下实验,验证表达式语句、复合语句的使用方法。

实验源程序如下:

```
#include <AT89x51.h>
#include <stdio.h>
void main(void)
  { unsigned int a,b,c,d;                    //这个定义会在整个 main 函数中
  SCON= 0x40;                                //串口方式 1
  REN=1;                                     //允许接收
  TMOD= 0x20;                                //定时器 1 定时方式 2
  TH1= 0xE6;                                 //12MHz 1200 波特率
  TL1=0xE6;
  TI=1;
  TR1=1;                                     //启动定时器
  a=5;b=6;c=7;d=8;                           //在整个主函数中有效
  printf("0: %d,%d,%d,%d\n",a,b,c,d);
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    {                                          //复合语句1
      unsigned int b,m;                        //只在复合语句1中有效
      b=10,m=100;
      printf("1: %d,%d,%d,%d,%d\n",a,b,c,d,m);
          {                                    //复合语句2
          unsigned int c,n;                    //只在复合语句2中有效
          c=11,n=200;
          printf("2: %d,%d,%d,%d,%d,%d\n",a,b,c,d,m,n);
          }                                    //复合语句2结束
      printf("1: %d,%d,%d,%d,%d\n",a,b,c,d,m);
      }                                        //复合语句1结束
  printf("0: %d,%d,%d,%d\n",a,b,c,d);
  while(1);
}                                              //主函数结束
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 yuju. c。对程序进行编译、链接，产生 yuju. hex 目标文件。

②取下 AT89C51 实验板上的 CPU，插入 RF-810 编辑器，对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接，将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件，将波特率调整为 1200，串口选择为 COM1，并打开串口。

⑤打开 AT89C51 实验板电源，串口调试软件运行结果如图 6-1 所示。

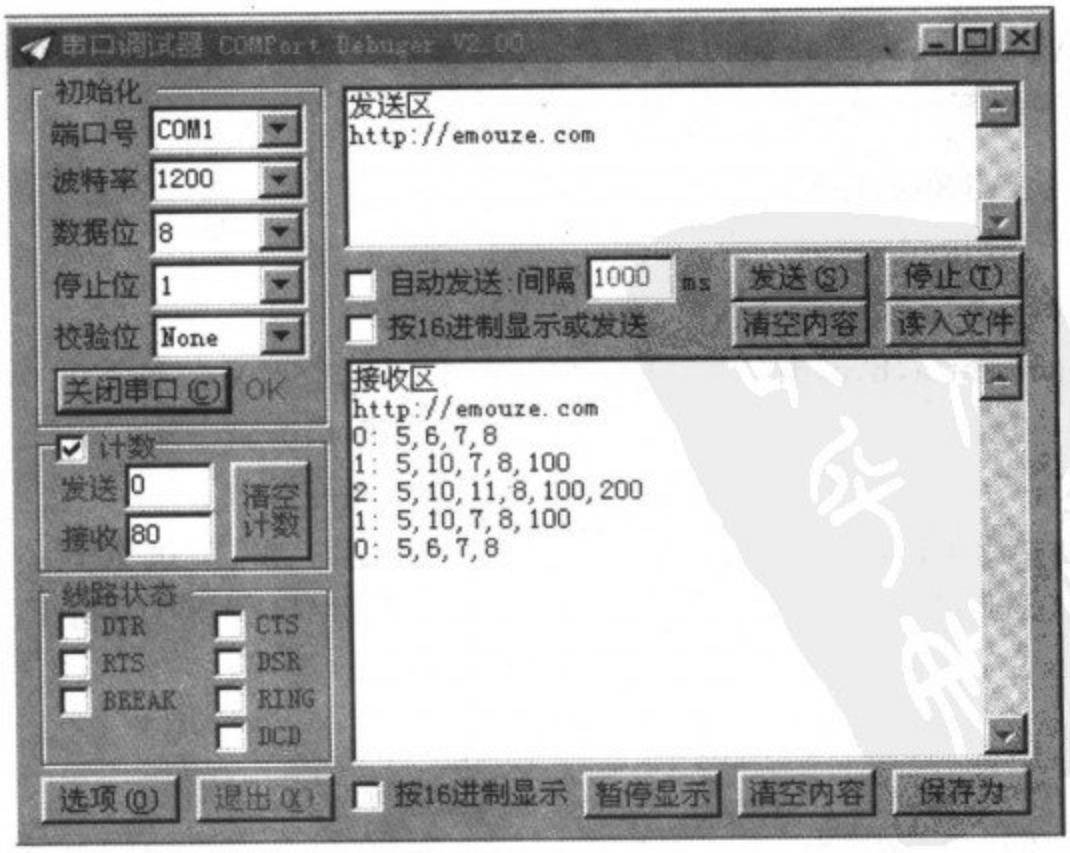


图 6-1　运行结果

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

在这个程序的主函数体开始处，定义了变量 a、b、c、d，它们在整个主函数体中都是有效的，在主函数体中的复合语句 1 和复合语句 2 中都可以使用它们。另外，在复合语句 1 中又定义了局部变量 m 和与主函数体中定义的变量同名的局部变量 b，这种局部变量 m 和 b 仅在定义它的复合语句 l 中有效，而且局部变量 b 的优先级高于在主函数体中定义的同名变量 b。因此在复合语句 1 中执行 printf 函数输出的 b 值为 10，而不是 6。同样，在复合语句 2 中定义了一个与主函数体中同名的局部变量 c，在复合语句 2 中执行 printf 函数输出的 c 值为 11，而不是 7。一旦出了复合语句，则其中的局部变量立即失效。如果有同名的局部变量，则恢复该变量在上一层位置所定义的初值。读者可以通过仔细分析本程序的执行结果，弄清各个变量的作用范围。

第二节　条件选择语句

一、if 语句及实验演练

if 条件语句又被称为分支语句，其关键字是由 if 构成。C 语言提供了 3 种形式的 if 条件语句。

1. if. . . else? 语句

if. . . else 语法格式：

```
if（条件表达式）
{
  语句 1
}
else
{
  语句 2
}
```

“语句 1”和“语句 2”表示当条件成立时和不成立时分别要执行的语句。用一句表达，就是：如果条件成立（也称条件为真），那么程序执行语句 1，否则（条件为假），程序执行语句 2。这就是 if. . . else 语句。

举一例子：

设 a，b，c 为已定义的三个 int 变量，有以下代码片段：

```
if(a>b)
{
  c=1;
}
else
{
  c=0;
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
}
```

若 a 值为 2，b 值为 1，那么执行上面代码，c 的值将为 1；相反，如果 a 值为 1，b 值为 1，由于条件 a＞b 不成立，所以程序将执行 else 后面一对{}中的代码，结果将是 c 值为 0。

2. if... 语句

if...else... 中的 else(否则)并不是必须的，因为有时候，当指定条件成立时，执行某些动作，否则，不执行那些动作。

if... 语句格式：

```
if (条件表达式)
{
  语句
}
```

该语句的执行过程是：如果条件成立(条件为真)，那么程序执行其后的 if 语句，然后执行 if 语句的下一条语句；如果条件不成立(条件为假)，则跳过 if 语句，直接执行 if 语句的下一条语句。

例如：

```
if (a==b) a++;
a--;
```

当 a 等于 b 时，a 就加 1，否则，a 就减 1。

3. 嵌套的 if...else 语句

嵌套的 if..else... 语法格式：

```
if(条件表达式 1)
{
语句 1
}
else if(条件表达式 2)
{
语句 2
}
else if(条件表达式 3)
{
语句 3
}

.....

else
{
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

语句 n}

以上形式的嵌套 if 语句执行过程可以这样理解：从上向下逐一对 if 后的条件表达式进行检测，当检测某一表达式的值为真时，就执行与此有关的语句。如果所有表达式的值均为零，则执行最后的 else 语句。

例如：

```
if(a>=0) {c=0;}
else if(a>=1) {c=1;}
else if(a>=2) {c=2;}
else if(a>=3) {c=3;}
else   {c=4;}
```

4. 实验演练

实验：用下载型实验板做以下实验：开关 K1 按下时，前两只 LED 灯亮，开关 K2 按下时，前三只 LED 灯亮，开关 K3 按下时，前四只 LED 灯亮，开关 K4 按下时，前五只 LED 灯亮，否则，所有灯熄灭。

在下载型实验板中，开关 K1 接 P3.2，开关 K2 接 P3.3，开关 K3 接 P3.4，开关 K4 接 P3.5。开关按下时，P3 口的相应脚为低电平，开关断开时，P3 口的相应脚为高电平。根据实验要求，编写的源程序如下：

```
#include<reg51.h>
void main()
{
unsigned char   SW;              //定义变量 SW
while(1)
    {
    SW=P3;                       //读入 P3 的状态
    SW= SW&0x3c;                 //取出 P3.2、P3.3、P3.4、P3.5 的状态(0x3c 即
                                   0011 1100)
    if(SW==0x38)                 //如果 P3.2 按下(0x38 即 0011 1000)
    P1=0xfc;                     //P1 口的低 2 位输出低电平，点亮低 2 位 LED
    else if(SW==0x34)            //如果 P3.3 按下(0x34 即 0011 0100)
    P1=0xf8;                     //P1 口的低 3 位输出低电平，点亮低 3 位 LED
    else if(SW==0x2c)            //如果 P3.4 按下(0x2c 即 0010 1100)
    P1=0xf0;                     //P1 口的低 4 位输出低电平，点亮低 4 位 LED
    else if(SW==0x1c)            //如果 P3.5 按下(0x1c 即 0001 1100)
    P1=0xe0;                     //P1 口的低 5 位输出低电平，点亮低 5 位 LED
    else
    P1=0xff;                     //否则，P1 口输出高电平，8 只灯熄灭
    }
}
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 if. c。对程序进行编译、链接和调试，产生 if. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target'，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 键进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_6/if 文件夹中。

注意事项 实验时，若实验结果与要求不符，请用万用表测量实验板 P3. 4(CPU 的 14 脚)的电压。正常情况下，该脚电压为+5V，若该电压跳动，请将实验板上的 X5045 拆下即可。

二、switch 语句及实验演练

1. switch 语句的形式

前面学习了 if 语句，用多个 if 语句可以实现多方向条件分支，但是可以发现，使用过多的 if 语句实现多方向分支会使条件语句嵌套过多，程序冗长，很不好读。这时如果使用开关语句，不但可以达到处理多分支选择的目的，而且还可以使程序结构清晰。开关语句的语法如下：

```
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 3：语句 3；break；
case 常量表达式 n：语句 n；break；
default：语句
}
```

运行时，switch 后面的表达式的值将会作为条件，与 case 后面的各个常量表达式的值相对比，如果相等时，则执行后面的语句，再执行 break(间断语句)语句，跳出 switch 语句；如果 case 没有和条件相等的值时，就执行 default 后的语句。当要求没有符合的条件时不做任何处理，则可以不写 default 语句。

注意事项 如果在 case 语句中遗忘了 break，则程序在执行了本行 case 选择之后，不会按规定退出 switch 语句，而是将执行后续的 case 语句。有经验的程序员可以在 switch 语句中预设一系列不含 break 的 case 语句，这样程序会把这些 case 语句加在一起执行。这对某些应用可能是很有效的，但对另一些情况则将引起麻烦，因此使用时必须小心谨慎。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

2. 实验演练

实验:用下载型实验板做以下实验:开关 K1 按下时,前两只 LED 灯亮,开关 K2 按下时,前三只 LED 灯亮,开关 K3 按下时,前四只 LED 灯亮,开关 K4 按下时,前五只 LED 灯亮,否则,所有灯熄灭。

前面用 if 语句编写了该实验的源程序,下面用 switch 语句编写,编写的源程序如下:

```
#include<reg51.h>
void main()
{
unsigned char   SW;                          //定义变量 SW
while(1)
  {
    SW=P3;                                   //读入 P3 的状态
    SW = SW&0x3c;                            //取出 P3.2、P3.3、P3.4、P3.5 的状态
                                               (0x3c 即 0011 1100)
    switch(SW)
      {
      case   0x38: P1=0xfc; break;           //如果 P3.2 按下,点亮低 2 位 LED
      case   0x34: P1=0xf8; break;           //如果 P3.3 按下,点亮低 3 位 LED
      case   0x2c: P1=0xf0; break;           //如果 P3.4 按下,点亮低 4 位 LED
      case   0x1c: P1=0xe0; break;           //如果 P3.5 按下,点亮低 5 位 LED
      default: P1=0xff;                      //以上状态都不是,8 只灯熄灭
      }
  }
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 switch.c。对程序进行编译、链接和调试,产生 switch.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 键进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_6/switch 文件夹中。

第三节　循环语句

在许多实际问题中,需要进行具有规律性的重复操作,如求累加和、数据块的搬移等。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

而计算机的基本特性之一就是具有重复执行一组语句的能力，即循环能力。利用这种循环能力，程序员只要编写一个包含重复执行语句的简短程序，就能执行所需的成千上万次的重复操作。几乎所有的实用程序都包含有循环结构。

作为构成循环结构的循环语句，一般是由循环体及循环终止条件两部分组成的。一组被重复执行的语句称为循环体，能否继续重复执行下去则取决于循环终止条件。在C语言中用来实现循环的语句有以下三种：while语句、do while语句和for语句。

一、while语句及实验演练

1. while语句的执行过程

while语句的语法形式为：

```
while(条件表达式)
{
循环体语句;
}
```

例如：

```
while(P3&0x10>0)
{
P1=0xf0;
}
```

while语句的语法形式中，while是C语言的关键字。while后一对圆括号中的表达式可以是C语言中任意合法的表达式，由它来控制循环体是否执行。在语法上，要求循环体可以是一条简单可执行语句，若循环体内需要多个语句，应该用大括号括起来，组成复合语句。

while语句的执行过程如下：

①计算while后一对圆括号中条件表达式的值。当值为非零时，执行步骤②；当值为零时，执行步骤④。

②执行循环体中语句。

③转去执行步骤①。

④退出while循环。

由以上叙述可知，while后一对圆括号中表达式的值决定了循环体是否执行，因此，进入while循环后，一定要有能使此表达式的值变为0的操作，否则，循环将会无限制地进行下去。图6-2为while语句的执行过程示意图。

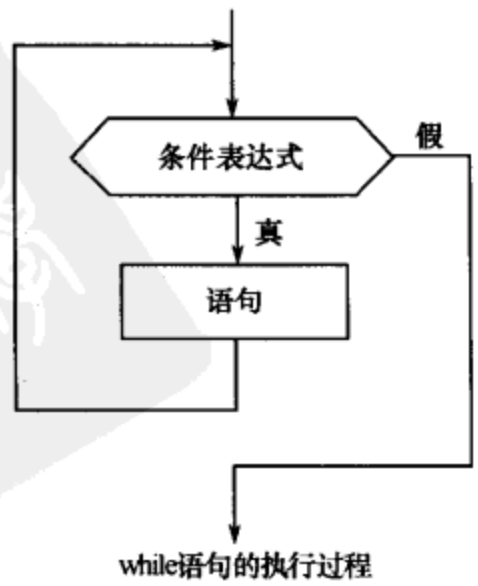


图6-2　while语句的执行过程

注意事项　不要把由if语句构成的选择结构与由while语句构成的循环结构混同起来。若if后条件表达式的值为非零时，其后的if子句只执行一次；而while后条件表达式的值为非零时，其后的循环体中的

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

语句将重复执行，而且在设计循环时，通常应在循环体内改变条件表达式中有关变量的值，使条件表达式的值最终变成0，以便能及时退出循环。

2. 实验演练

实验1：利用AT89C51实验板做以下实验，求1+2+3+…+100的累加和。

这是一个100个数的累加求和问题，所加的加数从1到100，可以看到加数是有规律变化的，后一个加数比前一个加数增加1，第一个加数为1，最后一个加数为100；因此可以在循环中使用一个整型变量i，每循环一次使i增加1，一直循环到i的值超过100，用这个办法就解决了所需的加数问题。但是要特别注意的是变量i需要有一个正确的初值，在这里它的初值应当设定为0。

下一个要解决的是求累加和，设用一个变量sum来存放这100个数的和值，可以先求0+1的和并将其放在sum中，然后把sum的数加上2再存放在sum中，依次类推，这和人们心算的过程没有什么区别，sum代表着人们脑中累加的那个和数，不同的是心算的过程由人们自己控制。在这里，sum累加的过程要放在循环中，由计算机来判断所加的数是否已经超过100，加数则放在变量i中，并在循环过程中一次次增加1。

根据以上分析，编写的源程序如下：

```
#include <AT89x51.h>
#include <stdio.h>
void main(void)
{
  int i,sum;                    //定义变量
  SCON= 0x40;                   //串口方式1
  REN=1;                        //允许接收
  TMOD= 0x20;                   //定时器1定时方式2
  TH1= 0xE6;                    //12MHz 1200波特率
  TL1= 0xE6;
  TI= 1;
  TR1= 1;                       //启动定时器
  i=1;
  sum=0;                        //变量赋初值
  while(i<=100)                 //i小于或等于100时执行循环体
  {sum=sum+i;i++;}              //在循环体中累加一次，i增1
  printf("sum=%d\n",sum);
  while(1);
}
```

实验步骤如下：

①打开Keil软件，输入上面的程序，保存为sum1.c。对程序进行编译、链接，产生sum1.hex目标文件。

②取下AT89C51实验板上的CPU，插入RF-810编辑器，对CPU进行编程。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

③将 AT89C51 实验板的串口和电脑的 COM1 连接，将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件，将波特率调整为 1200，串口选择为 COM1，并打开串口。

⑤打开 AT89C51 实验板电源，程序运行后的输出结果：sum＝5050。串口调试软件运行情况如图 6-3 所示。

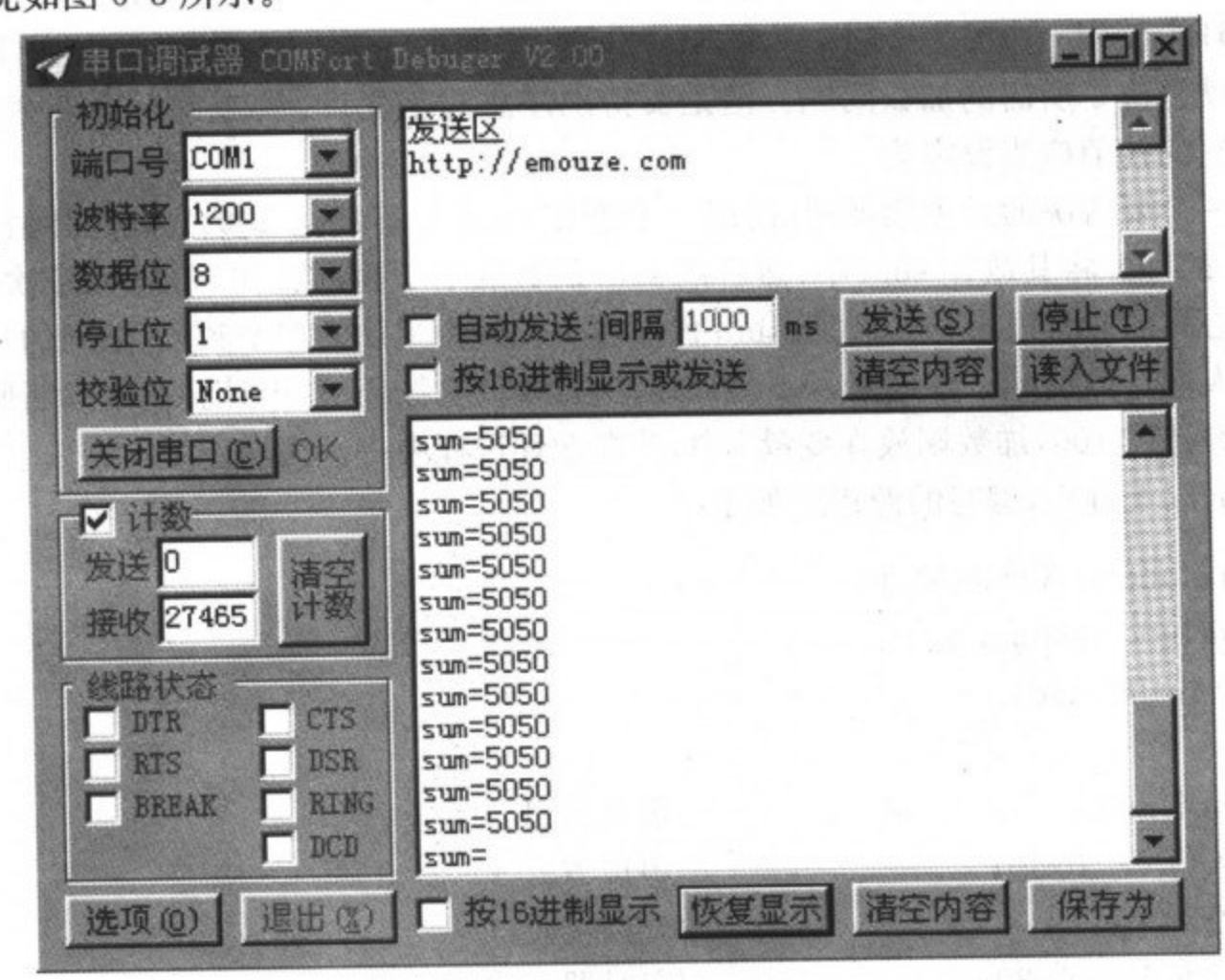


图 6-3　求累加和程序运行结果

该实验程序在本书附赠光盘的 ch_6/sum1 文件夹中。

重点提示　在循环体中一定要有使循环趋向结束的操作，以上循环体内的语句 i++使 i 不断增 1，当 i>100 时，循环结束。如果没有 i++这一语句，则 i 的值始终不变，循环将无限进行。在循环体中，语句的先后位置必须符合逻辑，否则将会影响运算结果。例如，若将上例中的 while 循环体改写成：

```
while(i<=100)
{i++;                          //先计算 i 值，后计算 sum 的值
sum=sum+i;
}
```

运行后，将输出：

sum=5150

运行的过程中，少加了第一项的值 1，而多加了最后一项的值 101。

实验 2：用下载型实验板做以下实验：计算 P3.2 上的开关 K1 按下的次数，并将结果输出到 P1 的 LED 上(灯灭表示 1，灯亮表示 0)。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

按钮开关 K1 的信号是，未按下时，输出为高电平，按下时为低电平。所以，按下 K1 后再放开 K1，就会产生一个负向脉冲输出到 P3.2 脚。欲判断按钮开关 K1 是否被按下后又被放开，必须先检测到 P3.2 为 0，然后再检测到 P3.2 为 1，即表示 K1 按下又放开。

根据以上分析，编写的源程序如下：

```
#include<reg51.h>
sbit P32=P3^2;
void Delay(unsigned int i)          //延时程序
{
    unsigned int j;
    for(;i>0;i--)
    for(j=0;j<125;j++)
    {;}
}
void main(void)
{
unsigned char  cout;                //定义变量 cout
while(1)
  {
  P1=cout;                          //将计数值输出到 P1
  while(P32);                       //等待键按下(等待 P3.2 为 0)
  while(! P32);                     //等待键放开(等待 P3.2 为 1)
  delay(10);                        //延时 10ms
  cout++;                           //计数值加 1
  }
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 cout.c。对程序进行编译、链接和调试，产生 cout.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target'Target'，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_6/cout 文件夹中。

重点提示 在上述程序中，加入了延时函数 Delay，目的是消除按键时的抖动现象。若取消程序中的延时函数，在按键时计数会出错。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

二、do while 语句及实验演练

1. do while 语句的执行过程

do while 语句语法形式为：

```
do
{
循环体语句；
}
while(条件表达式)；
```

例如：

```
do
{i++;S+=i;}
while(i<10)；
```

do while 语句的语法格式中，do 是 C 语言的关键字，必须和 while 联合使用。do while 循环由 do 开始，用 while 结束；必须注意的是：在 while(表达式)后的“;”不可丢，它表示 do while 语句的结束。while 后一对圆括号中的表达式，可以是 C 语言中任意合法的表达式，由它控制循环是否执行。在 do 和 while 之间的循环体内有多个语句时，应该用大括号括起来，组成复合语句。

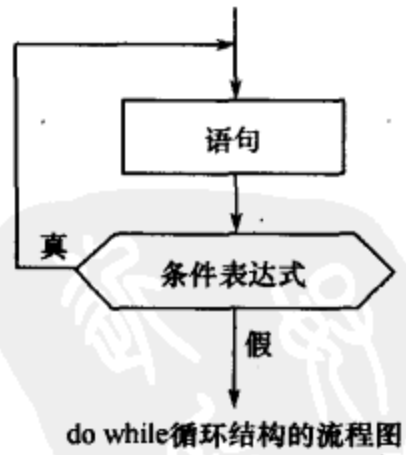
do while 语句的执行过程如下：

①执行 do 后面循环体中的语句。

②计算 while 后一对圆括号中表达式的值。当值为非零时，转去执行步骤①；当值为零时，执行步骤③。

③退出 do while 循环。

图 6-4 为 do while 语句的执行过程示意图。



do while循环结构的流程图

图 6-4　do while 语句的执行过程

由 do while 构成的循环与 while 循环十分相似，它们之间的重要区别是：while 循环的控制出现在循环体之前，只有当 while 后面表达式的值为非零时，才可能执行循环体。在 do while 构成的循环中，总是先执行一次循环体，然后再求表达式的值，因此，无论表达式的值是零还是非零，循环体至少要被执行一次。

和 while 循环一样，在 do while 循环体中，要有能使 while 后表达式的值变为 0 的操作，否则，循环将会无限制地进行下去。

以笔者的经验，do while 循环用得并不多，大多数的循环用 while 来实现会更直观。

请比较以下两段程序，前者使用 while 循环，后者使用 do while 循环。

程序 1：

```
int a = 0;
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
while(a>0)
{
a--;
}
```

变量 a 初始值为 0，条件 a>0 显然不成立。所以循环体内的 a--;语句未被执行。本段代码执行后，变量 a 值仍为 0。

程序 2：

```
int a = 0;
do
{
a--;
}
while(a>0);
```

尽管循环执行前，条件 a> 0 一样不成立，但由于程序在运行到 do 时，并不先判断条件，而是直接先运行一遍循环体内的语句：a--，于是 a 的值成为 -1，然后，程序才判断 a >0，发现条件不成立，循环结束。

2. 实验演练

实验：利用 AT89C51 实验板做以下实验，求 1+2+3+…+100 的累加和。

用 do while 循环语句编写的源程序如下：

```
#include <AT89x51.h>
#include <stdio.h>
void main(void)
{
  int i,sum;                  //定义变量
  SCON= 0x40;                 //串口方式 1
  REN=1;                      //允许接收
  TMOD= 0x20;                 //定时器 1 定时方式 2
  TH1= 0xE6;                  //12MHz 1200 波特率
  TL1= 0xE6;
  TI= 1;
  TR1= 1;                     //启动定时器
  i=1;
  sum=0;                      //变量赋初值
  do
  {sum=sum+i;i++;}            //在循环体中累加一次，i 增 1
  while(i<=100);              //i 小于或等于 100 时执行循环体
  printf("sum=%d\n",sum);
  while(1);
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 sum2. c。对程序进行编译、链接，产生 sum2. hex 目标文件。

②取下 AT89C51 实验板上的 CPU，插入 RF-810 编辑器，对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接，将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件，将波特率调整为 1200，串口选择为 COM1，并打开串口。

⑤打开 AT89C51 实验板电源，程序运行后的输出结果：sum＝5050。

该实验程序在本书附赠光盘的 ch_6/sum2 文件夹中。

三、for 语句及实验演练

1. for 语句的执行过程

for 语句构成的循环结构通常称为 for 循环。for 循环的一般形式为：

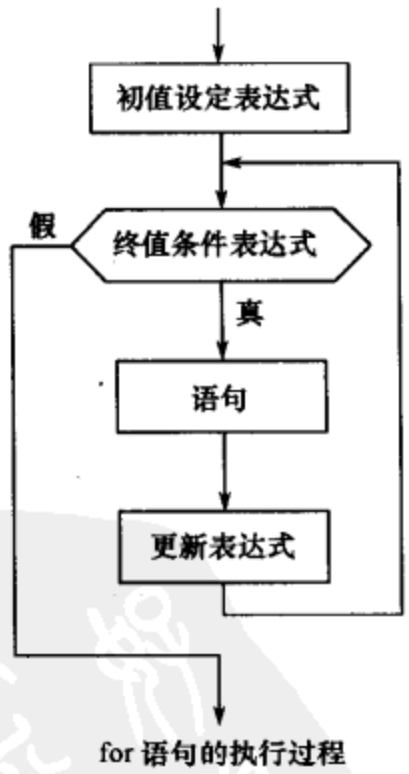
```
for(表达式 1;表达式 2;表达式 3)
{
循环体语句;
}
```

例如：

```
for(k=0;k<10;k++)
printf("* ");
```

以上 for 循环在一行上输出 10 个“*”号。

for 是 C 语言的关键字，其后的一对圆括号中通常含有三个表达式，各表达式之间用“;”隔开。这三个表达式可以是任意形式的表达式，通常主要用于 for 循环的控制。紧跟在 for(…)之后的循环体，在语法上要求是一条语句：若在循环体内需要多条语句，应该用大括号括起来组成复合语句。



for 语句的执行过程

图 6-5　for 循环的执行过程

for 循环的执行过程如下：

①计算“表达式 1”(“表达式 1”通常称为“初值设定表达式”)。

②计算“表达式 2”(“表达式 2”通常称为“终值条件表达式”)。若其值为非零，转步骤③；若其值为零，转步骤⑤。

③执行一次 for 循环体。

④计算“表达式 3”(“表达式 3”通常称为“更新表达式”)，转向步骤②。

⑤结束循环，执行 for 循环之后的语句。

图 6-5 为 for 循环的执行过程示意图。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

2. for 语句特例说明

下面对 for 循环语句的几种特例进行简要说明。

①for 语句中的小括号内的三个表达式全部为空。

例如：

```
for(;;)
{
循环体语句;
}
```

在小括号内只有两个分号，无表达式。这意味着没有设初值，无判断条件，循环变量为增值，它的作用相当于 while(1)，即构成一个无限循环过程。一般在编制 8051 单片机监控程序需要无限循环时，采用这种形式的 for 循环语句。

②for 语句三个表达式中，表达式 1 缺省。

例如：

```
for(;i<=100;i++)sum=sum+i;
```

即不对 i 设初值。

③for 语句三个表达式中，表达式 2 缺省。

例如：

```
for(i=1;;i++)sum=sum+i;
```

即不判断循环条件，认为表达式始终为真。循环将无休止地进行下去。它相当于：

```
i=1;
while(1)
{
  sum=sum+i;
  i++;
}
```

④for 语句三个表达式中，表达式 1、表达式 3 省略。

例如：

```
for(;i<=100;)
{
sum=sum+i;
i++;
}
```

它等效于：

```
while(i<100)
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
{
sum=sum+i;
i++;
}
```

⑤没有循环体的 for 语句。

例如：

```
int sum=2000;
for(t=0;t<sum;t++)
{;}
```

此例在程序中起延时作用。

3. 循环语句的嵌套

在一个 for 循环体内又包含了另一个循环，称为循环嵌套。循环嵌套主要用于产生时间延时。这种延时称为软件延时，软件延时会使控制器在延时循环时接收不到其他的输入，解决这个问题的方法是使用中断。

下面是一个实用的 lms 延时程序。如给这个函数的形参 i 程序传递一个 500 的数值，则可以产生约 0.5s 的延时。

例如：

```
void Delay(unsigned int i)    //延时程序函数
{
    unsigned int j;
    for(;i>0;i--)          //这里的 i 由实参传入一个值，因此，不能在这个式子
                             中赋初值
    for(j=0;j<125;j++)
    {;}
}
```

这个程序编写得近似正确，但并不精确。不同的编译器会产生不同的延时，因此 j 的上限值 125 应根据实验进行补偿调整。

4. 实验演练

实验：利用 AT89C51 实验板做以下实验，求 1+2+3+…+100 的累加和。

用 for 循环语句编写的源程序如下：

```
#include <AT89x51.h>
#include <stdio.h>
void main(void)
{
  int i,sum;                  //定义变量
  SCON= 0x40;                 //串口方式 1
  REN=1;                      //允许接收
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    TMOD= 0x20;              //定时器 1 定时方式 2
    TH1= 0xE6;               //12MHz 1200 波特率
    TL1= 0xE6;
    TI= 1;
    TR1= 1;                  //启动定时器
    sum=0;
    for(i=1;i<=100;i++)
    sum=sum+i;
    printf("sum=%d\n",sum);
    while(1);
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 sum3. c。对程序进行编译、链接，产生 sum3. hex 目标文件。

②取下 AT89C51 实验板上的 CPU，插入 RF-810 编辑器，对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接，将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件，将波特率调整为 1200，串口选择为 COM1，并打开串口。

⑤打开 AT89C51 实验板电源，程序运行后的输出结果：sum=5050。

该实验程序在本书附赠光盘的 ch_6/sum3 文件夹中。

四、break 和 continue 语句在循环体中的作用

1. break 语句

前面已经介绍过用 break 语句可以跳出 switch 语句体。在循环结构中，也可应用 break 语句跳出本层循环体，从而提前结束本层循环。

例如，执行如下程序段：

```
void main(void)
{
  int i,sum;
  sum=0;
  for(i=1;i<=10;i++)
  {
    sum=sum+i;
    if(sum>5)break;
    printf("sum=%d\n",sum);
  }
}
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

程序的输出结果如下：

```
sum=1
sum=3
```

上例中，如果没有 break 语句，程序将进行 10 次循环；但当 i=3 时，sum 的值为 6，if 语句中的表达式：s>5 的值为 1，于是执行 break 语句，跳出 for 循环，从而提前终止循环。

方法技巧 当 break 出现在循环体中的 switch 语句体内时，其作用只是跳出该 switch 语句体。当 break 出现在循环体中，但并不在 switch 语句体内时，则在执行 break 后，跳出本层循环体。因此，在一个循环程序中，既可以通过循环语句中的表达式来控制循环程序是否结束，除此之外，还可以通过 break 语句强行退出循环结构。

2. continue 语句

continue 意为继续，它的作用及用法和 break 类似，重要区别在于：当前循环遇到 break，是直接结束循环，而若遇上 continue，则是停止当前这一遍循环，然后直接尝试下一遍循环。可见，continue 并不结束整个循环，而仅仅是中断这一遍循环，然后跳到循环条件处，继续下一遍的循环。当然，如果跳到循环条件处，发现条件已不成立，那么循环也将结束，所以称为：尝试下一遍循环。

在 while 和 do while 循环中，continue 语句使得流程直接跳到循环控制条件的测试部分，然后决定循环是否继续进行。在 for 循环中，遇到 continue 后，跳过循环体中余下的语句，而去对 for 语句中的“表达式 3”求值，然后进行“表达式 2”的条件测试，最后根据“表达式 2”的值来决定 for 循环是否执行。

例如：输出整数 1～100 的累加值，但要求跳过所有个位为 3 的数。

在循环中加一个判断，如果该数个位是 3，就跳过该数不加。如何判断一个 1～100 的数中哪些数的个位是 3 呢？用求余运算符“%”，将一个 2 位以内的正整数，除以 10 以后，余数是 3，就说明这个数的个位为 3。比如 23 ，除以 10，商 2，余数 3。这里不需要商，所以用求余（也称为求模）运算：23 % 10 = 3。

根据以上分析，编写的源程序如下：

```
void main(void)
{
int i,sum = 0;
for(i = 1; i<=100;i++)
   {
   if( i % 10 == 3)
      continue;
      sum =sum+ i;
   }
   printf("sum=%d\n",sum);
}
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

3. 实验演练

实验:用下载型实验板做以下实验:开机后,全部 LED 不亮,按下 K1 ,则 8 个 LED 中只有 LED1 熄灭,并开始循环流动。如果中间 K2 键被按下,所有 LED 再从头开始循环流动显示。

实验源程序如下:

```
#include<reg51.h>
void Delay(unsigned int i)                    //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)                             //形参 i 由实际参数传入一个值
    for(j=0;j<125;j++)
    {;}
}
void main(void)
{
    unsigned char i,out;
    P3=P3|0x3c;                               //取出 P3.2、P3.3、P3.4、P3.5 的状态
    if((P3|0xfb)! =0xff)                      //如果 P3.2(K1)按下(0xfb 即 1111
                                                1011)
      while(1)
      {
        out=0x80;
        for(i=0;i<8;i++)
          {
            P1=out;
            Delay(500);                       //延时
            out=out>>1;
            if((P3|0xf7)! =0xff)              //如果 P3.3(K2)按下(0xf7 即 1111
                                                0111)
            break;
            }
          }
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 break.c。对程序进行编译、链接和调试,产生 break.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target'，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_6/break 文件夹中。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第七章　Cx51 函数

在前面各章的例子中，编写的 C 语言程序都用到了以“main”开头的主函数，并且在程序中频繁地调用了 C 语言提供的用于输入和输出的库函数——scanf、printf 函数。读者可能注意到：main 是由用户自己编写的，而 scanf、printf 则是由 C 语言提供的，用户只要学会如何正确调用就行。一个实用的 C 语言源程序总是由许多函数组成，这些函数都是根据实际任务，由用户来调用。在这些函数中，可以调用 C 语言提供的库函数，也可以调用由用户自己或他人编写的函数。但是，一个 C 语言源程序无论包含了多少函数，C 程序总是从 main 函数开始执行。本章主要讨论函数的分类，如何自定义函数并调用这些函数等内容。

第一节　函数的分类和定义

一、函数的分类

从 C 语言程序的结构上划分，C 语言函数分为主函数 main()和普通函数两种。而对普通函数，从不同的角度或以不同的形式又可以进行如下分类。

从用户使用的角度划分，函数有两种：一种是标准库函数；一种是用户自定义函数。

1. 标准库函数

标准库函数是由 Keil Cx51 编译器提供的，在 C 编译系统设计过程中，系统的设计者事先将一些独立的功能模块编写成公用函数，并将它们集中存放在系统的函数库中，供系统的使用者在设计应用程序时使用。故把这种函数称为库函数或标准库函数。Cx51 具有功能强大、资源丰富的标准函数库。因此，作为系统的使用者，在进行程序设计时，应该善于充分利用这些功能强大、内容丰富的标准库函数资源，以提高效率、节省时间。

在调用库函数时，用户在源程序 include 命令中应该包含头文件名。例如，调用输出函数 printf 时，要求程序在调用输出库函数前包含以下的 include 命令：

```
#include <stdio.h>
```

include 命令必须以 # 号开头，系统提供的头文件以“.h”作为文件的后缀，文件名用一对尖括号括起来。注意：include 命令不是 C 语句，因此不能在最后加分号。

2. 用户自定义函数

用户自定义函数，顾名思义，是用户根据自己的需要编写的函数。

从函数定义的形式上划分可以有三种形式：无参数函数、有参数函数和空函数。

(1)无参数函数

此种函数在被调用时，既无参数输入，也不返回结果给调用函数。它是为完成某种操

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

作而编写的。

(2)有参数函数

在调用此种函数时,必须提供实际的输入参数。此种函数在被调用时,必须说明与实际参数一一对应的形式参数,并在函数结束时返回结果,供调用它的函数使用。

(3)空函数

此种函数体内无语句,是空白的。调用此种空函数时,什么工作也不做,不起任何作用。而定义这种函数的目的并不是为了执行某种操作,而是为了以后程序功能的扩充。在程序的设计过程中,往往根据需要确定若干模块,分别由一些函数来实现。而在程序设计的第一阶段,往往只设计最基本的功能模块的函数,其他模块的功能函数,则可以在以后补上。为此,先将这些非基本模块的功能函数定义成空函数,先占好位置,以后再用一个编好的函数代替它。这样做,程序的结构清楚、可读性好,以后扩充新功能也方便。

应该指出的是,在C语言中,所有的函数定义,包括主函数 main 在内,都是平行的。也就是说,在一个函数的函数体内,不能再定义另一个函数,即不能嵌套定义。但是函数之间允许相互调用,也允许嵌套调用。习惯上把调用者称为主调函数。函数还可以自己调用自己,称为递归调用。main 函数是主函数,它可以调用其他函数,而不允许被其他函数调用。因此,C程序的执行总是从 main 函数开始,完成对其他函数的调用后再返回到 main 函数,最后由 main 函数结束整个程序。一个C源程序必须有,也只能有一个主函数 main。

二、函数的定义

C语言虽然提供了丰富的库函数,但这些函数是面向所有用户的,不可能满足每个用户的各种特殊需要,因此大量的函数必须由用户自己来编写。下面讨论3种函数的具体定义方法。

1. 无参数函数的定义方法

无参数函数的定义形式为:

```
返回值类型说明符 函数名()
{
说明部分
函数体语句
}
```

其中,返回值类型说明符和函数名称为函数头。返回值类型标识符指明了本函数的类型,即函数返回值的类型;若不写类型说明符,默认为 int 类型。函数名是由用户定义的标识符,函数名后有一个空括号,其中无参数,但括号不可少。{}中的内容称为函数体。在函数体中也有类型说明,这是对函数体内部所用到的变量的类型说明。在很多情况下都不要求无参数函数有返回值,此时函数类型符可以写为 void。

例如:

```
void Hello(void)
{
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
printf ("Hello,world\n");
}
```

这里，Hello 为函数名，Hello 函数是一个无参数函数，当被其他函数调用时，输出 Hello world 字符串。

2. 有参数函数的定义方法

有参数函数的定义方式为：

```
返回值类型说明符 函数名(形式参数表)
形式参数类型说明
{
说明部分
函数体语句
}
```

有参数函数比无参数函数多了两个内容，一是形式参数表，二是形式参数类型说明。在形式参数表中给出的参数称为形式参数(形参)，它们可以是各种类型的变量，各参数之间用逗号间隔。在进行函数调用时，主调函数将赋予这些形式参数实际的值。形式参数既然是变量，当然必须给以类型说明。在函数体中，除形式参数外，用到的其他变量必须在说明部分进行定义，这些变量(包括形式参数)只在函数被调用时才临时开辟存储单元，当退出函数时，这些临时开辟的存储单元全部被释放掉。也就是说，这些变量只在函数体内起作用，与其他函数体中的变量互不相关，它们可以和其他函数体中的变量同名。

例如，定义一个函数，用于求两个数中的大数，可写为：

```
int max(a,b)
int a,b;
{
if (a>b) return a;
else return b;
}
```

第一行说明 max 函数是一个整型函数，其返回的函数值是一个整数。形参为 a、b。第二行说明 a、b 均为整型量。a、b 的具体值是由主调函数在调用时传送过来的。在{}中的函数体内，除形参外没有使用其他变量，因此只有语句而没有变量类型说明。上边这种定义方法称为“传统格式”。这种格式不易于编译系统检查，从而会引起一些非常细微而且难于跟踪的错误。ANSI C 的新标准中把对形参的类型说明合并到形参表中，称为“现代格式”。

例如 max 函数用现代格式可定义为：

```
int max(int a,int b)
{
if(a>b) return a;
else return b;
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

}

现代格式在函数定义和函数说明时，给出了形式参数及其类型，在编译时易于对它们进行查错，从而保证了函数说明和定义的一致性。

在 max 函数体中的 return 语句是把 a(或 b)的值作为函数的值返回给主调函数。

在 C 程序中，一个函数的定义可以放在任意位置，既可放在主函数 main 之前，也可放在 main 之后。

3. 空函数的定义方法

空函数的定义形式为：

返回值类型说明符 函数名()
{}

例如：

```
int min()
{  }
```

三、实验演练

实验 1：用下载型实验板做如下实验：使 P1 所接 LED 以流水灯的形式显示。

实验源程序如下：

```
#include <reg51.h>
#include <intrins.h>                    //该文件包含有_crol_()函数的说明
void Delay(unsigned int i)              //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)                       //变量 i 由实际参数传入一个值,因此 i 不能
                                          赋初值
    for(j=0;j<125;j++)
    {;}
}
void main()
{
    unsigned char out=0xfe;
    while(1)
    {
        P1=out;
        out=_crol_(out,1);              //由库函数_crol_()进行循环左移
        Delay(1000);                    //延时
    }
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
}
```

以上程序中，Delay()是自定义函数，无返回值，起延时作用。_crol_()是循环左移库函数，在使用库文件前，应在源程序 include 命令中包含头文件名 intrins. h。

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 **liuLED**. c。对程序进行编译、链接和调试，产生 **liuLED**. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target'，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_7/liuLED 文件夹中。

实验 2：用 AT89C51 实验板做实验：输入两个整型数，输出最大数。

实验源程序如下：

```
#include <AT89x51.h>
#include <stdio.h>
int max(int a,int b)
  {
    if(a>b)
    return a;
    else
    return b;
  }
void main(void)
  {
    int x,y,z;
    SCON= 0x40;                          //串口方式 1
    REN=1;                               //允许接收
    TMOD= 0x20;                          //定时器 1 定时方式 2
    TH1= 0xE6;                           //12MHz 1200 波特率
    TL1= 0xE6;
    TI= 1;
    TR1= 1;                              //启动定时器
    printf("input two numbers:\n");
    scanf("%d%d",&x,&y);
    z=max(x,y);
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    printf("max number=%d",z);
    while(1);
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 max. c。对程序进行编译、链接，产生 max. hex 目标文件。

②取下 AT89C51 实验板上的 CPU，插入 RF-810 编辑器，对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接，将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件，将波特率调整为 1200，串口选择为 COM1，并打开串口。

⑤打开 AT89C51 实验板电源，串口调试软件要求输入数据。在串口调试软件的发送区输入第一个数据(设为 10)，按一下键盘上的回车键，以使 scanf 函数确认有数据输入，再按一下调试软件中的“发送”按钮，此时在调试软件的接收区会出现数据 10，在串口调试软件的发送区的第一个数据删除，再输入第二个数据(设为 20)，按一下键盘上的回车键，以使 scanf 函数确认有数据输入，再按一下调试软件中的“发送”按钮，此时，在调试软件的接收区会出现数据 20，并给出判断结果，如图 7-1 所示。

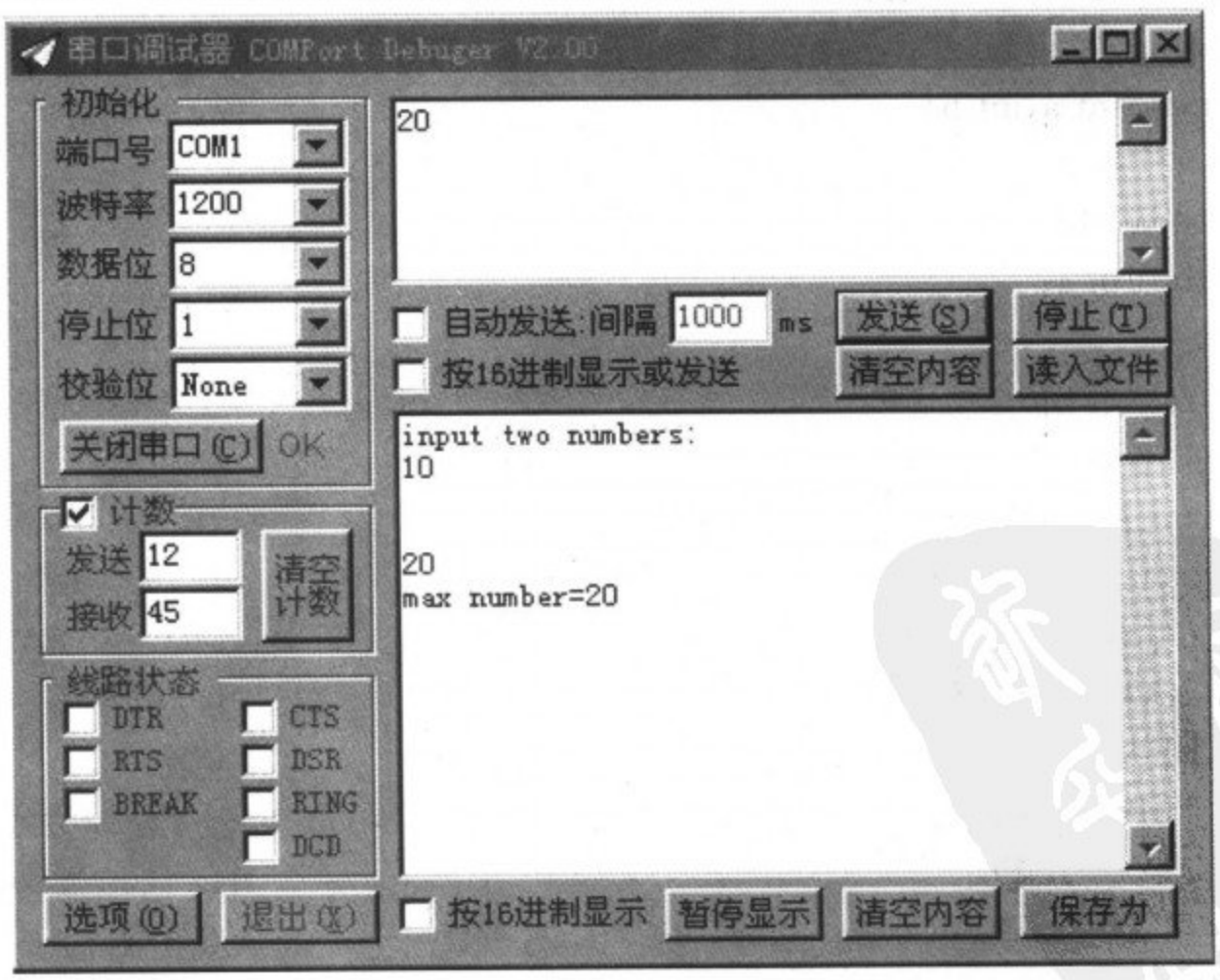


图 7-1　输出判断结果

该实验程序在本书附赠光盘的 ch_7/max 文件夹中。

在这个例子中，主函数 main()先调用输入函数 scanf()，从键盘输入两个值分别赋给局部变量 a、b，然后调用自定义函数 max()，求出 a、b 中的大数，再调用输出函数 printf()，输出 a、b 中的大数。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第二节　函数的参数和返回值

一、函数的参数

在进行函数调用时，主调用函数与被调用函数之间具有数据传递关系。这种数据传递是通过函数的参数实现的。在定义一个函数时，位于函数名后面圆括号中的变量名为“形式参数”(简称形参)，而在调用函数时，函数名后面括号中的表达式为“实际参数”(简称实参)。形式参数在未发生函数调用之前，不占用内存单元，因而也是没有值的。只有在发生函数调用时它才被分配内存单元，同时获得从主调用函数中实际参数传递过来的值。函数调用结束后，它所占用的内存单元也被释放。

实际参数可以是常数，也可以是变量或表达式，但要求它们具有确定的值。进行函数调用时，主调用函数将实际参数的值传递给被调用函数中的形式参数。为了完成正确的参数传递，实际参数的类型必须与形式参数的类型一致，如果两者不一致，则会发生“类型不匹配”错误。

函数调用中发生的数据传送是单向的。即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。

下面举例进行说明：

```
int sum(int n)
{
  int i;
  for(i=n-1;i>=1;i--)
  n=n+i;
  printf("n=%d\n",n);
}
void main()
{
  int n;
  printf("input number\n");
  scanf("%d",&n);
  sum(n);
  printf("n=%d\n",n);
}
```

本程序中定义了一个函数 sum，该函数的功能是求$\sum n$ 的值。在主函数中输入 n 值，并作为实参，在调用时传送给 sum 函数的形参量 n(注意，本例的形参变量和实参变量的标识符都为 n，但这是两个不同的量，各自的作用域不同)。在主函数中用 printf 语句输出一次 n 值，这个 n 值是实参 n 的值。在函数 sum 中也用 printf 语句输出了一次 n 值，这个 n 值是形参最后取得的 n 值。从运行情况看，输入 n 值为 100，即实参 n 的值为 100。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

把此值传给函数 s 时，形参 n 的初值也为 100，在执行函数过程中，形参 n 的值变为 5050。返回主函数之后，输出实参 n 的值仍为 100。可见实参的值不随形参的变化而变化。

二、函数的返回值

函数的返回值是指函数被调用之后，执行函数体中的程序段所取得的并返回给主调用函数的值。如调用前面介绍的 max 函数取得的最大数等。对函数返回值说明如下：

①函数的返回值只能通过 return 语句返回主调用函数。

return 语句的一般形式为：

return 表达式；

或者为：

return（表达式）；

该语句的功能是计算表达式的值，并返回给主调用函数。在函数中允许有多个 return 语句，但每次调用只能有一个 return 语句被执行，因此只能返回一个函数值。

②函数值的类型和函数定义中函数的类型应保持一致。如果两者不一致，则以函数类型为准，自动进行类型转换。

③如函数值为整型，在函数定义时可以省去类型说明。

④函数体内可以没有 return 语句，程序的流程就一直执行到函数末尾的“}”，然后返回到函数，这时也没有确定的函数值带回。在定义这类函数时，可以明确定义为“空类型”，类型说明符为“void”。例如，前面介绍的函数 sum 并不向主函数返函数值，因此可定义为：

```
void sum(int n)
{
……
}
```

一旦函数被定义为空类型后，就不能在主调用函数中使用被调用函数的函数值了。例如，在定义 sum 为空类型后，在主函数中写下述语句 s=sum(n)；就是错误的。为了使程序有良好的可读性并减少出错，凡不要求返回值的函数都应定义为空类型。

三、实验演练

实验：用下载型实验板做如下实验：按下开关 K1（接于 P3.2），P1 口外接的 LED 全亮，否则，P1 口外接的 LED 全灭。

实验源程序如下：

```
#include <reg51.h>
sbit P32=P3^2;
bit PB_K1(void)
{
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
if(P32==0)                    // P32==0 相当于开关 K1 被按下
{return 1;}                   //开关 K1 按下,返回 1
else
{return 0;}                   //不按开关,返回 0
}
void main(void)
{
while(1)
    {
     if(PB_K1()==1)           //开关 K1 被按下
     {P1=0;}                  //P1 口 LED 全亮
     if(PB_K1()==0)           //开关 K1 未被按下
     {P1=0xff;}               //P1 口 LED 全灭
     }
}
```

重点提示 函数 PB_K1 的功能是,当开关 K1 被按下时(P3.2 为 0),则返回 1;否则(P3.2 为 1),返回 0。因为函数 PB_K1 只返回两种状态,所以,可以定义其返回值的数据格式为 bit。

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 **sw.c**。对程序进行编译、链接和调试,产生 sw.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target'Target',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_7/sw 文件夹中。

第三节 函数的调用

一、函数调用的一般形式

函数调用的一般形式为:

函数名(实际参数表);

对于有参数型函数,若包含多个实际参数,则应将各参数之间用逗号分隔开。主调用函数的数目与被调用函数的形式参数的数目应该相等。实际参数与形式参数按实际顺序一一对应传递数据。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

如果调用的是无参数函数，则实际参数表可以省略，但函数名后面必须有一对空括号。

二、函数调用的方式

主调用函数对被调用函数的调用可以有以下 3 种方式。

1. 函数调用语句

即把被调用函数名作为主调用函数中的一个语句。

例如：

```
hello();
```

此时并不要求被调用函数返回结果数值，只要求函数完成某种操作。

2. 函数结果作为表达式的一个运算对象

例如：

```
result=2 * max(a,b);
```

此时被调用函数以一个运算对象的身份出现在一个表达式中。这就要求被调用函数带有 return 语句，以便返回一个明确的数值参加表达式的运算。被调用函数 max 为表达式的一部分。它的返回值乘 2 再赋给变量 result。

3. 函数参数

即被调用函数作为另一个函数的实际参数。

例如：

```
m=max(a,gcd(u,v));
```

其中 gcd(u,v)是一次函数调用。它的值作为另一个函数调用 max()的实际参数之一。

三、对被调用函数的说明

在一个函数中调用另一个函数必须具有以下条件：

①被调用函数必须是已经存在的函数(库函数或用户自定义函数)。

②如果程序中使用了库函数，或使用了不在同一文件中的另外的自定义函数，则应该在程序的开头处使用 #include 包含语句，将所用的函数信息包括到程序中来。

例如：

#include <stdio.h>将标准输入、输出头文件(在函数库中)包含到程序中来。

#include <math.h>将函数库中专用数学库的函数包含到程序中来。

这样，程序编译时，系统就会自动将函数库中的有关函数调入到程序中去，编译出完整的程序代码。

③如果被调用函数出现在主调用函数之后，一般应在主调用函数中，在对被调用函数调用之前，对被调用函数的返回值类型作出说明。一般的形式为：

返回值类型说明符　被调用函数的函数名();

例如：

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
main()
{
int max();                              //被调用函数说明
int a=60,b=55,m;
printf("a=%d,b=%d",a,b);
m=max(a,b);
printf("max=%d\n",m);
}
int max(int x,int y)
{
return(x>y? x:y);
}
```

④如果被调用函数的定义出现在主调用函数之前，可以不对被调用函数加以说明。因为C编译器在编译主调用函数之前，已经预先知道已定义了被调用函数的类型，并自动加以处理。

例如：

```
int max(int x,int y)
{
return(x>y? x:y);
}
main()
{
int a=60,b=55,m;
printf("a=%d,b=%d",a,b);
m=max(a,b);
printf("max=%d\n",m);
}
```

四、函数的嵌套

在C语言中，函数的定义都是相互独立的，即在定义函数时，一个函数的内部不能包含另一个函数。尽管C语言中函数不能嵌套定义，但允许嵌套调用函数。也就是说，在调用一个函数的过程中，允许调用另外一个函数。

五、函数的递归调用与再入函数

在调用一个函数的过程中，又直接或间接地调用该函数本身。这种情况称为函数的递归调用。

下面以计算一个数的阶乘为例来说明函数的递归调用。

一般说来，任何大于0的正整数 n 的阶乘等于 n 乘以 $(n-1)$ 的阶乘，即 $n!=n(n-$

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

1)!。用$(n-1)!$的值来表示$n!$的值的表达式就是一种递归调用，因为一个阶乘的值是以另一个阶乘的值为基础的。

再入函数是一种可以在函数体内不直接或间接调用其自身的一种函数，显然再入函数是可以进行递归调用的。Keil Cx51 编译器采用一个扩展关键字 reentrant 定义再入函数，再入函数的一般形式为：

函数类型　函数名(形式参数表)[reentrant]

再入函数可被递归调用，无论何时，包括中断服务函数在内的任何函数都可调用再入函数。与非再入函数的参数传递和局部变量的存储分配方法不同，Cx51 编译器为再入函数生成一个模拟栈，通过这个模拟栈来完成参数传递和存放局部变量。模拟栈所在的存储器空间根据再入函数存储器模式的不同，可以是 DATA、PDATA 或 XDATA 存储器空间。当程序中包含有多种存储器模式的再入函数时，Cx51 编译器为每种模式单独建立一个模拟栈并独立管理各自的栈指针。需要说明的是，再入函数不能传送 bit 类型的参数，也不能定义一个局部变量，再入函数不能包括位操作以及 8051 系列单片机的可位寻址区。

采用递归调用求正数 n 的阶乘的程序如下：

```
int fac(int n) reentrant
{
  int result;
  if(n==0)
  result=1;
  else
  result=n*fac(n-1);
  return(result);
}
main()
{
  int n;
  printf("please input a number:\n");
  scanf("%d",&n);
  printf("fac(%d)=%d\n",n,fac(n));
  while(1);
}
```

程序执行结果：

```
please input a number
4    回车
fac(4)=24
```

在这个程序中，定义了一个再入函数 fac(n)，它是用来计算阶乘 n! 的函数。在 fac()的函数体中又调用了 fac()函数本身，因此这是一种函数的递归调用。再入函数在进行

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

递归调用时，新的局部变量和参数在再入栈中重新分配存储单元，并以新的变量重新开始执行。每次递归调用返回时，前面压入的局部变量和参数会从再入栈中弹出，并恢复到上次调用自身的地方继续执行。如果是非再入函数进行递归调用，每次调用函数自身时，上次调用时使用的局部变量数据将被覆盖，因而在递归调用结束时不能得到正确的结果。对于上例，如果将函数 fac(n)定义成非再入函数，则程序的运行结果为 0，显然这是不正确的。

采用函数的递归调用可使程序的结构紧凑，但是递归调用要求采用再入函数，以便利用再入栈来保存有关的局部变量数据，从而要占据较大的内存空间。另外递归调用时对函数的处理速度也比较慢，因此一般情况下应尽量避免采用函数递归调用，定义函数时应尽量避免使用再入属性。

六、实验演练

实验：用 AT89C51 实验板做以下实验：计算一个整数的正整数次幂。

实验源程序如下：

```
#include <AT89x51.h>
#include <stdio.h>
int power(int x,int n)
{
  int i,p;
  p=1;
  for(i=1;i<=n;++i)
  p=p*x;
  return(p);
}
void main(void)
{
  int a,b,c;
  SCON= 0x40;                    //串口方式 1
  REN=1;                         //允许接收
  TMOD= 0x20;                    //定时器 1 定时方式 2
  TH1= 0xE6;                     //12MHz 1200 波特率
  TL1= 0xE6;
  TI= 1;
  TR1= 1;                        //启动定时器
  printf("please input X and n:\n");
  scanf("%d%d",&a,&b);
  c=power(a,b);
  printf("\n%d  to the power of %d is:%d",a,b,c);
  while(1);
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

}

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 power. c。对程序进行编译、链接，产生 power. hex 目标文件。

②取下 AT89C51 实验板上的 CPU，插入 RF-810 编辑器，对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接，将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件，将波特率调整为 1200，串口选择为 COM1，并打开串口。

⑤打开 AT89C51 实验板电源，串口调试软件要求输入数据。在串口调试软件的发送区输入 X 的数据(设为 5)，按一下键盘上的回车键，再输入 n 的数据(设为 5)，按一下键盘上的回车键，以使 scanf 函数确认有数据输入。最后，按调试软件中的“发送”按钮，此时在调试软件的接收区会出现如图 7-2 所示的输出结果。

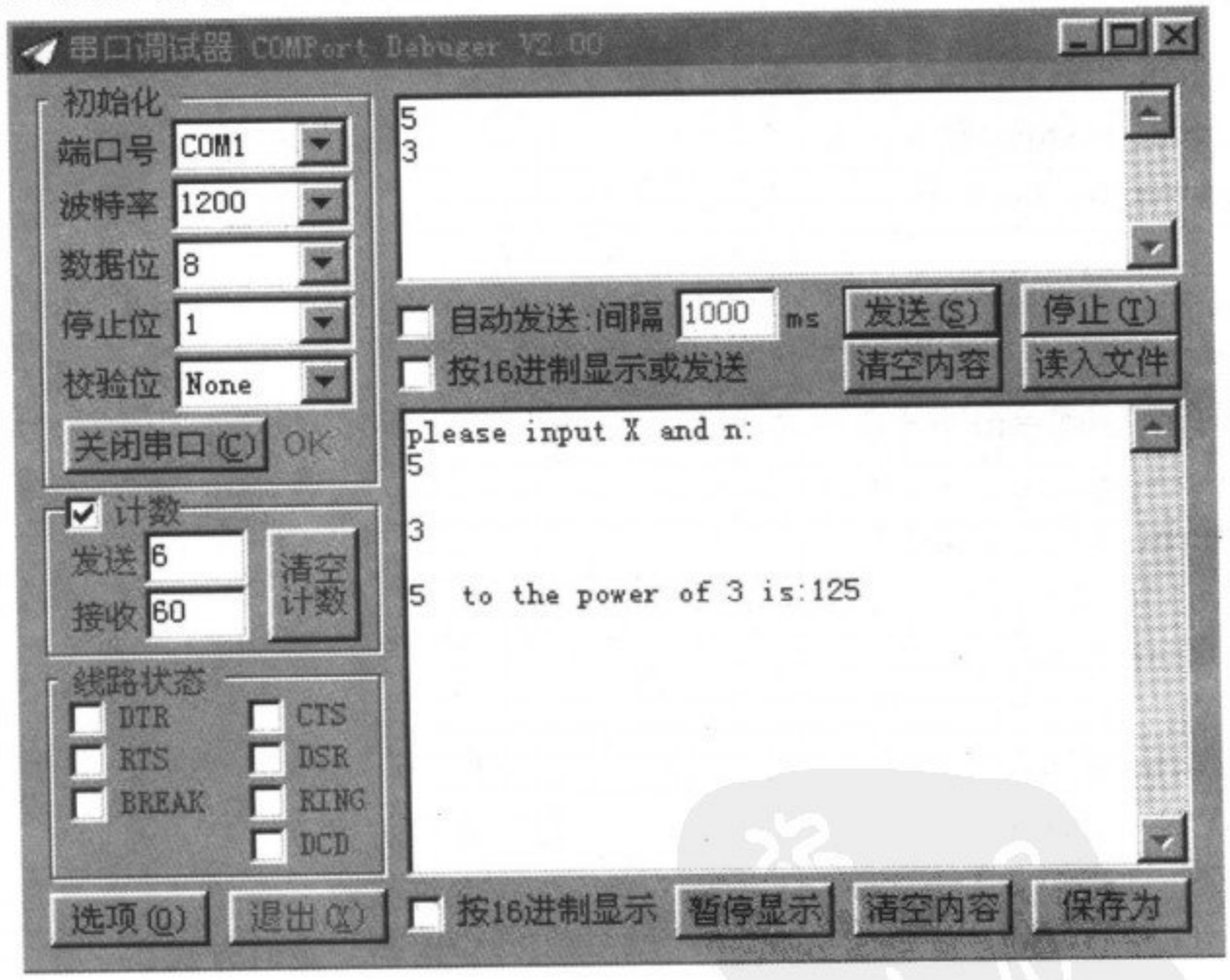


图 7-2　正整数次幂输出结果

该实验程序在本书附赠光盘的 ch_7/power 文件夹中。

重点提示　在这个程序中，定义了一个计算整数的正整数次幂的函数 int power(int x,int n)；它有两个整型的形式参数 x 和 n。在程序开始时，变量 x 和 n 是不占用内存单元的，因此也是没有值的。在主函数 main()中先从键盘输入两个整数值 a 和 b，然后通过函数调用语句 c=power(a,b)；将实际参数 a 和 b 的值传递给被调用函数 power()中的形式参数。调用发生时，形式参数变量 x 和 n 被赋以实际参数 a 和 b 的值，从而使函数 power()能按实际参数的值进行计算。从这个例子可以看到，形式参数和实际参数可以不同名，但它们的类型必须要一致。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

一般情况下，希望通过函数调用使主调用函数获得一个确定的值，这就是函数的返回值。例如，上例中的函数调用语句 c=power(a,b);就是将函数 power()的返回值赋给变量 c。

函数的返回值是通过 return 语句获得的。如果希望从被调用函数中带回一个值到主调用函数，被调用函数中必须包含有 return 语句。

函数返回值的类型确定了该函数的类型，因此在定义一个函数时，函数本身的类型应与 return 语句中变量或表达式的类型一致。例如，上例中 power()函数被定义为 int 类型，return 语句中的变量 p 也被定义为 int 类型。如果函数类型与 return 语句中表达式的值类型不一致，则以函数的类型为准。对于数值数据可以自动进行类型转换，即函数的类型决定返回值的类型。

第四节　函数变量的存储方式和种类

一、变量的存储方式

变量的存储方式可分为“静态存储”和“动态存储”两种。

静态存储变量通常是在变量定义时就分配存储单元并一直保持不变，直至整个程序结束。全局变量即属于此类存储方式。动态存储变量是在程序执行过程中，使用它时才分配存储单元，使用完毕立即释放。典型的例子是函数的形式参数，在函数定义时并不给形参分配存储单元，只是在函数被调用时，才予以分配，调用函数完毕立即释放。如果一个函数被多次调用，则会反复地分配、释放形参变量的存储单元。从以上分析可知，静态存储变量是一直存在的，而动态存储变量则时而存在时而消失。这种由于变量存储方式不同而产生的特性又称变量的生存期。生存期表示了变量存在的时间。生存期和作用域是从时间和空间这两个不同的角度来描述变量的特性，这两者既有联系，又有区别。一个变量究竟属于哪一种存储方式，并不能仅从其作用域来判断，还应有明确的存储种类说明。

二、变量的存储种类

通过前面章节的学习，知道变量有局部变量和全局变量之分。除此之外，变量按存储种类还可分为以下四种：auto(自动变量)、extern(外部变量)、static(静态变量)和 register(寄存器变量)。

1. 自动变量 auto

这种存储类型是 C 语言程序中使用最广泛的一种类型。C 语言规定，在函数内，凡未加存储类型说明的变量均视为自动变量。也就是说自动变量可省去说明符 auto。在前面各章的程序中所定义的变量凡未加存储类型说明符的都是自动变量。例如：

```
{
int i,j,k;
char c;
......
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

}等价于：

```
{
auto int i,j,k;
auto char c;
.....
}
```

自动变量具有以下特点：

①自动变量的作用域仅限于定义该变量的个体内。在函数中定义的自动变量，只在该函数内有效。在复合语句中定义的自动变量只在该复合语句中有效。

②自动变量属于动态存储方式，只有在使用它，即定义该变量的函数被调用时，才给它分配存储单元，开始它的生存期。函数调用结束，释放存储单元，结束生存期。因此函数调用结束之后，自动变量的值不能保留。在复合语句中定义的自动变量，在退出复合语句后也不能再使用，否则将引起错误。

例如以下程序：

```
main()
{
    auto int a;
    printf("\ninput a number:\n");
    scanf("%d",&a);
    if(a>0)
      {
        auto int s,p;
        s=a+a;
        p=a*a;
      }
    printf("s=%d p=%d\n",s,p);
}
```

s,p 是在复合语句内定义的自动变量，只能在该复合语句内有效。而程序的最后 1 行却是退出复合语句之后用 printf 语句输出 s,p 的值，这显然会引起错误。

③由于自动变量的作用域和生存期都局限于定义它的个体内（函数或复合语句内），因此不同的个体中允许使用同名的变量而不会混淆。即使在函数内定义的自动变量，也可与该函数内部的复合语句中定义的自动变量同名。

例如：

```
main()
{
  auto int a,s=100,p=100;
  printf("\ninput a number:\n");
  scanf("%d",&a);
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    if(a>0)
      {
      auto int s,p;
      s=a+a;
      p=a*a;
      printf("s=%d p=%d\n",s,p);
      }
    printf("s=%d p=%d\n",s,p);
}
```

本程序在 main 函数中和复合语句内两次定义了变量 s,p 为自动变量。按照 C 语言的规定,在复合语句内,应由复合语句中定义的 s,p 起作用,故 s 的值应为 a+ a,p 的值为 a×a。退出复合语句后的 s,p 应为 main 所定义的 s,p,其值在初始化时给定,均为 100。从输出结果可以分析出两个 s 和两个 p 虽变量名相同,但却是两个不同的变量。

2. 外部变量 extern

使用存储种类说明符"extern"定义的变量称为外部变量。按照默认规则,凡是在所有函数之前,在函数外部定义的变量都是外部变量,定义时可以不写 extern 说明符。但是,在一个函数体内说明一个已在该函数体外或别的程序模块文件中定义过的外部变量时,则必须使用 extern 说明符。一个外部变量被定义之后,它就被分配了固定的内存空间。外部变量的生存期为程序的整个执行时间,即在程序的执行期间外部变量可被随意使用,当一条复合语句执行完毕或是从某一个函数返回时,外部变量的存储空间并不被释放,其值也仍然保留,因此外部变量属于全局变量。

C 语言允许将大型程序分解为若干个独立的程序模块文件,各个模块可分别进行编译,然后再将它们连接在一起。在这种情况下,如果某个变量需要在所有程序模块文件中使用,只要在一个程序模块文件中将该变量定义成全局变量,而在其他程序模块文件中用 extern 说明该变量是已被定义过的外部变量就可以了。

函数是可以相互调用的,因此函数都具有外部存储种类的属性。定义函数时如果冠以关键字 extern,即将其明确定义为一个外部函数。

例如:

```
extern int fun(char a,b)
```

如果在定义函数时省略关键字 extern,则隐含为外部函数。如果要调用一个在本程序模块文件以外的其他模块文件所定义的函数,则必须用关键字 extern 说明被调用函数是一个外部函数。

3. 静态变量

静态变量的类型说明符是 static。静态变量当然是属于静态存储方式,但是属于静态存储方式的量不一定就是静态变量。例如外部变量虽属于静态存储方式,但不一定是静态变量,必须由 static 加以定义后才能成为静态外部变量,或称静态全局变量。对于自动变量,前面已经介绍它属于动态存储方式。但是也可以用 static 定义它为静态自动变量,或称静态局部变量,从而成为静态存储方式。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

由此看来，一个变量可由 static 进行再说明，并改变其原有的存储方式。

(1)静态局部变量

在局部变量的说明前再加上 static 说明符就构成静态局部变量。

静态局部变量属于静态存储方式，它具有以下特点：

①静态局部变量在函数内定义，但不像自动变量那样，当调用时就存在，退出函数时就消失。静态局部变量始终存在着，也就是说它的生存期为整个源程序。

②静态局部变量的生存期虽然为整个源程序，但是其作用域仍与自动变量相同，即只能在定义该变量的函数内使用该变量。退出该函数后，尽管该变量还继续存在，但不能使用它。

③对基本类型的静态局部变量，若在说明时未赋以初值，则系统自动赋予 0 值。而对自动变量不赋初值，则其值是不定的。根据静态局部变量的特点，可以看出它是一种生存期为整个源程序的量。虽然离开定义它的函数后不能使用，但如再次调用定义它的函数时，它又可继续使用，而且保存了前次被调用后留下的值。因此，当多次调用一个函数且要求在调用之间保留某些变量的值时，可考虑采用静态局部变量。虽然用全局变量也可以达到上述目的，但全局变量有时会造成意外的副作用，因此仍以采用局部静态变量为宜。

例如：

```
main()
{
int i;
void fun();                    //函数说明
for(i=1;i<=5;i++)
fun();
}
void fun()                     //函数定义
{
auto int j=0;
++j;
printf("%d\n",j);
}
```

程序中定义了函数 fun，其中的变量 j 说明为自动变量并赋予初始值为 0。当 main 中多次调用 fun 时，j 均赋初值为 0，故每次输出值均为 1。

现在把 j 改为静态局部变量，程序如下：

```
main()
{
int i;
void fun();
for (i=1;i<=5;i++)
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
fun();
}
void fun()
{
static int j=0;
++j;
printf("%d\n",j);
}
```

由于 j 为静态变量，能在每次调用后保留其值并在下一次调用时继续使用，所以输出值成为累加的结果。读者可自行分析其执行过程。

(2)静态全局变量

全局变量(外部变量)的说明之前再冠以 static 就构成了静态的全局变量。全局变量本身就是静态存储方式，静态全局变量当然也是静态存储方式，这两者在存储方式上并无不同。区别虽在于全局变量(外部变量)的作用域是整个源程序，当一个源程序由多个源文件组成时，全局变量(外部变量)在各个源文件中都是有效的。而静态全局变量则限制了其作用域，即只在定义该变量的源文件内有效，在同一源程序的其他源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用，因此可以避免在其他源文件中引起错误。从以上分析可以看出，把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域，限制了它的使用范围。因此 static 这个说明符在不同地方所起的作用是不同的，应予以注意。

4. 寄存器变量

上述各类变量都存放在存储器内，因此当对一个变量频繁读写时，必须要反复访问内存储器，从而花费大量的存取时间。为此，C 语言提供了另一种变量，即寄存器变量。这种变量存放在 CPU 的寄存器中，使用时，不需要访问内存，而直接从寄存器中读写，这样可提高效率。寄存器变量的说明符是 register。对于循环次数较多的循环控制变量及循环体内反复使用的变量，均可定义为寄存器变量。

需要说明的是，只有局部自动变量和形式参数才可以定义为寄存器变量。因为寄存器变量属于动态存储方式，凡需要采用静态存储方式的量不能定义为寄存器变量。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第八章　Cx51 构造数据类型

前面讲述的字符(char)、整型(int)和浮点型(float)等数据，都属于基本数据类型。C语言还提供了一些扩展的数据类型，它们是对基本数据类型的扩展，称之为构造数据类型。这些按一定规则构成的数据类型有：数组、指针、结构、共用体和枚举等。

第一节　数　组

C语言具有使用户能够定义一组有序数据项的能力，这组有序的数据即数组。数组是一组具有固定数目和相同类型成分分量的有序集合。其成分分量的类型为该数组的基本类型。如整型变量的有序集合称为整型数组，字符型变量的有序集合称为字符型数组。这些整型或字符型变量是各自所属数组的成分分量，称为数组元素。

构成一个数组的各元素必须是同一类型的变量，而不允许在同一数组中出现不同类型的变量。数组有一维、二维、三维和多维数组之分。常用的有一维、二维数组和字符数组。

一、一维数组

1. 一维数组的一般形式

一维数组的一般形式如下：

类型说明符　数组名[常量表达式]；

例如：

```
char ch[10]
```

该例定义了一个一维字符型数组，有10个元素，每个元素由不同的下标表示，分别为ch[0]，ch[1]，ch[2]，…，ch[9]。注意，数组的第一个元素的下标为0，而不是1，即数组的第一个元素是ch[0]而不是ch[1]，而数组的第十个元素为ch[9]。

对于数组，再说明以下几点：

①数组的类型实际上是指数组元素的取值类型。对于同一个数组，其所有元素的数据类型都是相同的。

②数组名的书写规则应符合标识符的书写规定。

③数组名不能与其他变量名相同。

例如：

```
void main()
{
int a;
float a[10];
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

……

}

是错误的。

2. 一维数组的初始化

所谓数组初始化，就是在定义说明数组的同时，给数组赋新值。对一维数组的初始化，可用以下方法实现：

①在定义数组时对数组的全部元素赋予初值。

例如：

int idata a[6]={0,1,2,3,4,5};

在上面进行的定义和初始化中，将数组的全部元素的初值依次放在花括号内。这样，在初始化后，a[0]=0，a[1]=1，a[2]=2，…，a[5]=5。

②只对数组的部分元素初始化。

例如：

int idata a[10]={0,1,2,3,4,5};

上面定义的 a 数组共有 10 个元素，但花括号内只有 6 个初值，则数组的前 6 个元素被赋予初值，而后 4 个元素的值为 0。

③在定义数组时，若不对数组的全部元素赋初值，则数组的全部元素被缺省地赋值为 0。

例如：

int idata a[10];

则 a[0]～a[9]全部被赋初值 0。

另外，C 语言规定：通过赋初值可用来定义数组的大小，这时数组说明符的一对方括号可以不指定数组的大小，例如：

int a[]={1,1,1,1};

以上语句一对花括号中出现了 4 个 1，它隐含地定义了 a 数组含有 4 个元素，此定义语句类似于以下语句：

int a[4]={1};

重点提示 数组元素类似于单个变量，与单个变量相比具有以下特殊之处：

①数组元素是通过数组名加上该元素在数组中的位置（下标）来访问的。

②数组元素的赋值是逐个进行的。

③数组名 a 代表的是数组 a 在内存中的首地址，因此，可以用数组名 a 来代表数组元素 a[0]的地址。

3. 一维数组的查表功能

数组的一个非常有用的功能之一就是查表。在许多嵌入式控制系统应用中，使用查表法不但比采用复杂的数学方法有效，而且执行起来速度更快，所用代码较少。表可以事先计算好后装入 EEPROM 中。

例如：以下程序可以将摄氏温度转换成华氏温度：

```
unsigned char code tempt[]={32,34,36,37,39,41};//数组
unsigned char chang(unsigned char degc)
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
{
return tempt[degc]  //返回华氏温度值
main()
{
x=chang(5);          //得到与5℃相应的华氏温度值
}
```

在程序的开始处，unsigned char tempt[]={32,34,36,37,39,41};定义了一个无符号字符型数组 tempt[]，并对其进行初始化，将摄氏温度 0,1,2,3,4,5 对应的温度 32,34,36,37,39,41 赋予数组 tempt[]，类型代码 code 指定编译器将此表定位在 EEPROM 中。

在主程序 main()中调用函数 chang(unsigned char degc)，从 tempt[]数组中查表获取相应的温度转换值。x=chang(5);执行后，x 的结果为与 5℃相应的华氏温度 41°F。

二、二维数组

1. 二维数组的一般形式

二维数组的一般形式是：

类型说明符数组名[常量表达式 1][常量表达式 2];

其中常量表达式 1 表示第一维下标的长度，常量表达式 2 表示第二维下标的长度。

例如：

int a[3][4];

说明了一个三行四列的数组，数组名为 a，其下标变量的类型为整型。该数组的下标变量共有(3×4)个，即：

a[0][0],a[0][1],a[0][2],a[0][3]

a[1][0],a[1][1],a[1][2],a[1][3]

a[2][0],a[2][1],a[2][2],a[2][3]

二维数组在概念上是二维的，也就是说其下标在两个方向上变化，下标变量在数组中的位置也处于一个平面之中，而不是像一维数组只是一个向量。但是，实际的硬件存储器却是连续编址的，也就是说存储器单元是按一维线性排列的。在一维存储器中存放二维数组是按行排列的，即放完一行之后顺次放入第二行。也就是说，先存放 a[0]行，再存放 a[1]行，最后存放 a[2]行。每行中有 4 个元素也是依次存放。由于数组 a 说明为 int 类型，该类型占两个字节的内存空间，所以每个元素均占有两个字节。

2. 二维数组的初始化

(1)所赋初值个数与数组元素的个数相同

可以在定义二维数组的同时给二维数组的各元素赋初值。例如：

int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

全部初值括在一对花括号中，每一行的初值又分别括在一对花括号中，之间用逗号隔开。

(2)每行所赋初值个数与数组元素的个数不同

当某行一对花括号内的初值个数少于该行中元素的个数时，例如：

int a[3][4]={{1,2,3},{4,5,6},{9,10,11}};

系统将自动给该行后面的元素补初值 0。因此，a[0][3]、a[1][3]、a[2][3]的初值为

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

0。也就是说，不能跳过每行前面的元素而给后面的元素赋初值。

(3)所赋初值行数少于数组行数

当代表着给每行赋初值的行花括号对少于数组的行数时，例如：

int a[3][4]={{1,2},{4,5}};

系统将自动给后面各行的元素补初值0。

(4)赋初值时省略行花括号对

在给二维数组赋初值时可以不用行花括号对。例如：

int a[3][4]={1,2,3,4};

在编译时，系统将按a数组元素在内存中排列的顺序，将花括号内的数据一一对应地赋给各个元素；若数据不足，系统将给后面的元素自动补初值0。以上将给a数组第一行中的元素依次赋予1、2、3、4，其他元素中的初值都为0。

注意事项 对于一维数组，可以在数组定义语句中省略方括号中的常量表达式，通过所赋初值的个数来确定数组的大小；对于二维数组，只可以省略第一个方括号中的常量表达式，而不能省略第二个括号中的常量表达式。例如：

int a[][4]={{1,2,3},{4,5},{8}};

以上语句中，a数组的第一维的方括号中的常量表达式省略，在所赋初值中，含有3个行花括号对，则第一维的大小由所赋初值的行数来决定。因此，它等同于：

int a[3][4]= {{1,2,3},{4,5},{8}};

三、字符数组

1. 字符数组的一般形式

用来存放字符量的数组称为字符数组。字符数组类型的一般形式与前面介绍的数值数组相同。例如：char c[10]；由于字符型和整型通用，也可以定义为int c[10]，但这时每个数组元素占两个字节的内存单元。字符数组也可以是二维或多维数组，例如：char c[5][10]；即为二维字符数组。

2. 字符数组的初始化

字符数组置初值的最直接的方法是将各字符逐个赋给数组中的各个元素。

例如：

char a[10]={'B','E','I','','J','I','N','G','\0'};

定义了一个字符型数组a[]，有10个数组元素，并且将9个字符(其中包括一个字符串结束标志'\0')分别赋给了a[0]～a[8]，剩余的a[9]被系统自动赋予空格字符。

C语言还允许用字符串直接给字符数组置初值。其方法有以下两种形式：

char a[10]={"BEI JING"};

char a[10]="BEI JING";

用双引号" "括起来的一串字符，称为字符串常量，比如"Happy"。C编译器会自动地在字符末尾加上结束符'\0'(NULL)。

用单引号''括起来的字符为字符的ASCII码值，而不是字符串。比如'a'表示a的ASCII码值97；而"a"表示一个字符串，由两个字符a和\0组成。

一个字符串可以用一维数组来装入，但数组的元素数目一定要比字符多一个，以便C

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

编译器自动在其后面加入结束符'\0'。

若干个字符串可以装入一个二维字符数组中，数组的第一个下标是字符串的个数，第二个下标定义每个字符串的长度。该长度应当比这批字符串中最长的串多一个字符，用于装入字符串的结束符'\0'。比如 char a[20][31]，定义了一个二维字符数组 a，可容纳 20 个字符串，每串最长可达 30 个字符。

除了上述用字符串赋初值的办法外，还可用 printf 函数和 scanf 函数一次性输出输入一个字符数组中的字符串。

例如：

```
main()
{
char c[10];
scanf("%s",c);
printf("%s\n",c);
while(1);
}
```

程序中用"%s"格式控制输入输出字符串，这里的输入输出操作是对整个字符数组进行的，因此输入项必须是数组名 c，而不能用数组元素名 c[i]。在 Keil 环境下对程序编译链接通过后进行仿真调试，启动程序全速运行，将光标移到串行窗口，从键盘输入 HELLO 并回车，系统会自动在输入的字符串后面加一个结束符'0'，然后输出 HELLO。如果输入的字符数大于 10，则只取前 10 个字符作为有效字符输出。

注意事项 当程序中设定了一个数组时，C 编译器就会在系统的存储空间中开辟一个区域，用于存放该数组的内容。数组就包含在这个由连续存储单元组成的模块的存储体内。对字符数组而言，占据了内存中一串连续的字节位置。对整型(int)数组而言，将在存储区中占据一串连续的字节对的位置。对长型(long)或浮点型(float)数组，一个成员将占有 4 字节的存储空间。对于多维数组来说，一个 10×10×10 的三维浮点数组需要大约 4KB 的存储空间，而一个 25×25×25 的三维浮点数组就需要大于 64KB 的存储空间(8051 单片机的最大可寻址空间只有 64KB)。

当数组特别是多维数组中大多数元素没有被有效地利用时，就会浪费大量的存储空间。8051 单片机这样的嵌入式控制器，不像复用式系统那样拥有大型的存储区。其存储资源极为有限，因此无论如何不能被不必要地占用。因此在进行 Cx51 编程开发时，要仔细地根据需要来选择数组的大小。

四、实验演练

实验 1：用下载型实验做流水灯实验。

实验源程序如下：

```
#include <reg51.h>
void Delay(unsigned int i)                    //延时程序，i 是形式参数
{
    unsigned int j;
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    for(;i>0;i--)                    //变量 i 由实际参数传入一个值
    for(j=0;j<125;j++)
        {;}
}
void main(void)
{
  unsigned char design[32]={0xFF,0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,
  0x7F,0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE,0xFF,0xFF,0xFE,
  0xFC, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x0, 0xE7, 0xDB, 0xBD, 0x7E,
  0xFF};                           //定义花样数据
  unsigned char a;                 //尽量使用少字节的类型
  do
    {
      for (a=0; a<32; a++)
        {
          Delay(200);              //延时 0.2s
          P1 = design[a];          //读已定义的花样数据并写花样数据到 P1 口
        }
    }
    while(1);
  }
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 liuLED.c。对程序进行编译、链接和调试,产生 liuLED.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target ',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_8/liuLED 文件夹中。

实验 2:用 AT89C51 实验板做如下实验:求具有 10 个元素的一维数组中的最大元素。

实验源程序如下:

```
#include <AT89x51.h>
#include <stdio.h>
max(int array[])
  {
  int   i,maxnum;
  maxnum=array[0];
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
  for(i=0;i<10;i++)
  if (array[i]>maxnum) maxnum=array[i];
  return(maxnum);
  }
void main()
{
  int a[10]={1,2,3,4,5,10,9,8,7,6};
  SCON=0x40;                          //串口方式 1
  REN=1;                              //允许接收
  TMOD=0x20;                          //定时器 1 定时方式 2
  TH1=0xE6;                           //12MHz 1200 波特率
  TL1=0xE6;
  TI=1;
  TR1=1;                              //启动定时器
  printf("maxnum is   %d\n",max(a));
  while(1);
}
```

或：

```
#include <AT89x51.h>
#include <stdio.h>
void main()
{
  int  i,maxnum;
  int a[10]={1,2,3,4,5,10,9,8,7,6};
  SCON=0x40;                          //串口方式 1
  REN=1;                              //允许接收
  TMOD=0x20;                          //定时器 1 定时方式 2
  TH1=0xE6;                           //12MHz 1200 波特率
  TL1=0xE6;
  TI=1;
  TR1=1;                              //启动定时器
  maxnum=a[0];
  for(i=0;i<10;i++)
  {if (a[i]>maxnum) maxnum=a[i];}
  printf("maxnum is   %d\n",maxnum);
  while(1);
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 max.c。对程序进行编译、链接，产生

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

max. hex 目标文件。

②取下 AT89C51 实验板上的 CPU，插入 RF-810 编辑器，对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接，将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件，将波特率调整为 1200，串口选择为 COM1，并打开串口。

⑤打开 AT89C51 实验板电源，此时在调试软件的接收区会出现如图 8-1 所示的输出结果。

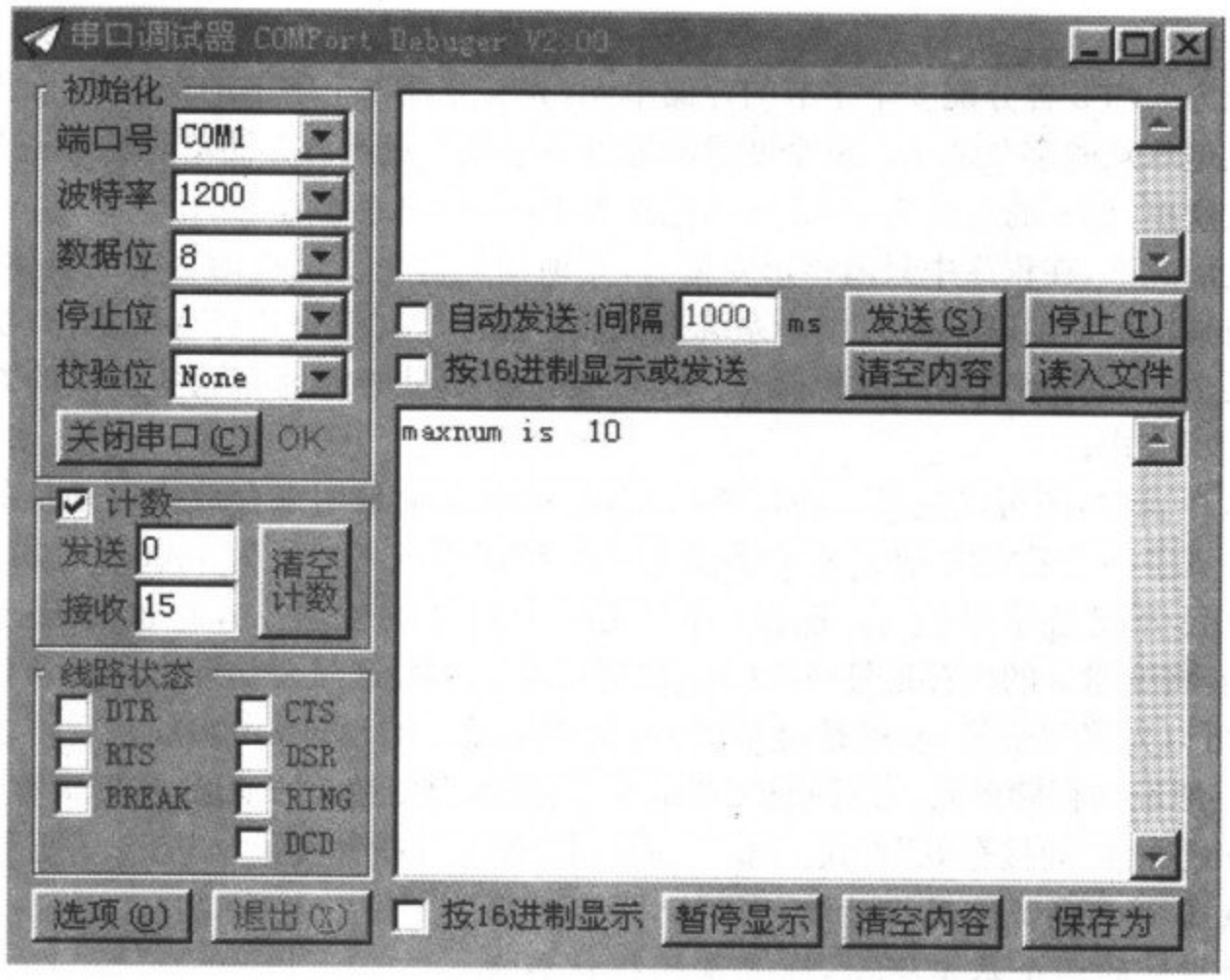


图 8-1　求数组最大数输出结果

第二节　指　针

指针是 C 语言中广泛使用的一种数据类型。运用指针编程是 C 语言最主要的风格之一。利用指针变量可以表示各种数据结构；能很方便地使用数组和字符串；并能像汇编语言一样处理内存地址，从而编出精练而高效的程序。指针极大地丰富了 C 语言的功能。学习指针是学习 C 语言中最重要的一环，能否正确理解和使用指针是是否掌握 C 语言的一个标志。同时，指针也是 C 语言中最为困难的一部分。

一、指针的基本概念

计算机的内存是以字节为单位的一片连续的存储空间，每一个字节都有一个编号，这个编号就称为内存地址。就像旅馆的每个房间都有一个房间号一样，如果没有房间号，旅馆的工作人员就无法进行管理；同样的道理，没有内存字节的编号，系统就无法对内存进

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

行管理。因为内存的存储空间是连续的，内存中的地址号也是连续的，并且用二进制数来表示，为了直观起见，这里用十进制数进行描述。

若在程序中定义了一个变量，C 编译系统就会根据定义中变量的类型，为其分配一定字节数的内存空间(如：字符型占 1 字节，整型占 2 字节，实型占 4 字节)，此后，这个变量的内存地址也就确定了。例如，若有定义：int a,b; float x;这时，将如图 8-2 所示。

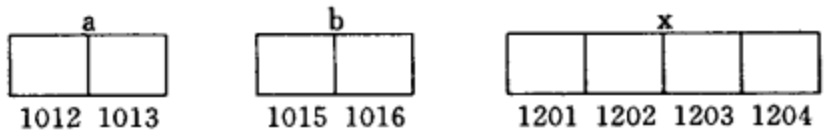


图 8-2 内存单元的分配

系统为 a 和 b 各分配 2 个字节的存储单元，为 x 分配 4 个字节的存储单元，图 8-2 中的数字只是示意的字节地址。每个变量的地址是指该变量所占存储单元的第一个字节的地址。在这里，称 a 的地址为 1012，b 的地址为 1015，x 的地址为 1201。

一般情况下，在程序中只需指出变量名，无须知道每个变量在内存中的具体地址，每个变量与具体地址的联系由 C 编译系统来完成。程序中对变量进行存取操作，实际上也就是对某个地址的存储单元进行操作。这种直接按变量的地址存取变量值的方式称为“直接存取”方式。

在 C 语言中，还可以定义一种特殊的变量，这种变量是用来存放内存地址的。假设程序中定义了一个整型变量 a，它们的值为 6，C 编译系统将地址为 1000 和 1001 的 2 字节内存单元分配给了变量 a。现在，再定义一个这样的变量 ap，它也有自己的地址(2010)；若将变量 a 的内存地址(1000)存放到变量 ap 中，这时要访问变量 a 所代表的存储单元，可以先找到变量 ap 的地址(2010)，从中取出 a 的地址(1000)，然后再去访问以 1000 为首地址的存储单元。这种通过变量 ap 间接得到变量 a 的地址，然后再存取变量 a 的值的方式称为“间接存取”方式。ap 称为指向变量 a 的指针变量，如图 8-3 所示。

重点提示 为了使用指针进行间接访问，必须弄清关于指针的两个基本概念，即变量的指针和指向变量的指针变量(简称指针变量)。

变量的指针就是变量的地址。对于上面提到的变量 a 而言，其指针就是 1000。

若有一个变量专门用来存放另一个变量的地址(即指针)，则该变量称为指向变量的指针变量(简称指针变量)。指针变量的值是指针。上面提到的地址为 2010 的内存单元，如果定义一个变量 ap，并使其定位在地址为 2010 的这个内存单元上，则 ap 就是一个指针变量。因为 ap 中，即 2010 址址单元中存放着变量 a 的地址 1000。

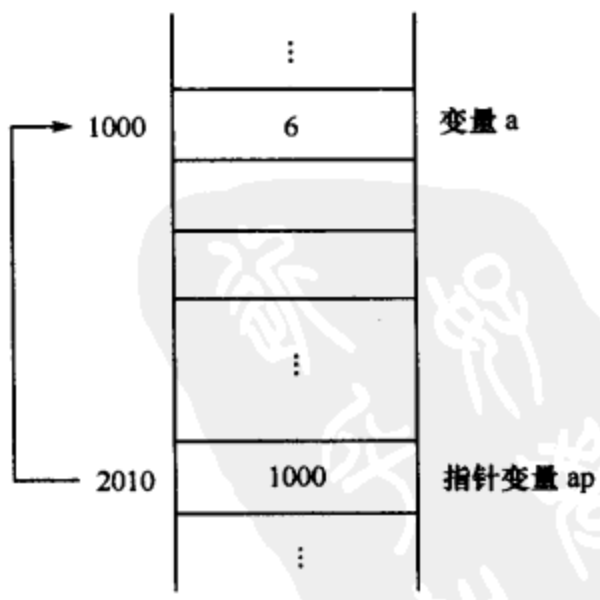


图 8-3 指针变量

可以说变量 a 的指针(地址)为 1000，不能说 a 的指针变量是 1000。变量 a 的指针变量应该是 ap，ap 的指针是 2010。

为了帮助读者对指针变量和变量有一个通俗的认识，可以这样打个比方，有一间房子，它的地址是人民路 108 号。这个房子相当于一个变量。那么：

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

①如果它是普通变量，则房子里可能今天住的是张三，明天住的是李四。张三、李四就是这个变量的值。通过访问这间房子，可以直接找到张三或李四。

②如果它是一个指针变量，则房子里不住具体的人，而是放一张纸条，上面写："南京东路 77 号"。"南京东路 77 号"是一个什么东西？是一个地址。通过该地址继续找，结果在"南京东路 77 号"里找到张三。

变量的存储的值可以改变，指针变量的值同样可以变更：

过一天，再去访问这个房子，纸条变了"珠海路 309 号"，通过它，找到的是另一个人。

二、指针变量的定义

C 语言规定，所有的变量在使用之前必须先定义，以确定其类型。指针变量也不例外，由于它是用来专门存放地址的，因此必须将它定义为"指针类型"。

指针定义的一般形式为：

数据类型 * 指针变量名；

例如：

```
int * ap;
```

要同时定义多个相同类型的指针，则为：

```
int * ap, * bp;
```

注意，每个指针变量名之前都必须有 * 。

三、初始化指针变量

一个指向不明的指针，是危险的。很多软件有 BUG，其最终原因，就是存在指向不明的指针。

设有如下语句：

```
int a=10,b=20,c=30;
int * ap, * bp, * cp;
ap=&a,bp=&b,cp=&c;
```

第 1 行定义了 3 个整型变量 a，b，c。

第 2 行定义了 3 个整型指针变量 ap，bp，cp。

而第 3 行，"指针变量 ap 存储了变量 a 的地址，指针变量 bp 存储了变量 b 的地址，指针变量 cp 存储了变量 c 的地址"。即"ap 指向了 a，bp 指向了 b，cp 指向了 c"。语句中的 & 为取地址运算符。

执行了上面 3 行代码后，结果是：ap 指向 a，bp 指向 b，cp 指向 c。

重点提示 当进行完变量、指针变量定义之后，如果对这些语句进行编译，那么 C 编译器就会给每一个变量和指针变量在内存中安排相应的内存单元。然而，这些单元的地址除非使用特殊的调试程序，否则是看不到的。为了能清楚地说明问题，假设 C 编译器将地址为 1000 和 1001 的 2 字节内存单元指定给变量 a 使用，将地址为 1002 和 1003 的 2 字节内存单元指定给变量 b 使用，将地址为 1004 和 1005 的 2 字节内存单元指定给变量 c 使用。同理指针变量 ap 的地址为 2010，指针变量 bp 的地址为 2012，指针变量 cp 的地址为 2014。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

通过取地址运算符运算操作后，指针 ap 就指向了变量 a，即指针变量 ap 地址单元中装入了变量 a 的地址 1000；指针变量 bp 指向了变量 b，即指针变量 bp 的地址单元中装入了变量 b 的地址 1002；而指针变量 cp 指向了变量 c，即指针变量 cp 的地址单元中装入了变量 c 的地址 1004。具体情形如图 8-4 所示。

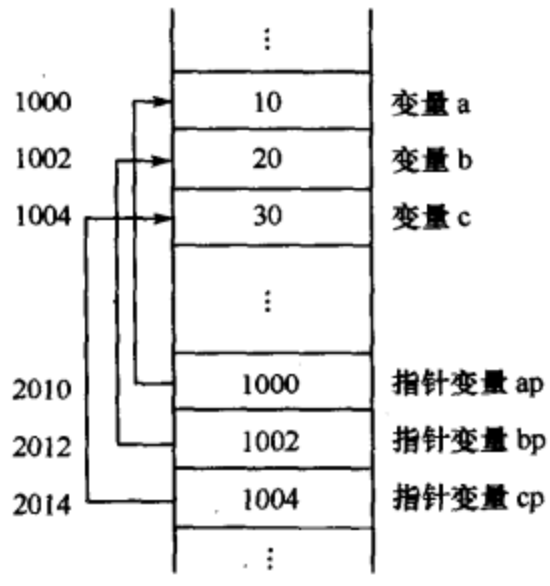


图 8-4　指针示意图

四、指针变量的运算

指针变量可以进行某些运算，但其运算的种类是有限的。它只能进行赋值运算和部分算术运算及关系运算。

1. 指针运算符

(1)取地址运算符 &

取地址运算符 & 是单目运算符，其结合性为自右至左，其功能是取变量的地址。

(2)取内容运算符 *

取内容运算符 * 是单目运算符，其结合性为自右至左，用来表示指针变量所指的变量。在 * 运算符之后跟的变量必须是指针变量。需要注意的是，取内容运算符 * 和指针变量说明中的指针说明符 * 不是一回事。在指针变量说明中，“ * ”是类型说明符，表示其后的变量是指针类型。而表达式中出现的“ * ”则是一个取内容运算符，用以表示指针变量所指的变量。

例如：

```
main()
{
int a=5, * ap;
ap=&a;
printf ("%d", * ap);
}
```

表示指针变量 ap 取得了整型变量 a 的地址。本语句表示输出变量 a 的值。

2. 指针变量的赋值运算

指针变量的赋值运算有以下几种形式：

(1)把一个变量的地址赋予指向相同数据类型的指针变量

例如：

```
int a, * ap;
ap=&a;
```

把整型变量 a 的地址赋予整型指针变量 ap。

例：输入两个整数 x 和 y，经比较后按大小顺序输出。

解：根据要求，编写源程序如下：

```
#include <stdio. h>
main()
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
{
int x,y;
int *p,*pl,*p2;
printf("Input x and y:\n");
scanf("%d  %d",&x,&y);
pl=&x;
p2=&y;
if(x<y)
  {
   p=pl;
   pl=p2;
   p2=p;
   printf("max=%d,min=%d,\n",*pl,*p2);
   while(1);
  }
```

程序执行结果：

```
Input x and y:
10  20  回车
max=20,min=10
```

这个程序中定义了 3 个指针变量 *p，*pl 和 *p2，它们都指向整型变量。经过赋值之后，pl 指向 x，p2 指向 y，然后比较变量 x 和 y 的大小，若 x<y，则将 pl 和 p2 交换，使 pl 指向 y，p2 指向 x；若 x>y，则不交换。最后的结果必然使指针 pl 指向较大的数，p2 指向较小的数，按顺序输出 *pl 和 *p2 的值，就可得到正确的结果。

值得注意的是，在程序执行过程中，变量 x 和 y 的值并未交换，所交换的只是它们的指针。由于指针 p、pl 和 p2 都是指向 int 类型数据的指针，故可以相互赋值，实现指针 pl 和 p2 的交换。

(2)把一个指针变量的值赋予指向相同类型变量的另一个指针变量

例如：

```
int a,*ap,*bp;
ap=&a;
bp=ap;
```

把 a 的地址赋予指针变量 bp。

由于 ap、bp 均为指向整型变量的指针变量，因此可以相互赋值。

(3)把数组的首地址赋予指向数组的指针变量

例如：

```
int a[5],*ap;
ap=a;(数组名表示数组的首地址，故可赋予指向数组的指针变量 ap)
```

也可写为：

```
ap=&a[0];
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

数组第一个元素的地址也是整个数组的首地址，也可赋予 ap，当然也可采取初始化赋值的方法：

```
int a[5], * ap=a;
```

(4)把字符串的首地址赋予指向字符类型的指针变量

例如：

```
char  * cp;
cp="c language";
```

或用初始化赋值的方法写为：

```
char  * pc="C Language";
```

这里应说明的是，并不是把整个字符串装入指针变量，而是把存放该字符串的字符数组的首地址装入指针变量。

3. 指针变量的加减算术运算

对于指向数组的指针变量，可以加上或减去一个整数 n。设 pa 是指向数组 a 的指针变量，则 pa+n，pa-n，pa++，++pa，pa--，--pa 运算都是合法的。指针变量加或减一个整数 n 的意义是把指针指向的当前位置(指向某数组元素)向前或向后移动 n 个位置。应该注意，数组指针变量向前或向后移动一个位置和地址加 1 或减 1 在概念上是不同的。因为数组可以有不同的类型，各种类型的数组元素所占的字节长度是不同的。如指针变量加 1，即向后移动 1 个位置，表示指针变量指向下一个数据元素的地址，而不是在原地址基础上加 1。

例如：

```
int a[5], * ap;
ap=a;          //ap 指向数组 a，也是指向 a[0]
ap=ap+1;       //ap 指向 a[1]，即 ap 的值为 &a[1]
```

指针变量的加减运算只能对数组指针变量进行，对指向其他类型变量的指针变量作加减运算是毫无意义的。

4. 指针运算说明

若先使指针变量 ap 指向数组 a[]的首地址，即 ap=a 或 ap=&a[0]，则指针运算符含义如下：

(1)ap++

其作用将使指针变量 ap 指向下一个数组元素 a[1]，若再执行 x= * ap，则将取出 a[1]的值，将其赋给变量 x。

(2)(* ap)++

(* ap)++等价于 * ap= * ap+1 或 * ap+=1，其作用是取指针变量 ap 所指的存储单元中的值，加 1 后再放入 ap 所指向的存储单元中，即使得 a[0]中的值增 1(若 a[0]中原来为 100，则增 1 后变为 101)。

重点提示

若有以下定义：

```
int  * ap;
 * ap=100;
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

*ap=*ap+1;

执行后，则指针变量ap所指的存储单元中的值增1，变为101。

可见，*ap出现在赋值号左边时，代表的是指针所指的存储单元，当*ap出现在赋值号右边时，代表的是指针所指的存储单元的内容。

(3) *ap++

由于++与*运算符优先级相同，而结合方向为自右向左，故*ap++等价于*(ap++)。其作用是先得到p指向的变量的值(即*ap)，然后再执行ap自加运算(并不使ap所指的存储单元中的值增1)。

例如：

```
for(i=0;i<10;i++)
{
printf("%d",*ap);
ap++;
}
```

等价于：

```
 for(i=0;i<10;i++)
 printf("%d",*ap++);
```

(4) *ap++与*++ap

ap++与++ap作用不同，前者先取*ap值，后使ap自加1；后者先使ap自加1，再取*ap值。若ap的初值为&a[0]，则执行x=*ap++时，x值为a[0]的值，而执行x=*++ap后，x值等于a[1]的值。

五、指针和数组

1. 一维数组和指针

指针和数组有着密切的关系，任何能由数组下标完成的操作也都可用指针来实现，但程序中使用指针可使代码更紧凑、更灵活。

定义一个一维整型数组和一个指向整型的指针变量如下：

```
int a[10],*ap;
```

和前面介绍过的方法相同，可以使整型指针ap指向数组中任何一个元素，假定给出赋值运算：

```
ap=&a[0];
```

此时，ap指向数组中的第0号元素，即a[0]，指针变量ap保存了数组元素a[0]的地址。由于数组元素在内存中是连续存放的，因此，可以通过指针变量ap及其有关运算间接访问数组中的任何一个元素。

在C语言中，数组名是数组的第0号元素的地址，因此ap=&a[0]与ap=a是等价的。

根据地址运算规则，a+1为a[1]的地址，a+i就是a[i]的地址。

下面给出数组元素的地址和内容用指针来表示的几种形式(假定ap指向a[0])：

①ap+i和a+i均表示a[i]的地址；或者说，它们均指向数组第i号元素，即指向a[i]。

②*(ap+i)和*(a+i)都表示ap+i和a+i所指对象的内容，即为a[i]。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

③允许指向数组元素的指针变量带下标，如 ap[i]与 *(ap+i)等价。

例：设一个整型数组 a，有 10 个元素。要求输出全部元素的值。

解：要输出数组的全部元素的值有 3 种方法。

①下标法。程序如下：

```
main()
{
int a[10]{1,2,3,4,5,6,7,8,9,10};
int i;
for(i=0;i<10,i++)
printf("%d",a[i]);
```

②通过数组名计算数组元素的地址，找出元素的值。程序如下：

```
main()
{
int a[10]={1,2,3,4,5,6,7,8,9,10};
int i;
for(i=0;i<10;i++)
printf("%d", *(a+i));
}
```

③用指针变量指向数组元素。程序如下：

```
main()
{
int a[10]={1,2,3,4,5,6,7,8,9,10};
int    *ap;
for(ap=a;ap-a<10;ap++)
printf("%d", *ap);
}
```

以上 3 个程序的运行结果均为

1　2　3　4　5　6　7　8　9　10

在这个例子中，第①、②种方法执行的效率相同。C 编译器在进行编译时，就是将 a[i]转换成 *(a+i)来处理的。由于要进行元素的地址计算，因此第①、②种方法找数组元素很费时间，而第③种方法由于用指针变量直接指向数组元素，不必每次都重新计算地址，因此像 ap++这样的自加计算比第①、②种方法要快得多，能大大提高运算的效率。

2. 二维数组和指针

(1)通过地址来引用二维数组

为了说明问题，定义以下二维数组：

```
int a[3][4]={{0,1,2,3},{4,5,6,7},{8,9,10,11}};
```

a 为二维数组名，此数组有 3 行 4 列，共 12 个元素。但也可这样来理解，数组 a 由 3 个元素组成：a[0]，a[1]，a[2]。而每个元素又是一个一维数组，且都含有 4 个元素(相当于 4 列)，例如，a[0]所代表的一维数组所包含的 4 个元素为 a[0][0]，a[0][1]，a[0][2]，a

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

[0][3],如图 8-5 所示。

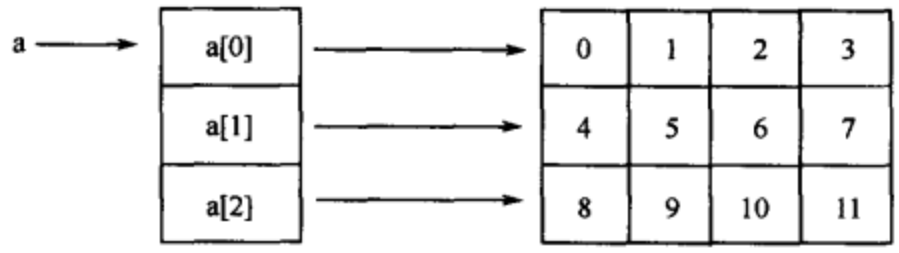


图 8-5　把 a[3][4]理解成 3 个一维数组

但从二维数组的角度来看，a 代表二维数组的首地址，当然也可看成是二维数组第 0 行的首地址。a+1 就代表第 1 行的首地址，a+2 就代表第 2 行的首地址。

如果此二维数组的首地址为 1000，由于第 0 行有 4 个整型元素，所以 a+1 为 1008，a+2 也就为 1016，如图 8-6 所示。

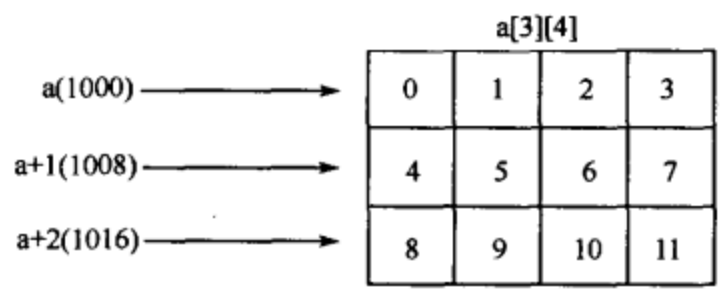


图 8-6　二维数组的行指针

既然把 a[0]，a[1]，a[2]看成是一维数组名，可以认为它们分别代表它们所对应的数组的首地址。例如：a[0]代表第 0 行元素的起始地址，即 &a[0][0]，a[1]是第 1 行元素的起始地址，即 &a[1][0]。

根据地址运算规则，a[0]+1 即代表第 0 行第 1 列元素的地址，即 &a[0][1]。一般而言，a[i]+j 即代表第 i 行第 j 列元素的地址，即 &a[i][j]。

另外，由于 a[0]与 *(a+0)等价，a[1]与 *(a+1)等价，于是，a[i]+j 就与 *(a+i)+j 等价，它表示数组元素 a[i][j]的地址。因此，二维数组元素 a[i][j]可表示成以下几种形式：

a[i][j];

*(a[i]+j);

((a+i)+j);

(*(a+i))[j];

(2)通过建立一个指针数组来引用二维数组元素

若有以下定义：

```
int *p[3],a[3][4],i,j;
```

在这里，说明符 *p[3]中，也遵照运算符的优先级，一对[]的优先级高于 * 号，因此 p 首先与[]结合，构成 p[3]，说明了 p 是一个数组名，系统将为它开辟 3 个连续的存储单元；在它前面的 * 号则说明了数组 p 是指针类型，它的每个元素都是基类型为 int 的指针。若满足条件：0≤i<3，则 p[i]和 a[i]的基类型相同，p[i]=a[i]是合法的赋值表达式。若有以下循环：

```
for(i=0;i<3;i++)
p[i]=a[i];
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

在这里，赋值号右边的 a[i]是常量，表示 a 数组每行的首地址，赋值号左边的 p[i]是指针变量，循环执行的结果使 p[0]，p[1]，p[2]分别指向 a 数组每行的开头。这时，数组 p 和数组 a 之间的关系如图 8-7 所示。

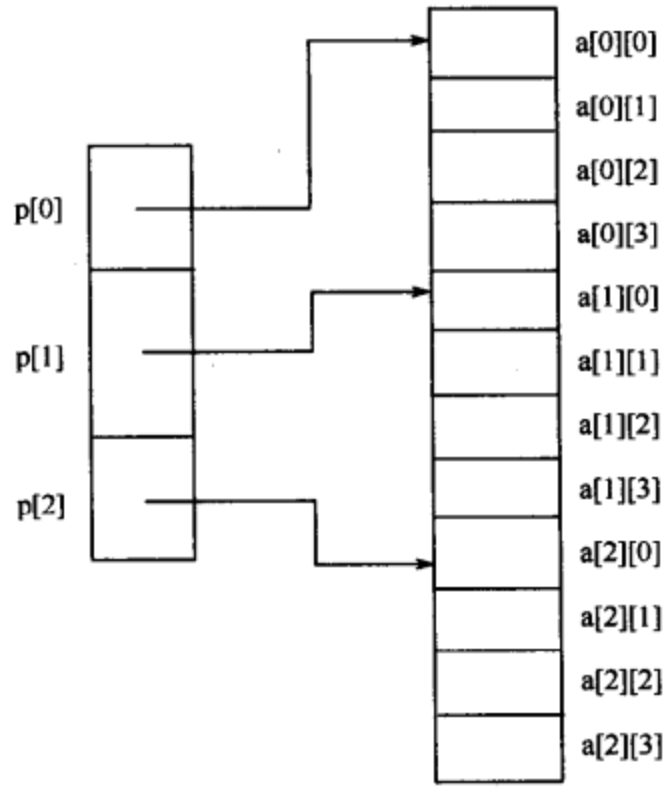


图 8-7　指针数组 p 和数组 a 之间的关系

当 p 数组的每个元素按图 8-7 所示指向 a 数组每行的开头时，则 a 数组元素 a[i][j]的引用形式 *(a[i]+j)和 *(p[i]+j)是完全等价的。由此可见，这时可以通过指针数组 p 来引用 a 数组元素，它们的等价形式如下：

```
p[i][j];                //与 a[i][j]对应
*(p[i]+j);              //与 *(a[i]+j)对应
*(*(p+i)+j);            //与(*(*a+i)+j)对应
(*(p+i))[j];            //与(*(a+i))[j]对应
```

不同的是：p[i]中的值是可变的，而 a[i]中的值是不可变的。

(3)通过建立一个行指针来引用二维数组元素

若有以下定义：

```
int a[3][4],(*prt)[4];
```

在这里，说明符(*prt)[4]中，由于一对圆括号的存在，所以 * 号首先与 prt 结合，说明 prt 是一个指针变量，然后再与说明符[4]结合，说明指针变量 prt 的基类型是一个包含有 4 个 int 元素的数组。在这里，prt 的基类型与 a 的相同，因此 prt=a;是合法的赋值语句。prt+1 等价于 a+1 或 a[1]。当 prt 指向 a 数组的开头时，可以通过以下形式来引用 a[i][j]：

```
prt[i][j];              //与 a[i][j]对应
*(prt[i]+j);            //与 *(a[i]+j)对应
*(*(prt+i)+j);          //与 *(*(a+i)+j)对应
(*(prt+i))[j];          //与(*(a+i))[j]对应
```

在这里，prt 是个指针变量，它的值可变，而 a 是一个常量。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

例如:输出二维数组中任一行、列元素的值。

解:main()

```
{
int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,2l,23}};
int ( * prt)[4],i,j;
ap=a;
i=2;
j=2;
printf("a[%d,%d]=%d\n ",i,j, * ( * (prt+i)+j));
    }
```

运行结果:

a[2,2]=21

六、指针变量作为函数参数

函数的参数不仅可以是整型、实型、字符型等数据,还可以是指针类型的数据。指针变量作为函数的参数的作用是将一个变量的地址传送到另一个函数中去,地址传递是双向的,即主调用函数不仅可以向被调用函数传递参数,而且还可以从被调用函数返回其结果。下面通过一个例子来说明。

例:利用指针变量作为函数的参数实现两个元素的相互交换。

解:源程序如下:

```
#include <stdio. h>
swap(int  * pi,int  * pj)
  {
  int temp;
  temp= * pi;
  * pi= * pj;
  * pj=temp;
  }
main()
{
int a,b;
int  * pa, * pb;
printf("Please input a and b:\n ");
scanf{"%d %d ",&a,&b);
pa=&a;
pb=&b;
if(a<b)
swap(pa,pb);
printf("\n max=%d,min=%d\n ",a,b);
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
while(1);
```

程序执行结果：

```
Please input a and b:
10 20    回车
max=20,min=10
```

程序中，自定义函数 swap()的功能是交换两个变量 a 和 b 的值，swap()函数的两个形式参数 pi 和 pj 是指针变量。程序开始执行时，先输入 a 和 b 的值，然后将 a 和 b 的地址分别赋给指针变量 pa 和 pb，使 pa 指向 a，pb 指向 b。接着执行 if 语句，如果 a＜b，则调用 swap()函数。注意函数调用时的实参 pa 和 pb 也是指针变量，在调用开始时，实参变量将它的值传递给形参变量，采取的仍然是“值传递”方式，但这时传递的是指针的值（即地址）。参数传递完成后，形参 pi 的值为 &a，pj 的值为 &b，即指针变量 * pi 和 * pa 都指向了变量 a，* pj 和 * pb 都指向了变量 b。接着执行 swap()函数体，使 * pi 和 * pj 的值互换，从而实现了 a 和 b 的值的互换。函数返回时，虽然 pi 和 pj 被释放而不再存在，但 main()函数中 a 和 b 的值已经被交换，因此最后能输出正确的结果。

由此可见，以指针变量作为函数的参数，被调用函数在执行过程中使指针变量所指向的变量值发生变化，函数调用结束后，这些变量值的变化将被保留下来，从而可以实现“在被调用函数中改变变量的值，在主调用函数中使用这些被改变了的值”。

七、一般指针和基于存储器的指针

Cx51 编译器支持使用“ * ”字符声明的变量指针，且与标准 C 语言中指针的操作相同。然而由于 8051 和其派生产品的体系结构独特，Cx51 编译器支持两种类型的指针：一般指针和基于存储器的指针。

1. 一般指针

一般指针的声明与标准 C 语言中指针的声明相同，例如：

```
char * p;          //字符指针
int * ap;          //整型指针
long * bp;         //长整型指针
```

一般指针在内存中占用 3 字节：2 字节偏移和 1 字节存储器类型，即

地址	+0	+1	+2
内容	存储器类型	偏移量高位	偏移量低位

其中，第一个字节代表了指针的存储器类型，存储器类型编码如下：

存储器类型	idata/data/bdata	xdata	pdata	code
值	0x00	0x01	0xFE	0xFF

注意 使用其他类型值可能会导致不可预测的程序动作。类型值和编译器的版本有关。

例如：以 xdata 类型的 0x2001 地址作为指针可以表示如下：

地址	+0	+1	+2
内容	0x01	0x20	0x01

一般指针可用于存取任何变量而不必考虑变量在 8051 单片机存储空间中的位置。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

因此，很多 Cx51 库的程序都使用一般指针。函数可通过使用一般指针存取位于任何存储空间的数据。

重点提示 由于一般指针所指对象的存储空间位置只有在运行期间才能确定，编译器在编译期间无法优化存储器的访问方式，必须生成通用代码以保证能对任意空间的对象进行存取，因此，一般指针产生的代码的执行速度较慢。如果系统优先考虑运行速度，那么设计中就要尽可能地用基于存储器的指针代替一般指针。

前面介绍的一般指针 char * p，int * ap，long * bp 都存储在 8051 的内部数据存储器中。然而，也可以使用存储类型说明符为这些一般指针指定具体的存放位置。

```
char * xdata p;          //一般指针存在 xdata
int * data ap;           //一般指针存在 data
long * idata bp;         //一般指针存在 idata
```

上面例子中的变量可以存放在 8051 的任何一个存储区内，而指针分别存储在 xdata，data 和 idata 空间内。

2. 基于存储器的指针

基于存储器的指针在声明中包括存储类型说明，表示指针指向特定的存储区。例如：

```
char data * p;          //指针指向 data 中的字符串
int xdata * ap          //指针指向 xdata 中的整型数
long code * bp;         //指针指向 code 中长整型数
```

由于基于存储器的指针在编译期间即可确定存储类型，因此不必像一般指针那样需要一个存储类型字节。基于存储器的指针在存储时占用 1 字节(idata，data，bdata 和 pdata 指针)或 2 字节(xdata 和 code 指针)。

重点提示 由于基于存储器的指针所指对象的存储空间位置在编译期间即可确定存储类型，因此基于存储器的指针所产生的代码的执行速度快。编译器可以用此信息优化存储器访问。如果系统优先考虑运行速度，那么设计中就要尽可能地用基于存储器的指针代替一般指针。

像一般指针一样，也可以为基于存储器的指针指定存放的位置。其做法是在指针声明前面加上存储类型说明符。例如：

```
char data * xdata  p;     //声明指针存放在 xdata 空间内并指向 data char 变量
int xdata * data ap;      //声明指针存放在 data 空间内并指向 xdata int 变量
long code * idata bp;     //声明指针存放在 idata 空间内并指向 code long 变量
```

基于存储器的指针只用来访问 8051 存储区内的变量，为访问数据对象提供了最为高效的方法，但缺乏灵活性。

八、绝对地址的访问

Cx51 提供了 3 种访问绝对地址的方法。

1. 绝对宏

在程序中，用“#include”即可使用其中定义的宏来访问绝对地址，包括：CBYTE(code 区)、XBYTE(xdata 区)、DBYTE(data 区)、PBYTE(分页寻址 xdata 区)，具体使用参考头文件 absacc.h 中的内容。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

例如：XBYTE[0x8000]=0x41；//将常数值 0x41 写入地址为 0x8000 的外部数据存储器

2. _at_关键字

在 C 程序中，使用关键字_at_就可以将变量存放到指定的绝对存储器位置，一般形式如下：

变量类型 [存储类型] 变量名 _at_常量

其中，存储类型表示变量的存储空间，若声明中省略该项，则使用默认的存储空间。

_at_后的常量用于定位变量的绝对地址，绝对地址必须是位于物理空间范围内，Cx51 编译器会检查非法的地址指定。绝对地址还有以下限定：绝对变量不能被初使化；bit 型函数及变量不能定位在绝对地址。

例如：

unsigned char xdata dis_buff[16] _at_ 0x6020 ；

定位 RAM，将 dis_buff[16]定位在 0x6020 开始的 16 个字节。

3. 连接定位控制

此法是利用连接控制指令进行操作，在此不作讨论。

九、实验演练

实验 1：设两个 unsigned int 变量 ABC、CBA 存放在 0x0028，0x002A 中，指针变量 portA 存放在 0x002C 中，根据以下程序，用 AT89C51 实验板进行实验，观察 *，& 的运算结果，理解指针的用法和含义。

实验源程序：

```
unsigned int data ABC _at_ 0x0028;
unsigned int data CBA _at_ 0x002A;
unsigned int data  * Port _at_ 0x002C;
#include <AT89x51. h>
#include <stdio. h>
void main(void)
{
   SCON=0x40;                    //串口方式 1
   REN=1;                        //允许接收
   TMOD=0x20;                    //定时器 1 定时方式 2
   TH1=0xE6;                     //12MHz 1200 波特率
   TL1=0xE6;
   TI=1;
   TR1=1;                        //启动定时器
   ABC=10;                       //设初值
   CBA=20;
   Port=&CBA;                    //取 CBA 的地址放到指针变量 Port
    * Port=100;                  //更改指针变量 Port 所指向的地址的内容
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
printf("1: CBA=%d\n",CBA);      //显示此时 CBA 的值
Port=&ABC;                      //取 ABC 的地址放到指针变量 Port
  CBA= *Port;                   //把当前 Port 所指的地址的内容赋给变量 CBA
printf("2: CBA=%d\n",CBA);      //显示此时 CBA 的值
  printf("ABC=%d\n",ABC);       //显示 ABC 的值
  while(1);                     //等待,不加此行,调试窗口会反复出现输出
                                结果
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 port.c。对程序进行编译、链接,产生 port.hex 目标文件。

②取下 AT89C51 实验板上的 CPU,插入 RF-810 编辑器,对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接,将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试 comdebug 软件,将波特率调整为 1200,串口选择为 COM1,并打开串口。

⑤打开 AT89C51 实验板电源,此时在调试软件的接收区会出现如图 8-8 所示的输出结果。

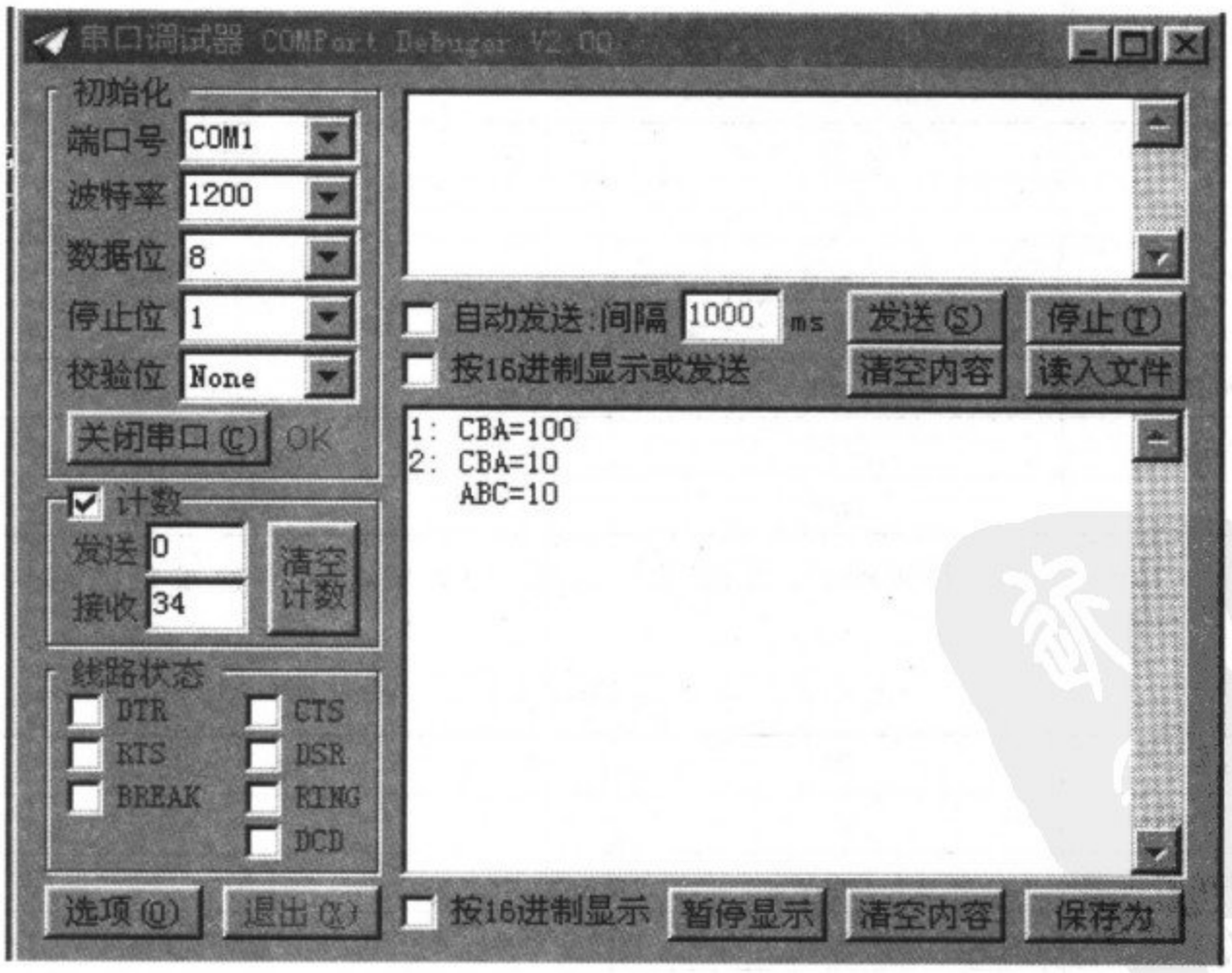


图 8-8 输出的结果

重点提示 执行 ABC = 10;向 ABC 所指的地址 0x28H 写入 10(0xA),因 ABC 是 int 类型,要占用 0x28H 和 0x29H 两个字节的内存空间,低位字节会放入高地址中,所以 0x28H 中放入 0x00,0x29H 中放入 0x0A,如表 8-1 所列。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

表 8-1　执行 ABC=10 的情况

data 地址	值	说　明
0x002DH	0x00	
0x002CH	0x00	
0x002BH	0x00	
0x002AH	0x00	
0x0029H	0x0A	ABC 为 int 类型，占用两字节
0x0028H	0x00	

执行 CBA=20(即 0x14)；原理和上一句一样，如表 8-2 所列。

表 8-2　执行 CBA=20 的情况

data 地址	值	说　明
0x002DH	0x00	
0x002CH	0x00	
0x002BH	0x14	CBA 为 int 类型，占用两字节
0x002AH	0x00	
0x0029H	0x0A	ABC 为 int 类型，占用两字节
0x0028H	0x00	

执行 Port=&CBA；取 CBA 的首地址放到指针变量 Port，如表 8-3 所列。

表 8-3　执行 Port=&CBA 的情况

data 地址	值	说　明
0x002DH	0x00	
0x002CH	0x2A	CBA 的首地址存入 Port
0x002BH	0x14	
0x002AH	0x00	
0x0029H	0x0A	
0x0028H	0x00	

执行 * Port=100(即 0x64)；更改指针变量 Port 所指向的地址的内容，如表 8-4 所列。

表 8-4　执行 * Port=100 的情况

data 地址	值	说　明
0x002DH	0x00	
0x002CH	0x2A	
0x002BH	0x64	Port 指向了 CBA 所在地址 2AH
0x002AH	0x00	并存入 100
0x0029H	0x0A	
0x0028H	0x00	

其他的语句也是一样的道理，大家可以用 Keil 的单步执行和打开存储器查看器看

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

看，这样就更容易理解了。

实验 2：用 AT89C51 实验板做以下实验：用强制类型转换来处理不同类型的数据间运算和赋值。

实验源程序如下：

```
#include <AT89x51.h>
#include <stdio.h>
void main(void)
{
  char data * ROM;
  char D;
  int A=0xFB1C;
  long B=0x893B7832;
  float C=3.4534;
  SCON=0x40;                    //串口方式 1
  REN=1;                        //允许接收
  TMOD=0x20;                    //定时器 1 定时方式 2
  TH1=0xE6;                     //12MHz 1200 波特率
  TL1=0xE6;
  TI=1;
  TR1=1;                        //启动定时器
  ROM=(char data *) 0x0012;     //给指针变量赋 ROM 初值
  *ROM ='R';                    //给 ROM 指向的绝对地址赋值
  D=*((char data *)0x0012);     //等同于 D=*ROM
  printf ("%bx %x %d %c \n",(char) A, (int) B,(int)C,D); //转换类型并输出
  while(1);
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 abc.c。对程序进行编译、链接，产生 abc.hex 目标文件。

②取下 AT89C51 实验板上的 CPU，插入 RF-810 编辑器，对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接，将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件，将波特率调整为 1200，串口选择为 COM1，并打开串口。

⑤打开 AT89C51 实验板电源，此时在调试软件的接收区会出现如图 8-9 所示的输出结果。

重点提示 在上面这段程序中，可以很清楚地看到各种类型进行强制类型转换的基本用法，程序中先在内部数据存储器 DATA 中定义了一个字符型指针变量 ROM，当用 ROM=(char data *) 0x0012 这一语句时，便把 0x0012 这个地址指针赋予了 ROM。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

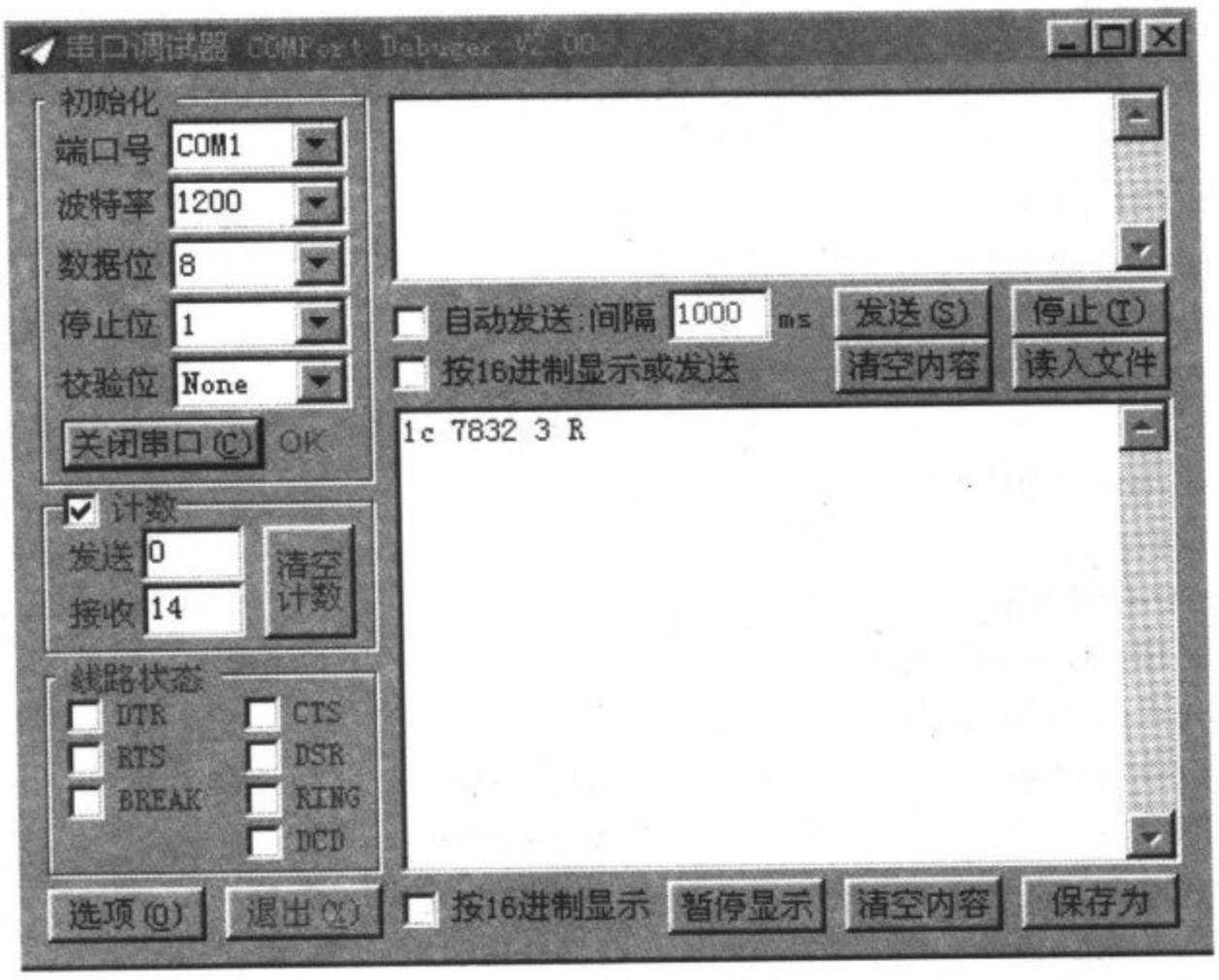


图 8-9　输出的转换结果

第三节　结　构

在实际问题中，一组数据往往具有不同的数据类型。例如，在学生登记表中，姓名应为字符型；学号可为整型或字符型；年龄应为整型；性别应为字符型；成绩可为整型或实型。显然不能用一个数组来存放这一组数据，因为数组中各元素的类型和长度都必须一致，以便于编译系统处理。为了解决这个问题，C 语言中给出了另一种构造数据类型——“结构”。

“结构”是一种构造类型，它是由若干“成员”组成的。每一个成员可以是一个基本数据类型或者又是一个构造类型。结构既是一种“构造”而成的数据类型，那么在说明和使用之前必须先定义它，也就是构造它。如同在说明和调用函数之前要先定义函数一样。

一、结构的定义

定义一个结构的一般形式为：

```
struct 结构名
{
结构成员说明
};
```

结构成员由若干个成员组成，每个成员都是该结构的一个组成部分。对每个成员也必须作类型说明，结构成员说明的格式为：

类型说明符成员名；

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

成员名的命名应符合标识符的书写规定。例如：

```
struct stu
{
int num;
char name[20];
char sex;
float score;
};
```

在这个结构定义中，结构名为 stu，该结构由 4 个成员组成。第一个成员为 num，整型变量；第二个成员为 name，字符数组；第三个成员为 sex，字符变量；第四个成员为 score，实型变量。应注意在括号后的分号是不可少的。结构定义之后，即可进行变量说明。凡说明为结构 stu 的变量都由上述 4 个成员组成。由此可见，结构是一种复杂的数据类型，是数目固定、类型不同的若干有序变量的集合。

二、结构类型变量的说明

说明结构变量有以下 3 种方法，现以上面定义的 stu 为例来加以说明。

1. 先定义结构，再说明结构变量

例如：

```
struct stu
{
int num;
char name[20];
char sex;
float score;
};
struct stu boy1,boy2;
```

说明了两个变量 boy1 和 boy2 为 stu 结构类型。也可以用宏定义使一个符号常量来表示一个结构类型，例如：

```
#define STU struct stu
STU
{
int num;
char name[20];
char sex;
float score;
};
STU boy1,boy2;
```

2. 在定义结构类型的同时说明结构变量

例如：

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
struct stu
{
int num;
char name[20];
char sex;
float score;
}boy1,boy2;
```

3. 直接说明结构变量

例如：

```
struct
{
int num;
char name[20];
char sex;
float score;
}boy1,boy2;
```

第三种方法与第二种方法的区别在于第三种方法中省去了结构名，而直接给出结构变量。说明了 boy1,boy2 变量为 stu 类型后，即可向这两个变量中的各个成员赋值。在上述 stu 结构定义中，所有的成员都是基本数据类型或数组类型。成员也可以又是一个结构，即构成了嵌套的结构。例如：

```
struct date{
int month;
int day;
int year;
}
struct{
int num;
char name[20];
char sex;
struct date birthday;
float score;
}boy1,boy2;
```

首先定义一个结构 date，由 month(月)、day(日)、year(年)3 个成员组成。在定义并说明变量 boy1 和 boy2 时，其中的成员 birthday 被说明为 data 结构类型。成员名可与程序中其他变量同名，互不干扰。结构变量成员的表示方法在程序中使用结构变量时，往往不把它作为一个整体来使用。

三、结构变量的引用

在 C 语言中，除了允许具有相同类型的结构变量相互赋值以外，一般对结构变量的

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

使用，包括赋值、输入、输出、运算等都是通过结构变量的成员来实现的。

表示结构变量成员的一般形式是：

结构变量名．成员名

例如：

boy1. num，即第一个人的学号；boy2. sex，即第二个人的性别。

如果成员本身又是一个结构，则必须逐级找到最低级的成员才能使用。

例如：boy1. birthday. month，即第一个人出生的月份，成员可以在程序中单独使用，与普通变量完全相同。

四、结构变量的赋值

结构变量的赋值就是给各成员赋值。可用输入语句或赋值语句来完成。

例如：给结构变量赋值并输出其值，程序如下：

```
main()
{
struct stu
   {
   int num;
   char *name;
   char sex;
   float score;
   } boy1,boy2;
   boy1. num=102;
   boy1. name="wang ping";
   printf("input sex and score\n");
   scanf("%c %f",&boy1. sex,&boy1. score);
   boy2=boy1;
   printf("Number=%d\nName=%s\n",boy2. num,boy2. name);
   printf("Sex=%c\nScore=%f\n",boy2. sex,boy2. score);
   }
```

本程序中用赋值语句给 num 和 name 两个成员赋值，name 是一个字符串指针变量。用 scanf 函数动态地输入 sex 和 score 成员值，然后把 boy1 的所有成员的值整体赋予 boy2。最后分别输出 boy2 的各个成员值。本例表示了结构变量的赋值、输入和输出的方法。

五、结构数组

数组的元素也可以是结构类型的。因此可以构成结构型数组。结构数组的每一个元素都是具有相同结构类型的下标结构变量。在实际应用中，经常用结构数组来表示具有相同数据结构的一个群体。如一个班的学生档案，一个车间职工的工资表等。

结构数组的定义方法和结构变量相似，只需说明它为数组类型即可。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

例如：

```
struct stu
{
int num;
char *name;
char sex;
float score;
}boy[5];
```

定义了一个结构数组 boy，共有 5 个元素，boy[0]～boy[4]。每个数组元素都具有 struct stu 的结构形式。对外部结构数组或静态结构数组可以作初始化赋值，例如：

```
struct stu
{
int num;
char *name;
char sex;
float score;
}boy[5]={
{101,"Li ping","M",45},
{102,"Zhang ping","M",62.5},
{103,"He fang","F",92.5},
{104,"Cheng ling","F",87},
{105,"Wang ming","M",58};
}
```

当对全部元素作初始化赋值时，也可不给出数组长度。

六、结构指针变量

当一个指针变量用来指向一个结构变量时，称之为结构指针变量。结构指针变量中的值是所指向的结构变量的首地址。通过结构指针即可访问该结构变量。结构指针变量说明的一般形式为：

struct 结构名 *结构指针变量名

例如，在前面定义了 stu 这个结构，如要说明一个指向 stu 的指针变量 pstu，可写为：

```
struct stu *pstu;
```

当然也可在定义 stu 结构时同时说明 pstu。结构指针变量必须先赋值后才能使用。赋值是把结构变量的首地址赋予该指针变量，不能把结构名赋予该指针变量。如果 boy 是被说明为 stu 类型的结构变量，则 pstu=&boy 是正确的，而 pstu=&stu 是错误的。

结构名和结构变量是两个不同的概念，不能混淆。结构名只能表示一个结构形式，编译系统并不对它分配内存空间。只有当某变量被说明为这种类型的结构时，才对该变量分配存储空间。因此上面 &stu 这种写法是错误的，不可能去取一个结构名的首地址。有了结构指针变量，就能更方便地访问结构变量的各个成员，其访问的一般形式为：

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

(*结构指针变量).成员名

或

结构指针变量->成员名

例如：(*pstu).num 或 pstu->num

应该注意(*pstu)两侧的括号不可少，因为成员符“.”的优先级高于“*”。如去掉括号写作*pstu.num，则等效于*(pstu.num)，这样，意义就完全不对了。

下面通过例子来说明结构指针变量的具体说明和使用方法。

例如：

```
struct stu
{
int num;
char *name;
char sex;
float score;
} boy1={102,"Zhang ping",'M',78.5},*pstu;
main()
{
pstu=&boy1;
printf("Number=%d\nName=%s\n",boy1.num,boy1.name);
printf("Sex=%c\nScore=%f\n\n",boy1.sex,boy1.score);
printf("Number=%d\nName=%s\n",(*pstu).num,(*pstu).name);
printf("Sex=%c\nScore=%f\n\n",(*pstu).sex,(*pstu).score);
printf("Number=%d\nName=%s\n",pstu->num,pstu->name);
printf("Sex=%c\nScore=%f\n\n",pstu->sex,pstu->score);
}
```

本程序定义了一个结构 stu，定义了 stu 类型结构变量 boy1 并作了初始化赋值，还定义了一个指向 stu 类型结构的指针变量 pstu。在 main 函数中，pstu 被赋予 boy1 的地址，因此 pstu 指向 boy1。然后在 printf 语句内用三种形式输出 boy1 的各个成员值。从运行结果可以看出：“结构变量.成员名”、“(*结构指针变量).成员名”和“结构指针变量->成员名”这三种用于表示结构成员的形式是完全等效的。

第四节　共用体

无论任何数据，在使用前必须定义其数据类型，只有这样，在编译时，C 编译器才会根据其数据类型，在内存中指定相应长度的内存单元，供其使用。不同类型的数据占据各自拥有的内存空间，彼此互不“侵犯”。那么是否存在某种数据类型，使 C 编译器在编译时为其指定一块内存空间，并允许各种类型的数据共同使用呢？回答是肯定的。这种数据类型就是共用体或称联合(union)。

共用体是 C 语言的构造类型数据结构之一。它与数组、结构等一样，也是一种比较

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

复杂的构造数据类型。

共用体与结构类似，也可以包含多个不同数据类型的元素，但其变量所占有的内存空间不是各成员所需存储空间的总和，而是在任何时候，其变量至多只能存放该类型所包含的一个成员，即它所包含的各个成员只能分时共享同一存储空间。这是共用体与结构的区别所在。

一、共用体的定义

定义一个共用体类型的一般形式为：

```
union 共用体名 {
类型说明符 变量名
};
```

例如：

```
union perdata
{
int class;
char office[10];
};
```

定义了一个名为 perdata 的共用体，它含有两个成员，一个为整型，成员名为 class；另一个为字符数组，数组名为 office。共用体定义之后，即可进行共用体变量说明，被说明为 perdata 类型的变量，可以存放整型量 class 或存放字符数组 office。

二、共用体变量的说明

共用体变量的说明和结构变量的说明方式相同，也有 3 种形式，即先定义，再说明；定义同时说明和直接说明。以 perdata 类型为例，说明如下：

```
union perdata
{
int class;
char officae[10];
};
union perdata a,b;   //说明 a,b 为 perdata 类型
```

或者同时说明：

```
union perdata
{
int class;
char office[10];
}a,b;
```

或者直接说明：

```
union
{
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
int class;
char office[10];
}a,b
```

经说明后的 a,b 变量均为 perdata 类型。a,b 变量的长度应等于 perdata 的成员中最长的长度,即等于 office 数组的长度,共 10 个字节。a,b 变量如赋予整型值时,只使用了 2 个字节,而赋予字符数组时,可用 10 个字节。

第五节 枚 举

生活中的很多信息,在计算机中都适于用数值来表示,比如,从星期一到星期天,可以用数字来表示。在西方,人们认为星期天是一周的开始,按照这种说法,定星期天为 0,而星期一到六分别用 1 到 6 表示。

现在,有一行代码,它表达今天是周 3:

```
int today=3;
```

很多时候,可以认为这已经是比较直观的代码了,不过可能在 6 个月以后,看到这行代码,会在心里想:是说今天是周三呢,还是说今天是 3 号? 其实可以做到更直观,并且方法很多。

第一种是使用宏定义:

```
#define SUNDAY 0
#define MONDAY 1
#define TUESDAY 2
#define WEDNESDAY 3
#define THURSDAY 4
#define FRIDAY 5
#define SATURDAY 6
int today = WEDNESDAY;
```

第二种是使用常量定义:

```
const int SUNDAY=0;
const int MONDAY=1;
const int TUESDAY=2;
const int WEDNESDAY=3;
const int THURSDAY=4;
const int FRIDAY=5;
const int SATURDAY=6;
int today=WEDNESDAY;
```

第三种方法就是使用枚举。

一、枚举类型的定义

枚举类型的定义一般格式为:

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

enum 枚举类型名{枚举值1,枚举值2,……};

enum:是定义枚举类型的关键字。

枚举类型名:要自定义的新的数据类型的名字。

枚举值:可能的个值。

比如:

enum Week
{SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY};

这就定义了一个新的数据类型:Week。

Week数据类型来源于int类型(默认)。

Week类型的数据只能有7种取值,它们是:SUNDAY,MONDAY,TUESDAY……SATURDAY。

其中SUNDAY=0,MONDAY=1.....SATURDAY=6。也就是说,第1个枚举值代表0,第2个枚举值代表1,这样依次递增1。

不过,也可以在定义时,直接指定某个或某些枚举值的数值。比如,对于中国人,可能对于用0表示星期日不是很好接受,不如用7来表示星期天。这样需要的个值就是1,2,3,4,5,6,7。可以这样定义:

enum Week {MONDAY=1,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY,SUNDAY};

如果希望星期一仍然从1开始,枚举类型默认枚举值从0开始,所以直接指定MONDAY等于1,这样,TUESDAY就等于2,直接到SUNDAY等于7。

二、枚举变量的说明

如同结构和共用体一样,枚举变量也可用不同的方式说明。

例如:设有变量a,b,c被说明为上述的Week,可采用下述任一种方式:

```
enum Week
{
……
};
enum Week a,b,c;
```

或者为:

```
enum Week
{
……
}a,b,c;
```

或者为:

```
enum
{
……
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

}a,b,c;

三、枚举类型变量的赋值和使用

枚举类型在使用中有以下规定：

①枚举值是常量，不是变量。它一经定义后，就不可再改变，以下用法是错误的：

TUESDAY=10;

一个枚举值的数值不能改变。

②枚举元素本身由系统定义了一个表示序号的数值，例如，在以下程序中，在 Week 中，SUNDAY=0，MONDAY=1.....SATURDAY=6。

```
main(){
enum Week {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY};a,b,c;
a=SUNDAY;
b=MONDAY;
c=TUESDAY;
printf("%d,%d,%d",a,b,c);
}
```

③只能把枚举值赋予枚举变量，不能把元素的数值直接赋予枚举变量。如：a=SUNDAY;b=MONDAY;是正确的。而 a=0;b=1;是错误的。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第九章　单片机中断、定时器和串行口的C语言编程

在前面的章节中，已经向读者详细讲解了 Cx51 语言的语法和 Keil 51 的使用方法，但是，对于一个想编好单片机应用程序的程序员来说，仅仅了解这些显然是不够的，用单片机的 C 语言的开发人员所需要了解的知识远比在 PC 机上使用 ANSI C 的程序员要多得多。其中，熟悉硬件就是 Cx51 编写者必备的素质之一。本章将首先介绍单片机的中断、定时器和串行口的基础知识，然后，结合大量实例和实验，分析单片机中断、定时器和串行口的 C 语言编程的方法和技巧。

第一节　单片机中断系统C语言编程

一、什么是中断

所谓中断，是指在计算机执行程序过程中，当出现某种情况，如发生停电和其他情况时，由服务对象向 CPU 发出中断请求信号，要求 CPU 暂时中断当前程序的执行，而转去执行相应的处理程序，待处理程序执行完毕后，再返回来继续执行原来被中断的程序。

在中断系统中，通常将 CPU 正常情况下运行的程序称为主程序，把引起中断的设备或事件称为中断源。由中断源向 CPU 所发出的请求中断的信号称为中断请求信号，CPU 接受中断申请终止现行程序而转去为服务对象服务称中断响应，为服务对象服务的程序称为中断服务程序(也称中断处理程序)。现行程序中断的地方称为断点，为中断服务对象服务完毕后返回原来的程序称为中断返回。整个过程称为中断。

二、中断源

向 CPU 发出中断请求的来源称之为中断源。MCS-5l 有 3 类共 5 个中断源，即 2 个外中断$\overline{\text{INT0}}$和$\overline{\text{INT1}}$(由 P3.2 和 P3.2 引入)、2 个定时中断(定时器 T0、定时器 T1)和 1 个串行中断。其中，定时中断和串行中断属于内中断。每个中断源对应一个中断标志位，当某个中断源有中断请求时，相应的中断标志位置 1，外中断和定时中断源的中断标志位在 TCON 中，串行中断源的中断标志位在 SCON 中。

1. 外中断

外中断是由外部信号引起的，共有两个中断源，即外部中断“0”和外部中断“1”。它们的中断请求信号分别由引脚$\overline{\text{INT0}}$(P3.2)和$\overline{\text{INT1}}$(P3.3)引入。

外部中断请求有两种信号方式，即电平方式和脉冲方式。可通过设置有关控制位进行定义。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

电平方式的中断请求是低电平有效。只要单片机在中断请求引入端($\overline{INT0}$或$\overline{INT1}$)上采样到有效的低电平时,就激活外部中断。

而脉冲方式的中断请求则是脉冲的后沿负跳有效。这种方式下,CPU 在两个相邻机器周期对中断请求引入端进行的采样中,如前一次为高电平,后一次为低电平,即为有效中断请求。因此在这种中断请求信号方式下,中断请求信号的高电平状态和低电平状态都要至少维持一个机器周期,以确保电平变化能被单片机采样到。

2. 定时中断

定时中断是为满足定时或计数的需要而设置的。在单片机内部,有两个定时器/计数器,通过对其中的计数结构进行计数,来实现定时或计数功能。当计数结构发生计数溢出时,即表明定时时间到或计数值已满,这时就以计数溢出信号作为中断请求,去置位一个溢出标志位,作为单片机接受中断请求的标志。由于这种中断请求是在单片机芯片内部发生的,因此无需在芯片上设置引入端。

3. 串行中断

串行中断是为串行数据传送的需要而设置的。每当串行接收或发送完一组串行数据时,就产生一个中断请求。因为串行中断请求也是在单片机芯片内部自动发生的,所以同样不需在芯片上设置引入端。

三、中断控制

在 80C51 单片机中,有 4 个寄存器是供用户对中断进行控制的,这 4 个寄存器分别是定时器控制寄存器 TCON、串行口控制寄存器 SCON、中断允许控制寄存器 IE 以及中断优先控制寄存器 IP。这 4 个控制寄存器都属于专用寄存器,可完成中断请求标志寄存、中断允许管理和中断优先级的设定,由它们所构成的中断系统如图 9-1 所示。

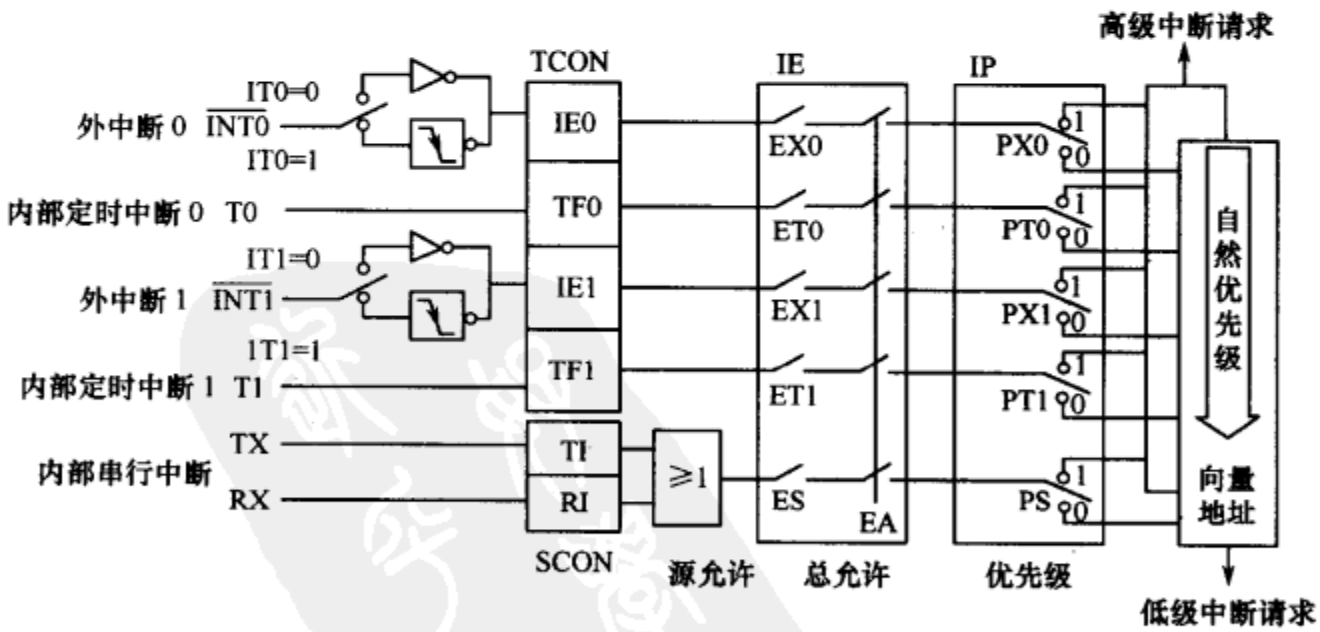


图 9-1　中断系统的结构

1. 定时器控制寄存器 TCON

定时器控制寄存器 TCON 用于保存外部中断请求以及定时器的计数溢出。寄存器地址 88H,位地址 8FH～88H。寄存器的内容及位地址表示如下:

位地址	8FH	8EH	8DH	8CH	8BH	8AH	89H	88H
位名称	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

TCON 寄存器既有定时器/计数器的控制功能，又有中断控制功能，其中；与中断有关的控制位共 6 位，对控制寄存器中的每一位要从含义、功能和置位/清除的方式（硬件方式还是软件方式）等方面来进行理解。

（1）IE0（IE1）——外部中断请求标志位

当 CPU 采样到$\overline{\text{INT0}}$（或$\overline{\text{INT1}}$）端出现的有效中断请求信号时，此位由硬件置 1，在中断响应完成后转向中断服务子程序时，再由硬件自动清 0。

（2）IT0（IT1）——外部中断触发方式控制位

IT0（IT1）＝1，脉冲触发方式，下降沿触发有效。

IT0（IT1）＝0，电平触发方式，低电平有效。

此位由软件置位或清除。

（3）TF0（TF1）——内部定时/计数器 0（定时/计数器 1）溢出标志位

当片内定时/计数器 0（定时/计数器 1）产生计数溢出时，TF0（TF1）由硬件置 1。当转向中断服务时，再由硬件自动清 0。

此外还有两位——TR0 和 TR1，在后面介绍定时/计数器时再作介绍。

2. 串行口控制寄存器 SCON

串行口控制寄存器 SCON 地址 98H，位地址 9FH～98H，具体格式如下：

位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
位名称	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

与中断有关的控制位共两位，即 TI 和 RI。

TI 是串行口发送中断请求标志位，当发送完一帧中行数据后，由硬件置 1；表示串行口发送器正在向 CPU 申请中断，CPU 响应发送器中断请求，转向执行中断服务程序时，不会自动清 0，必须由用户在中断服务程序中用 CLR TI 等指令清 0。

RI 是串行口接收中断请求标志位。当接收完一帧串行数据后，由硬件置 1；在转向中断服务程序后，用软件清 0。

TI 和 RI 由逻辑或得到，也就是说，无论是发送标志还是接收标志，都产生串行中断请求。

3. 中断允许控制寄存器 IE

计算机中断系统中有两种不同类型的中断：一类称为非屏蔽中断，另一类称为可屏蔽中断。所谓非屏蔽中断，是指用户不能用软件方法加以禁止，一旦有中断申请，CPU 必须予以响应。对于可屏蔽中断，用户则可以通过软件方法来控制是否允许某中断源的中断。

从图 9-1 中可以看出，80C51 单片机的 5 个中断源都是可屏蔽中断，CPU 对中断源的中断开放（允许）或中断屏蔽（禁止）是通过中断允许寄存器 IE 设置的。IE 既可按字节地址寻址，其字节地址为 A8H；又可按位寻址，位地址 AFH～A8H，具体格式如下：

位地址	0AFH	0AEH	0ADH	0ACH	0ABH	0AAH	0A9H	0A8H
位名称	EA	—	—	ES	ET1	EX1	ET0	EX0

（1）EA——中断允许总控制位

EA＝0，中断总禁止，关闭所有中断，由软件设置。

EA＝1，中断总允许，总允许后，各中断的禁止或允许由各中断源的中断允许控制位

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

进行设置。

(2)EX0(EX1)——外部中断允许控制位

EX0(EX1)=0,禁止外中断 0(外中断 1)。

EX0(EX1)=1,允许外中断 0(外中断 1)。

(3)ET0(ET1)——定时中断允许控制位

ET0(ET1)=0,禁止定时中断 0(定时中断 1)。

ET0(ET1)=1,允许定时中断 0(定时中断 1)。

(4)ES——串行中断允许控制位

ES=0,禁止串行中断。

ES=1,允许串行中断。

可见,80C51 单片机通过中断允许控制寄存器对中断的允许实行两级控制。以 EA 位作为总控制位,以各中断源的中断允许位作为分控制位。当总控制位为禁止时,不管分控制位状态如何,整个中断系统为禁止状态;当总控制位为允许时,才能由各中断源的分控制位设置各自的中断允许与禁止。单片机复位后,(IE)=00H,因此,整个系统处于禁止状态。

需要说明的是,单片机在中断响应后不会自动关闭中断。因此,在转向中断服务程序后,应使用有关指令禁止中断,即用软件方式关闭中断。

4. 中断优先级控制寄存器 IP

80C51 的中断优先级控制比较简单,只有高低两个优先级。当多个中断源同时申请中断时,CPU 首先响应优先级最高的中断请求,在优先级最高中断处理完之后,再响应级别较低的中断。

80C51 单片机各中断源的优先级由优先级控制寄存器 IP 进行设定(软件设置)。

IP 寄存器地址 B8H,位地址 BFH~B8H,具体格式如下:

位地址	0BFH	0BEH	0BDH	0BCH	0BBH	0BAH	0B9H	0B8H
位名称	—	—	—	PS	PT1	PX1	PT0	PX0

PX0(PX1)是外中断 0(外中断 1)优先级设定位;

PT0(PT1)是定时中断 0(定时中断 1)优先级设定位;

PS 是串行中断优先级设定位。

各位为 0 时,为低优先级;各位为 1 时,为高优先级。

80C51 中断优先级的控制原则是:

①低优先级中断请求不能打断高优先级的中断服务;反之,则可以,从而实现中断嵌套。

②如果一个中断请求已被响应,则同级的其他中断响应被禁止。

③如果同级的多个中断请求同时出现,则按 CPU 查询次序确定哪个中断请求被响应。从高到低依次为:外部中断 0→定时中断 0→外部中断 1→定时中断 1→串行中断。

例如,某单片机应用系统需使用 3 个中断:外部中断 0、定时中断 0 和串行口中断,若要求它们的优先级次序为定时中断 0 优先级最高,外部中断 0 次之,串行口中断最低,则只要把中断优先级控制寄存器的 PT0 位置 1,就可以实现该系统对优先级次序的要求。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

方法技巧 上面所讲的4个寄存器都是为用户需要而设置的，因此在采用中断方式时，要在程序初始化时进行设置，外中断初始化主要有：中断总允许、外中断允许、中断方式和中断优先级设定。定时中断则没有中断方式控制。

例如，假定要开放外中断1，采用脉冲触发方式，则需要做如下工作：

设置中断允许位：　　EA=1(中断总允许)

　　EX1=1(外中断1允许)

设置中断请求信号方式：　　IT1=1(脉冲触发方式)

设置优先级：　　PX1=1(外中断1优先级最高)

四、中断的响应

中断响应就是单片机CPU对中断源提出的中断请求的接受。中断请求被响应后，再经过一系列的操作，而后转向中断服务程序，完成中断所要求的处理任务。下面简要说明80C51的中断响应过程。

1. 外中断采样和内中断置位

(1)外中断采样

要想知道外中断是否有请求发生，需要对外中断进行采样。

当通过软件将寄存器TCON的IT0(或IT1)位设置为0时，$\overline{\text{INT0}}$(或$\overline{\text{INT1}}$)为电平触发方式，CPU在每个机器周期的S5P2(第5状态第2拍节)期间对$\overline{\text{INT0}}$(或$\overline{\text{INT1}}$)采样，一旦在P3.2(或P3.3)上检测到低电平时，则认为有外中断申请，随即由硬件使TCON的IE0(或IE1)位置1，向CPU申请中断。在中断响应完成后转向中断服务子程序时，再由硬件自动对IE0(或IE1)位清0。

当寄存器TCON的IT0(或IT1)位为1时，$\overline{\text{INT0}}$(或$\overline{\text{INT1}}$)为脉冲触发方式，则CPU在每个机器的S5P2期间对$\overline{\text{INT0}}$(或$\overline{\text{INT1}}$)采样，当检测到前一周期为高电平，后一周期为低电平时，由硬件使TCON的IE0(IE1)位置1，向CPU申请中断，在中断响应完成后转向中断服务子程序时，再由硬件自动对IE0(IE1)位清0。在边沿触发方式中，为保证CPU在两个机器周期内检测到由高到低的负跳变，高电平与低电平的持续时间不得少于一个机器周期的时间。

(2)内中断置位

80C51把所有中断标志都集中到TCON和SCON寄存器中。其中外中断是使用采样的方法把中断请求锁定在TCON寄存器的IE0(IE1)标志位上，而定时中断和串行中断的中断请求由于都发生在芯片的内部，定时中断可以直接去置位TCON的TF0(TF1)，串行中断可以直接去置位SCON的RI和TI。内中断不存在采样问题。

2. 中断查询

所谓查询，就是由CPU测试TCON和SCON中各标志位的状态，以确定有没有中断请求发生以及是哪一个中断请求。80C51单片机是在每一个机器周期的最后一个状态(S6)，按优先级顺序对中断请求标志位进行查询，即先查询高级中断后再查询低级中断，同级中断按“外部中断0→定时中断0→外部中断1→定时中断1→串行中断”的顺序查询。如果查询到有标志位为“1”，则表明有中断请求发生，接着就从相邻的下一个机器周期的S6状态开始进行中断响应。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

由于中断请求是随机发生的，CPU 无法预先得知，因此在程序执行过程中，中断查询要在指令执行的每个机器周期中不停地重复进行。如果换成人来说，就相当于你在看书的时候，每一秒钟都会抬起头来听一听，看一看，是不是有人按门铃，是否有电话，烧的水是否开了……由此看来，单片机比人蠢多了。

3. 中断响应

中断响应就是对中断源提出的中断请求的接受，是在中断查询之后进行的，当查询到有效的中断请求时，紧接着就进行中断响应。中断响应时，根据寄存器 TCON、SCON 中的中断标记，由硬件自动生成一条长调用指令，转向与该中断对应的服务程序入口地址。对于 80C51 的 5 个独立中断源，这些入口地址已由系统设定。80C51 中 5 个独立中断源所对应的入口地址如表 9-1 所列。

表 9-1　入口地址表

中断号 n	中断源	中断向量	中断号 n	中断源	中断向量
0	外部中断 0	0003H	3	定时器 1	001BH
1	定时器 0	000BH	4	串行口	0023H
2	外部中断 1	0013H			

它们的自然优先级由高到低排列。

Cx51 编译器支持在 C 源程序中直接开发中断程序，因此减轻了用汇编语言开发中断程序的烦琐过程。

使用该扩展属性的函数定义语法如下：

函数类型　函数名(形式参数表)[interrupt n][using n]

interrupt　后面的 n 对应中断源的编号。

中断服务程序的不同分支取决于中断源。在重新允许全局 CPU 中断(EA)之前，必须仔细清除各种标志。标志会引起立即的重复中断。8051 对中断实际上有特殊的返回指令——RETI，不是 RET。RETI 重新允许系统识别其他中断。因而，没必要在正常使用中断时复位 EA，只要在程序初始化时开中断一次即可。

在处理一个中断时，不能识别同级或更低优先级的其他中断。当高优先级或低优先级(优先级 0)程序运行时，其他低优先级中断被屏蔽；当高优先级程序运行时，其他高优先级(优先级 1)中断服务程序被屏蔽；仅当 RETI 执行或 EA 设置为 1 时，中断硬件才对其他中断识别。

一旦设置了中断请求标志，则标志保留，不管标志是由硬件还是软件设置。可以通过设置 TCON 中的 TF0、TF1、IE0 或 IE1，SCON 中的 RI 或 TI，用软件引发中断程序。

同样，在向量中断硬件识别之前，可以用软件清除中断请求标志(通常它们被屏蔽，或者是更高或同优先级的中断正在执行中断服务)。注意外部电平触发的中断不锁存。若在电平出现时被屏蔽，而在中断识别之前电平消失，则它被完全忽略，因为中断处理本身不能锁存外部电平请求。若有重要的电平触发外部中断，就应该仔细考虑中断服务程序。

80C51 系列单片机可以在片内 RAM 中使用 4 个不同的工作寄存器组，每个寄存器组中包含 8 个工作寄存器(R0～R7)。Keil Cx51 编译器扩展了一个关键字“using”，专门用来选择 8051 单片机中不同的工作寄存器组。using 后面的 n 是一个 0～3 的常整数，分

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

别选中 4 个不同的工作寄存器组。在定义一个函数时 using 是一个选项，如果不用该选项，则由编译器自动选择一个寄存器组作绝对寄存器组访问。需要注意的是，关键字 using和 interrupt 的后面都不允许跟带运算符的表达式。

关键字 using 对函数目标代码的影响如下：在函数的入口处将当前工作寄存器组保护到堆栈中；指定的工作寄存器内容不会改变；函数退出之前将被保护的工作寄存器组从堆栈中恢复。

使用关键字 using 在函数中确定一个工作寄存器组时必须十分小心，要保证任何寄存器组的切换都只在仔细控制的区域内发生，如果不做到这一点，将产生不正确的函数结果。

另外还要注意，带 using 属性的函数原则上不能返回 bit 类型的值，并且关键字 using 不允许用于外部函数。

关键字 interrupt 也不允许用于外部函数，它对中断函数目标代码的影响如下：在进入中断函数时，特殊功能寄存器 ACC、B、DPH、DPL、PSW 将被保存入栈；如果不使用关键字 using 进行工作寄存器组切换，则将中断函数中所用到的全部工作寄存器都入栈保存；函数退出之前所有的寄存器内容出栈恢复，中断函数由 80C51 单片机指令 RETI 结束。

五、中断的撤除

中断响应后，TCON 或 SCON 中的中断请求标志应及时清除。否则就意味着中断请求仍然存在，有可能造成中断的重复查询和响应，因此就存在一个中断请求的撤除问题。

1. 定时中断请求的撤除

定时中断响应后，硬件自动把标志位 TF0(或 TF1)清 0，因此定时中断的中断请求是自动撤除的，不需要用户干预。

2. 串行中断软件撤除

对于串行中断，CPU 响应中断后，没有用硬件清除它们的中断标志 RI、TI，必须在中断服务程序中用软件清除，以撤除其中断请求。

3. 外中断请求的撤除

外部中断的撤除包括中断标志位 IE0(或 IE1)的清 0 和外中断请求信号的撤除。其中，IE0(或 IE1)清 0 是在中断响应后由硬件电路自动完成的。剩下的只是外中断引脚请求信号的撤除。下面对脉冲和电平两种触发方式分别进行讨论。

(1)脉冲方式外部中断请求的撤除

对于脉冲方式的中断请求，由于脉冲信号过后就消失了，也可以说中断请求信号是自动撤除的。

(2)电平方式外部中断请求的撤除

对于电平方式的外部中断，中断标志的撤除是自动的，但中断请求信号的低电平可能继续存在，在以后机器周期采样时，又会把已清 0 的 IE0 或 IE1 标志位重新置 1。为此，要彻底解决电平方式外中断的撤除，除了标志位清 0 之外，必要时还需在中断响应后把中断请求信号引脚从低电平强制改变为高电平，为此，可在系统中增加如图 9-2 所示的电路(以外中断 0 为例)。

从图中可以看出，外部中断 0 请求信号加在 D 触发器(可选用 74LS74)的时钟输入

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

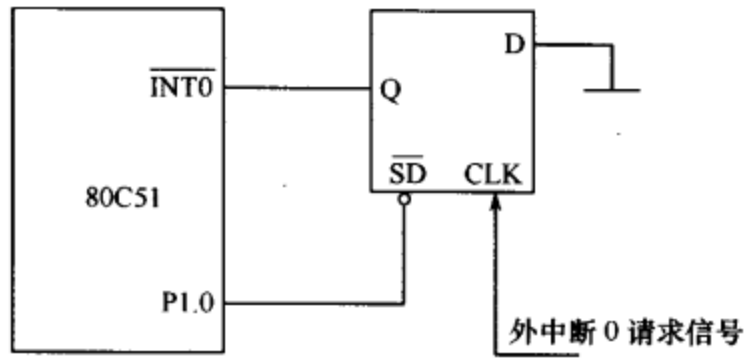


图 9-2　外中断请求标志清除电路

端。当外部设备有中断请求信号(为低电平)出现时,Q 端输出为低电平,$\overline{INT0}$有效,向 CPU 发出中断请求信号。CPU 响应中断后,在中断服务程序中由软件安排 1 个低电平中断应答信号,从 P1.0 送至 D 触发器的$\overline{SD}$(置位端,低电平有效),使 D 触发器的 Q 端输出为高电平,从而撤除了低电平的外中断 0 请求信号。$\overline{SD}$端所需的低电平可通过在中断服务程序中增加以下指令得到:

P1=P1&0xfe; 使 P1.0 输出为低电平,D 触发器置位

在中断服务程序中还要加上清除外中断 0 标志指令,如下所示:

IE0=0; 清外中断标志,以便下次可再次中断

可见,电平方式外部中断请求信号的撤除是通过软硬件相结合的方法实现的。

六、实验演练

为实现中断而设计的有关程序称为中断程序。中断程序由中断控制程序和中断服务程序两部分组成。

中断控制程序也称中断初始化程序。要使 CPU 在执行主程序的过程中能够响应中断,就必须先对中断系统进行初始化。80C51 单片机中断系统初始化包括:选择中断触发方式(对外部中断而言)、开中断、设置中断优先级等。

系统复位后,寄存器 TCON、SCON、IE、IP 等均复位为 00H,需要根据中断控制的要求,在中断控制程序中对这些寄存器进行设置。此外,定时器/计数器的启动、初始值的预置、工作方式的设定等往往也是在中断控制程序中完成的。

从程序所处位置看,中断控制程序在主程序中,作为主程序的一部分并和主程序一起运行,中断服务程序则存放在主程序之外的其他存储区,只是在主程序运行过程中发生中断时,CPU 才暂停主程序执行,转而去执行中断服务程序。中断服务完毕之后,还得再转回来继续执行主程序。

实验 1:在下载型实验板上,P3.2($\overline{INT0}$)上接有一只开关 K1,用 K1 按钮来模拟外部中断 0 的输入信号,并用 P1 口外接的 LED 作为中断响应。要求将外中断 0 触发方式设置为下降沿触发。程序启动时,P1 口上的 8 只 LED 亮,按一次按键 K1 引发一次外中断 0,在外部中断 0 服务程序中,让左右 4 个 LED 交替闪烁。

实验源程序如下:

```
#include<reg51.h>
//以下是延时函数
void Delay(unsigned int i)          //延时程序,i 是形式参数
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
{
    unsigned int j;
    for(;i>0;i--)                    //变量 i 由实际参数传入一个值,因此 i 不能赋
                                     初值
    for(j=0;j<125;j++)
    {;}
}
//以下是主程序
void main()
{
    EA=1;                            //开放总中断
    EX0=1;                           //允许外部中断 0 中断
    IT0=1;                           //设置外部中断为边沿中断方式
    while(1)                         //等待中断
    {P1=0;}                          //P1 口 LED 全亮

}
//以下是外部中断 0 中断服务子程序
void int0_int(void) interrupt 0
{
    EX0=0;                           //禁止外部中断 0 再次中断
    P1=0x0f;
    Delay(1000);
    P1=0xf0;
    Delay(1000);
    EX0=1;                           //允许外部中断 0 中断
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 int0.c。对程序进行编译、链接和调试,产生 int0.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target ',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_9/int0 文件夹中。

实验 2:用下载型实验板做如下实验:不按按键时,P1 口的 LED 呈流水灯显示,按下 K1(P3.2),P1 口的左右 4 个 LED 交替闪烁,按下 K2(P3.3),P1 口的 LED 闪亮,外中断

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

0 与外中断 1 优先级相同。

实验源程序如下：

```
#include<reg51.h>
//以下是延时函数
void Delay(unsigned int i)          //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)                   //变量 i 由实际参数传入一个值,因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
}
//以下是主程序
void main()
{
  unsigned char design[9]={0xFF,0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,
0x7F};                              //定义花样数据
    unsigned char a;                //尽量使用少字节的类型
    for(;;)
    {
        for (a=0; a<9; a++)
        {
          Delay(500);               //延时
          P1 = design[a];           //读已定义的花样数据并写花样数据到 P1 口
        }
    EA=1;                           //开放总中断
    EX0=1;                          //允许外部中断 0 中断
    EX1=1;                          //允许外部中断 1 中断
    IT0=1;                          //设置外部中断 0 为边沿中断方式
    IT1=1;                          //设置外部中断 1 为边沿中断方式
    IP=0;                           //设定中断优先级
  }
}
//以下是外部中断 0 中断服务子程序
void int0_int(void) interrupt 0
{
    for(;;)                         //无穷循环
    {P1=0x0f;                       //左 4 个 LED 亮
    Delay(500);                     //延时
    P1=0xf0;                        //右 4 个 LED 亮
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    Delay(500);}                    //延时
}
//以下是外部中断 1 中断服务子程序
void int1_int(void) interrupt 2
{
    for(;;)                         //无穷循环
    {P1=0xff;                       //全灭
    Delay(500);                     //延时
    P1=0;                           //全亮
    Delay(500);}                    //延时
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 int. c。对程序进行编译、链接和调试，产生 int. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_9/int 文件夹中。

第二节　定时/计数器及实验

一、定时/计数器概述

1. 什么是计数

所谓计数是指对外部事件进行计数。外部事件的发生以输入脉冲表示，因此计数功能的实质就是对外来脉冲进行计数。80C51 单片机有 T0(P3. 4)和 T1(P3. 5)两个信号引脚，分别是这两个计数器的计数输入端。外部输入的脉冲在负跳变时有效，进行计数器加 1(加法计数)。

计数方式下，单片机在每个机器周期的 S5P2 拍节对外部计数脉冲进行采样。如果前一个机器周期采样为高电平，后一个机器周期采样为低电平，即为一个有效的计数脉冲。在下一机器周期的 S3P1 进行计数。可见采样计数脉冲是在两个机器周期进行的。鉴于此，计数脉冲的频率不能高于振荡脉冲频率的 1/24。

2. 什么是定时

定时是通过计数器的计数来实现的，不过此时的计数脉冲来自单片机的内部，即每个机器周期产生一个计数脉冲。也就是每个机器周期计数器加 1。定时和计数的脉冲来源如图 9-3 所示。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

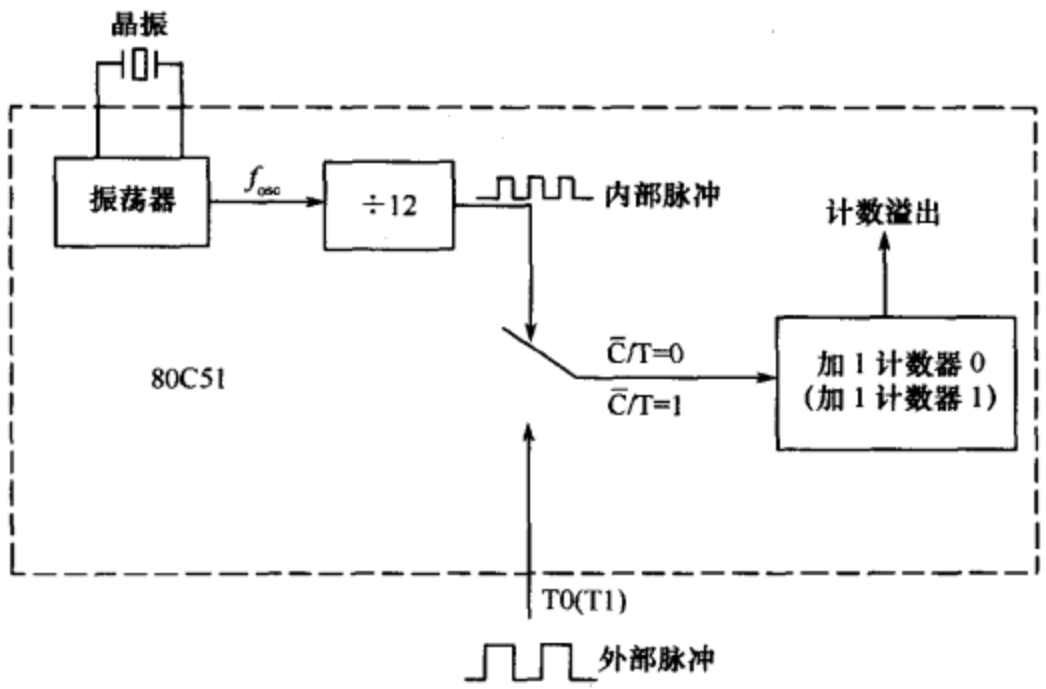


图 9-3　定时和计数脉冲的来源

由于一个机器周期等于 12 个振荡脉冲周期，因此计数频率为振荡频率的 1/12。如果单片机采用 12 MHz 晶体，则计数频率为 1 MHz。即每微秒计数器加 1。这样不但可以根据计数值计算出定时时间，也可以反过来按定时时间的要求计算出计数器的预置值。

重点提示　在前面章节中曾介绍过延迟程序，它是通过循环程序来进行延时的，特点是不需外加硬件电路，但定时要占用 CPU，增加 CPU 开销，因此，定时的时间不宜太长。下面介绍的利用 80C51 内部定时/计数器实现的定时控制，不但可实现任意定时和计数，而且 CPU 不必通过等待来实现延时，因此，可以提高 CPU 的效率。

3. 定时/计数器的组成

图 9-4 是 80C51 单片机内部定时/计数器结构框图。

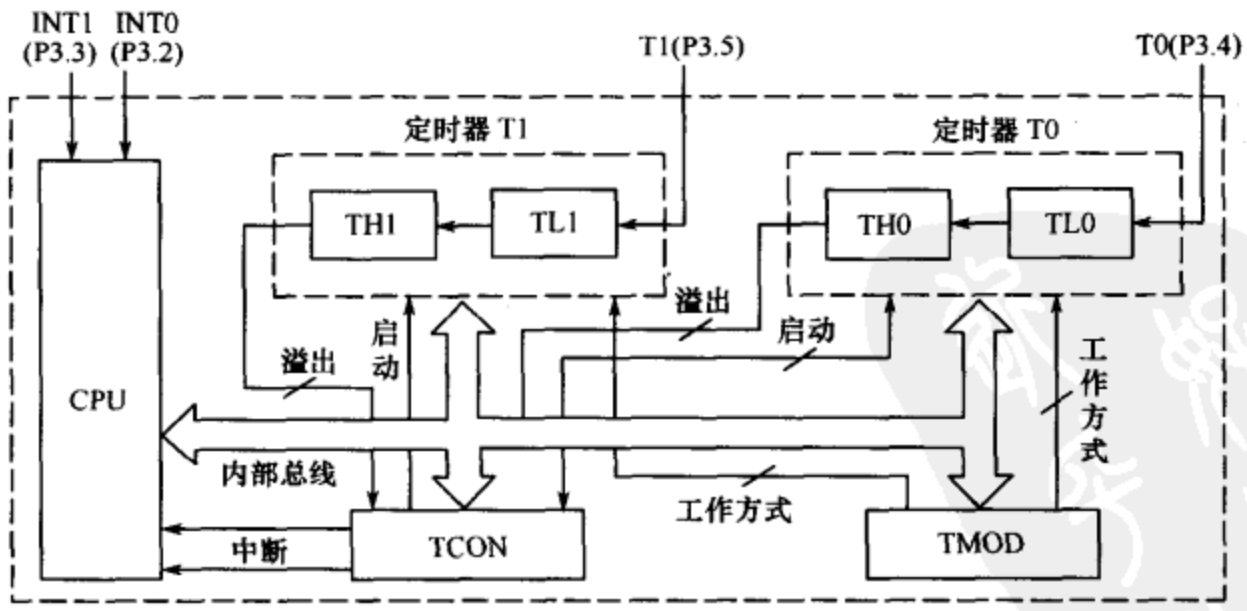


图 9-4　定时/计数器的结构框图

从图中可以看出，定时/计数器主要由几个特殊功能寄存器 TH0、TL0、TH1、TL1 以及 TMOD、TCON 组成。TH0(高 8 位)、TL0(低 8 位)构成 16 位计数器，用来存放定时器 T0 的计数初值，TH1(高 8 位)、TL1(低 8 位)构成 16 位计数器，用来存放定时器 T1 的计数初值，这两个 16 位计数器都是 16 位的加 1 计数器。TMOD 用来控制两个定时/

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

计数器的工作方式，TCON 用作中断溢出标志并控制定时器的启停。

两个定时/计数器都可由软件设置为定时或计数的工作方式，其中 T1 还可作为串行口的波特率发生器。不论 T0 或 T1 是工作于定时方式还是计数方式，它们在对内部时钟或外部事件进行计数时，都不占用 CPU 时间，直到定时/计数器产生溢出。如果满足条件，CPU 才会停下当前的操作，去处理“时间到”或者“计数溢出”这样的事件。因此，定时/计数器是与 CPU“并行”工作的，不会影响 CPU 的其他工作。

二、定时/计数器的控制寄存器

与两个定时/计数器 T0 和 T1 有关的控制寄存器有 TMOD、TCON 和 IE，它们主要用来设置各个定时/计数器的工作方式、选择定时或计数功能、控制启动运行以及作为运行状态的标志等。

1. 工作方式控制寄存器 TMOD

TMOD 寄存器是一个特殊功能寄存器，字节地址为 89H，不能位寻址。各位定义如下：

位号	D7	D6	D5	D4	D3	D2	D1	D0
符号	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0

TMOD 的低半字节用来定义定时/计数器 0，高半字节定义定时/计数器 1。复位时 TMOD 为 00H。

(1)M1、M0——工作方式选择位

M1、M0 用来选择工作方式，对应关系如表 9-2 所列。

表 9-2　定时/计数器的方式选择

M1　M0	工 作 方 式	功　　能
0　0	工作方式 0	13 位计数器
0　1	工作方式 1	16 位计数器
1　0	工作方式 2	自动再装入 8 位计数器
1　1	工作方式 3	定时器 0：分成两个 8 位计数器
		定时器 1：停止计数

(2)C/$\overline{T}$——定时/计数功能选择位

C/$\overline{T}$=0 为定时方式。在定时方式中，以振荡输出时钟脉冲的 12 分频信号作为计数信号，如果单片机采用 12 MHz 晶体，则计数频率为 1 MHz，计数脉冲周期为 1μs，即每微秒计数器加 1。

C/$\overline{T}$=1 为计数方式。在计数方式中，单片机在每个机器周期的 S5P2 拍节对外部计数脉冲进行采样。如果前一个机器周期采样为高电平，后一个机器周期采样为低电平，即为一个有效的计数脉冲。在下一机器周期的 S3P1 进行计数。

(3)GATE——门控位

GATE=1，定时/计数器的运行受外部引脚输入电平的控制，即$\overline{INT0}$控制 T0 运行，$\overline{INT1}$控制 T1 运行。

GATE=0，定时/计数器的运行不受外部输入引脚的控制。

2. 定时器控制寄存器 TCON

TCON 寄存器既参与中断控制又参与定时控制。前面已作了简要介绍，下面对与定

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

时控制有关的功能加以说明。

(1)TF0 和 TF1——计数溢出标志位

当计数器计数溢出(计满)时,该位置 1;使用查询方式时,此位作状态位供查询,但应注意查询有效后应用软件方法及时将该位清 0;使用中断方式时,此位作中断标志位,在转向中断服务程序时由硬件自动清 0。

(2)TR0 和 TR1——定时器运行控制位

TR0(TR1)=0,停止定时/计数器工作。

TR0(TR1)=1,启动定时/计数器工作。

该位根据需要靠软件来置 1 或清 0,以控制定时器的启动或停止。

3. 中断允许控制寄存器 IE

IE 寄存器在前面已作了简要介绍,下面对与定时/计数器有关的位介绍如下。

ET0 和 ET1——定时/计数中断允许控制位:

ET0(ET1)=0,禁止定时/计数中断。

ET0(ET1)=1,允许定时/计数中断。

三、定时/计数器的工作方式

80C51 单片机的定时/计数器共有 4 种工作方式,由寄存器 TMOD 的 M1M0 位进行控制,现以定时/计数器 0 为例进行介绍,定时/计数器 1 与定时/计数器 0 完全相同。

1. 工作方式 0

(1)逻辑电路结构

工作方式 0 是 13 位计数结构的工作方式,其计数器由 TH0 全部 8 位和 TL0 的低 5 位构成,TL 的高 3 位未用,图 9-5 为工作方式 0 的逻辑电路结构图。

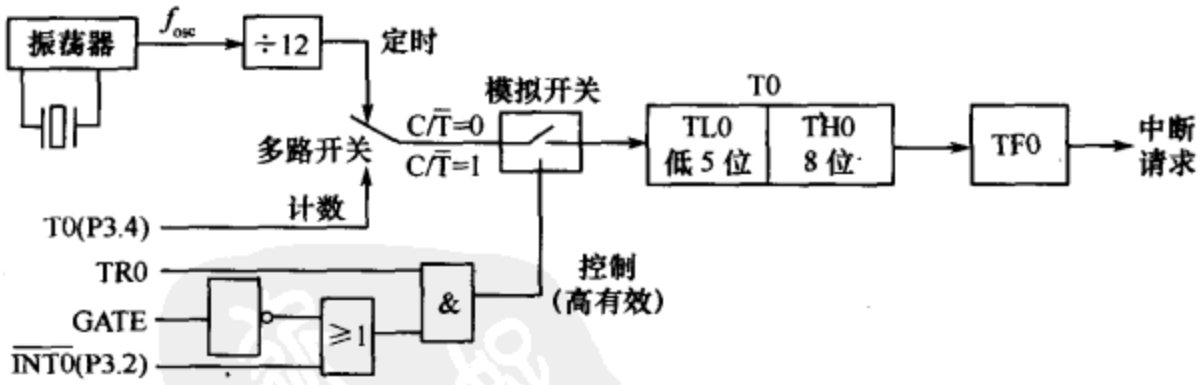


图 9-5　工作方式 0 的逻辑电路结构图

当 C/$\overline{T}$=0 时,多路开关接通振荡脉冲的 12 分频输出,13 位计数器以此进行计数,这就是定时方式。

当 C/$\overline{T}$=1 时,多路开关接通计数引脚 P3.4(T0),外部计数脉冲由引脚 P3.4 输入。当计数脉冲发生负跳变时,计数器加 1,这就是计数方式。

不管是定时方式还是计数方式,当 TL0 的低 5 位计数溢出时,向 TH0 进位,而全部 13 位计数溢出时,则向计数溢出标志位 TF0 进位。在满足中断条件时,向 CPU 申请中断,若需继续进行定时或计数,则应用指令对 TL0、TH0 重新置数,否则,下一次计数将会从 0 开始,造成计数或定时时间不准确。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

这里要特别说明的是，T0 能否启动，取决于 TR0、GATE 和引脚$\overline{\text{INT0}}$的状态。

当 GATE＝0 时，GATE 信号封锁了或门，使引脚$\overline{\text{INT0}}$信号无效。而或门输出端的高电平状态却打开了与门。这时如果 TR0＝1，则与门输出为 1，模拟开关接通，定时/计数器 0 工作。如果 TR0＝0，则断开模拟开关，定时/计数器 0 不能工作。

当 GATE＝1，同时 TR0＝1 时，模拟开关是否接通由$\overline{\text{INT0}}$控制。当$\overline{\text{INT0}}$＝1 时，与门输出高电平，模拟开关接通，定时/计数器 0 工作；当$\overline{\text{INT0}}$＝0 时，与门输出低电平，模拟开关断开，定时/计数器 0 停止工作。这种情况可用于测量外信号的脉冲宽度。

(2)计数初值的计算

工作方式 0 是 13 位计数结构，其最大计数为 $2^{13}=8192$，也就是说，每次计数到 8192 都会产生溢出，去置位 TF0。但在实际应用中，经常会有少于 8192 个计数值的要求，例如，要求计数到 1000 就产生溢出，这时可在计数时，不从 0 开始，而是从一个固定值开始，这个固定值的大小，取决于被计数的大小。如要计数 1000，预先在计数器里放进 7192，再来 1000 个脉冲，就到了 8192，这个 7192 计数初值，也称作预置值。

定时也有同样的问题，并且也可采用同样的方法来解决。假设单片机的晶振是 12 MHz，那么每个计时脉冲是 1μs，计满 8192 个脉冲需要 8.192ms，如果只需定时 1ms，可以作这样的处理：1ms 即 1000μs，也就是计数 1000 时满。因此，计数之前预先在计数器里放进 8192－1000＝7192，开始计数后，计满 1000 个脉冲到 8192 即产生溢出。如果计数初值为 X，则可按以下公式计算定时时间：

$$\text{定时时间}=(2^{13}-X)\times\text{机器周期}$$

其中，机器周期＝12×时钟周期(机器周期的频率是时钟频率的 12 分频)

$$\text{时钟周期}=\frac{1}{\text{晶振频率}}$$

所以，

$$\text{定时时间}=(2^{13}-X)\times\frac{12}{\text{晶振频率}}$$

例如，如果需要定时 3ms(3000μs)，晶振为 12MHz，设计数初值为 X，则根据上述公式可得：

$$3000=(2^{13}-X)\times\frac{12}{12}$$

由此得 $X=5192$

需要说明的是，单片机中的定时器通常要求不断重复定时，一次定时时间到之后，紧接着进行第二次的定时操作。一旦产生溢出，计数器中的值就回到 0，下一次计数从 0 开始，定时时间将不正确。为使下一次的定时时间不变，需要在定时溢出后马上把计数初值送到计数器。

2. 工作方式 1

(1)逻辑电路结构

工作方式 1 是 16 位计数结构的工作方式，计数器由 TH0 全部 8 位和 TL0 全部 8 位构成。其逻辑电路和工作情况与工作方式 0 基本相同，如图 9-6 所示(以定时/计数器 0 为例)。所不同的只是组成计数器的位数，它比工作方式 0 有更宽的计数范围，因此，在实际应用中，工作方式 1 可以代替工作方式 0。

(2)计数初值的计算

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

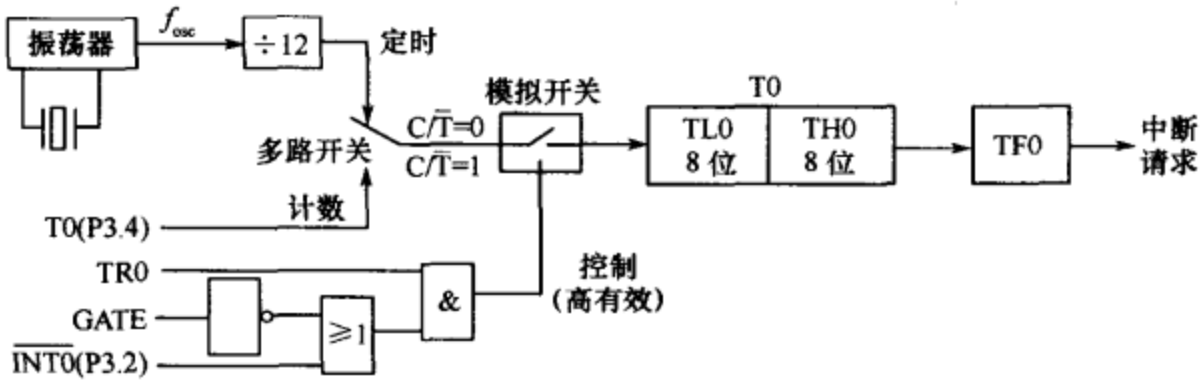


图 9-6　工作方式 1 的逻辑电路结构图

由于工作方式 1 是 16 位计数结构，因此，其最大计数为 $2^{16}=65536$。也就是说，每次计数到 65536 都会产生溢出，去置位 TF0。如果计数初值为 X，则可按以下公式计算定时时间：

$$\text{定时时间}=(2^{16}-X)\times\text{机器周期}$$

3. 工作方式 2

(1)逻辑电路结构

工作方式 0 和工作方式 1 若用于循环重复定时或计数时，每次计满溢出后，计数器回到 0，要进行新一轮的计数，就得重新装入计数初值。因此，循环定时或循环计数应用时就存在反复设置计数初值的问题，这项工作是由软件完成的，需要花费一定时间；这样就会造成每次计数或定时产生误差。如果用于一般的定时，则是无关紧要的；但是有些工作，对时间的要求非常严格，不允许定时时间不断变化，用工作方式 0 和工作方式 1 就不行了，所以就引入了工作方式 2。图 9-7 是定时/计数器 0 在工作方式 2 的逻辑电路结构图。

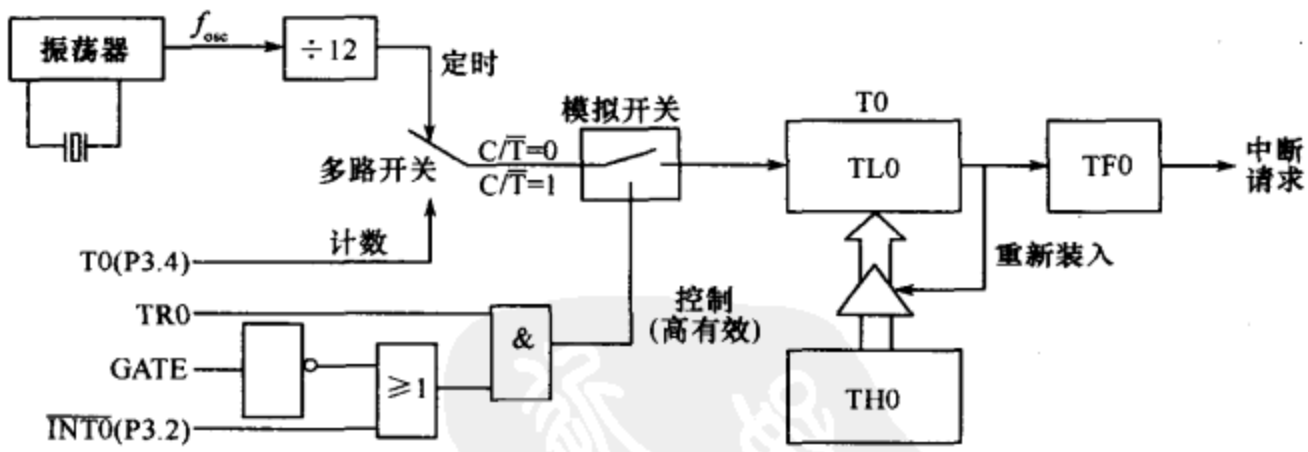


图 9-7　工作方式 2 的逻辑电路结构图

在工作方式 2 下，把 16 位计数器分为两部分，即以 TL0 作计数器，以 TH0 作预置寄存器，初始化时把计数初值分别装入 TL0 和 TH0 中。当计数溢出后，不是像前两种工作方式那样通过软件方法，而是由预置寄存器 TH0 以硬件方法自动给计数器 TL0 重新加载。变软件加载为硬件加载。这样不但省去了用户程序中的重装指令，而且也有利于提高定时精度。

(2)计数初值的计算

由于工作方式 2 是 8 位计数结构，因此，其最大计数值为 $2^8=256$，计数值十分有限。如果计数初值为 X，则可按以下公式计算定时时间：

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

定时时间=(2^8-X)×机器周期

4. 工作方式3

(1)逻辑电路结构

工作方式3的作用比较特殊，只适用于定时器T0。如果企图将定时器T1置为方式3，则它将停止计数，其效果与置TR1=0相同，即关闭定时器T1。

当T0工作在方式3时，它被拆成两个独立的8位计数器TL0和TH0，其逻辑电路结构如图9-8所示。

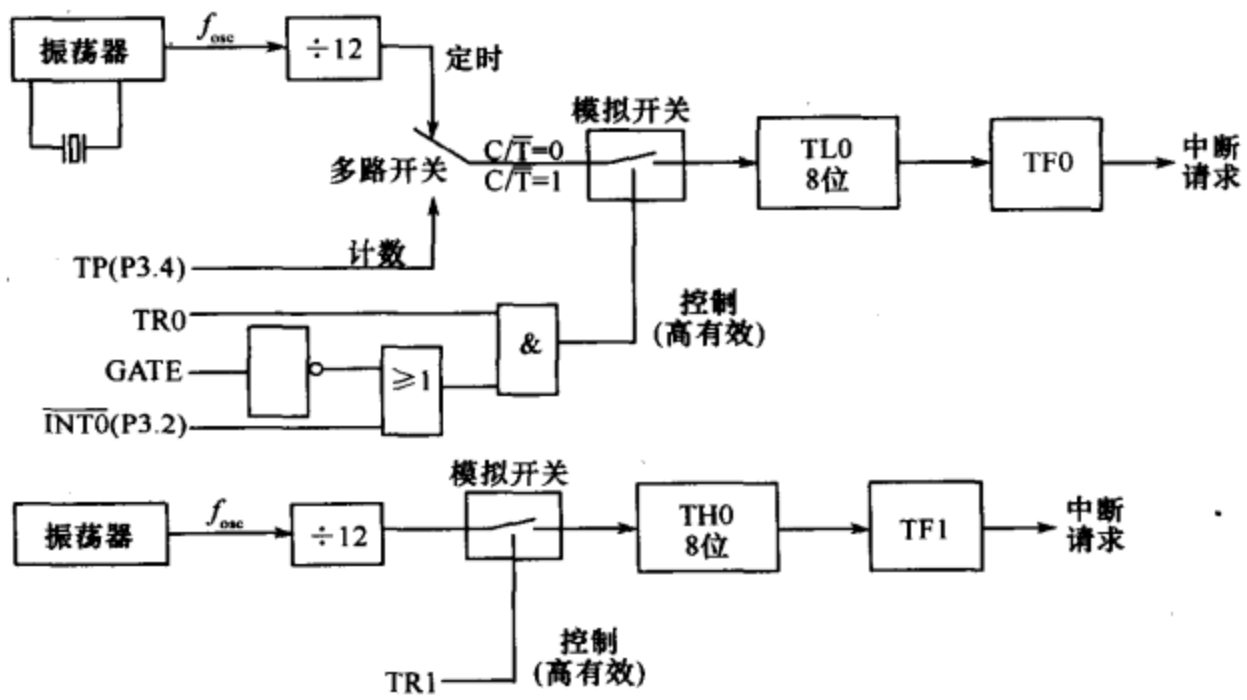


图9-8　工作方式3的逻辑电路结构

图9-8中，上方的8位计数器TL0使用原定时器T0的控制位C/$\overline{T}$、GATE、TR0和$\overline{INT0}$，TL0既可以计数使用，又可以定时使用，其功能和操作与前面介绍的工作方式0或工作方式1完全相同。下方的TH0只能作为简单的定时器使用。而且由于定时/计数器0的控制位已被TL0独占，因此只好借用定时/计数器1的控制位TR1和TF1。即以计数溢出去置位TF1，而定时的启动和停止则受TR1的状态控制。

由于TL0既能作定时器使用也能作计数器使用，而TH0只能作定时器使用却不能作计数器使用，因此在工作方式3下，定时/计数器0可以构成两个定时器或一个定时器和一个计数器。

注意事项　如果定时/计数器0已工作在工作方式3，则定时/计数器1只能工作在工作方式0、工作方式1或工作方式2下，因为它的运行控制住TR1及计数溢出标志位TF1已被定时/计数器0借用。在这种情况下，定时/计数器1通常是作为串行口的波特率发生器使用，以确定串行通信的速率，因为已没有计数溢出标志位TF1可供使用，因此只能把计数溢出直接送给串行口。当作为波特率发生器使用时，只需设置好工作方式，便可自动运行。如要停止工作，只需送入一个把它设置为工作方式3的方式控制字就可以了。因为定时/计数器1不能在工作方式3下使用，如果硬把它设置为工作方式3，就停止工作。

(2)计数初值的计算

由于工作方式3是8位计数结构，因此，其最大计数值为$2^8=256$。如果计数初值为X，则可按以下公式计算定时/计数器0的定时时间：

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

$$定时时间=(2^8-X)\times机器周期$$

四、实验演练

1. 定时/计数器的初始化

在利用定时/计数器进行定时和计数之前，首先要通过软件对它进行初始化，初始化主要包括以下几个方面：

①对工作方式寄存器 TMOD 赋值，确定工作方式。

②计算计数初值，并将其写入寄存器 TH0、TL0 和 TH1、TL1。

③根据需要，对中断允许控制寄存器 IE 赋初值，开放定时器中断。

④使用定时器控制寄存器 TCON 的 TR0、TR1 位启动定时/计数器，TR0 或 TR1 置位之后，计数器即可按规定的工作方式进行定时或开始计数。

2. 应用举例及实验

实验 1：用下载型实验板做如下实验：采用定时器 0 的工作方式 1 写一段延时 5ms 函数，然后，调用延时函数，使 P1 口外接的 LED 每 0.5s 闪亮一次。

(1)对 TMOD 寄存器赋值

使用定时器 0 的工作方式 1，应使 M1M0＝01；为实现定时功能，应使 $C/\overline{T}=0$；为实现定时/计数器 0 的运行控制，则 GATE＝0。定时/计数器 1 不用，有关位设定为 0。因此 TMOD 寄存器初始化为 0x01。

(2)计算计数初值

设延时函数为 delay()，定时时间为 5ms。设计数初值为 X，由于下载型实验板使用 11.059MHz 晶振和工作方式 1，根据

$$定时时间=(2^{16}-X)\times\frac{12}{晶振频率}$$

可得

$$5000=(65536-X)\times\frac{12}{11.059}$$

所以

$$X=60928$$

将 60928 转换为十六进制后为 0xee00。其中，高 8 位为 0xee，放入 TH0；低 8 位为 0x00，放入 TL0。

(3)对 IE 赋初值

对于本例，因采用查询方法，不需要中断，因此，将 IE 置 00H。

(4)启动定时器 0

将定时器控制寄存器 TCON 中的 TR0 设置为 1，可启动定时器 0，TR0 设置为 0，定时器 0 停止定时。

根据以上分析，采用查询方法，设计的源程序如下：

```
#include<reg51.h>
//以下是延时函数
void delay(unsigned int i)                    //延时程序,i 是形式参数
{
TMOD=0x01;                                    //设定定时器 0 为工作方式 1
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
IE=0x00;                          //禁止中断
TR0=1;                            //启动定时器 0
while(i! =0)                      //执行 i 次循环
  {
    TH0=0xee;
    TL0=0x00;                     //设定计数脉冲为 4608 次
    while(TF0! =1);               //计时时间不到,等待;计时时间到,TF0=1
    TF0=0;                        //计时时间到,将 TF0 清 0
    i--;                          //循环次数减 1
  }
  TR0=0;                          //关闭定时器 0
}
//以下是主程序
void main()
{
  for(;;)
    {
    P1=0x00;                      //P1 口 LED 点亮
    delay(100);                   //延时 100×5ms=0.5s
    P1=0xff;                      //P1 口 LED 熄灭
    delay(100);                   //延时 100×5ms=0.5s
    }
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 delay.c。对程序进行编译、链接和调试,产生 delay.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_9/delay 文件夹中。

方法技巧 TF0 是定时/计数器 0 的溢出标志位,当产生溢出后,该 TF0 由 0 变 1,所以查询该位就可以知道定时时间是否已到。该位为 1 后,不会自动清 0,必须用软件将标志位清 0;否则,在下一次查询时,即便时间未到,这一位仍是 1,会出现错误的执行结果。

实验 2:用下载型实验板做如下实验:采用定时器 0 工作方式 1 中断的方法,使 P1 口外接的 LED 每 0.5s 闪亮一次。

(1)对 TMOD 寄存器赋值

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

使用定时器0的方式1，应使M1M0＝01；为实现定时功能，应使C/$\overline{T}$＝0；为实现定时/计数器0的运行控制，则GATE＝0。定时/计数器1不用，有关位设定为0。因此TMOD寄存器初始化为0x01。

(2)计算计数初值

设定时时间为5ms，计数初值为X，由于下载型实验板使用11.059MHz晶振和工作方式1，根据

$$\text{定时时间}=(2^{16}-X)\times\frac{12}{\text{晶振频率}}$$

可得

$$5000=(65536-X)\times\frac{12}{11.059}$$

所以

$$X=60928$$

将60928转换为十六进制后为0xee00。其中，高8位为0xee，放入TH0；低8位为0x00，放入TL0。

(3)对IE赋初值

对于本例，因需要定时器0中断，因此，将IE的EA、ET0位置1。

(4)启动定时器0

将定时器控制寄存器TCON中的TR0设置为1，可启动定时器0，TR0设置为0，定时器0停止定时。

根据以上分析，采用中断方法，设计的源程序如下：

```
#include<reg51.h>
//以下是主函数
char i=100;                    //给变量i赋值
void main()
{
  TMOD=0x01;                   //设定定时器0为工作方式1
  TH0=0xee;
  TL0=0x00;                    //设定计数脉冲为4608次
  P1=0x00;                     //P1口外接LED全亮
  EA=1;                        //开中断
  ET0=1;                       //允许定时器0中断
  TR0=1;                       //启动定时器0
  while(1);                    //无穷循环，等待中断
  }
  //以下是定时器0中断服务程序
void timer0_int(void) interrupt 1
{
  TH0=0xee;
  TL0=0x00;                    //加载计数脉冲为4608次
  i--;                         //循环次数自减1，循环100次，延时100×5ms=0.5s
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    if(i<=0)
    {P1=~P1;                    //P1 口按位取反
    i=100;}                     //重新设定循环次数
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 timer0_int. c。对程序进行编译、链接和调试，产生 timer0_int. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_9/ timer0_int 文件夹中。

实验 3：在下载型实验板上做以下实验：计数定时器 T0 引脚(P3. 4)上外接开关 K3 脉冲数，并通过 P1 口上的 LED 显示出来。

(1)对 TMOD 寄存器赋值

使用定时器 0 的工作方式 1，应使 M1M0=01；为实现计数功能，应使 $C/\overline{T}=1$；为实现定时/计数器 0 的运行控制，则 GATE=0。定时/计数器 1 不用，有关位设定为 0。因此 TMOD 寄存器器初始化为 0x05。

(2)计数初值

由于采用计数方式，计数初值 TH0=0，TL0=0。

(3)对 IE 赋初值

对于本例，不需要定时器 0 中断，因此，将 IE 置 0。

(4)启动定时器 0

将定时器控制寄存器 TCON 中的 TR0 设置为 1，可启动定时器 0，TR0 设置为 0，定时器 0 停止定时。

根据以上分析，设计的源程序如下：

```
#include<reg51. h>
void main()
{
TMOD=0x05;          //定时器 0 为模式 1 计数方式
TH0=0;              //计数初值为 0
TL0=0;
TF0=0;              //TF0 清 0
TR0=1;              //启动定时器 0
while(1)            //无穷循环
P1=TL0;             //将 TL0 输出到 P1 口(注：TH0 中的数无法输出)
}
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 timer0_count. c。对程序进行编译、链接和调试，产生 timer0_count. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl＋F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_9/ timer0_count 文件夹中。

实验 4：在下载型实验板上做以下实验：采用定时器 1 的工作方式 2 的计数方式，计数 T1 引脚(P3. 5)上外接开关 K4 脉冲数，每按两次 K4 计数一次，并通过 P1 口显示出来。

(1)对 TMOD 寄存器赋值

使用定时器 1 的工作方式 2，应使 M1M0＝10；为实现计数功能，应使 C/$\overline{T}$＝1；为实现定时器 1 的运行控制，则 GATE＝0。定时/计数器 0 不用，有关位设定为 0。因此 TMOD 寄存器初始化为 0x60。

(2)计数初值

由于每按两次 K4 计数一次，因此，计数初值为 TH1＝256－2＝0xfe，TL1＝256－2＝0xfe。

(3)对 IE 赋初值

对于本例，不需要定时器 1 中断，因此，将 IE 置 0。

(4)启动定时器 1

将定时器控制寄存器 TCON 中的 TR1 设置为 1，可启动定时器 1，TR1 设置为 0，定时器 1 停止定时。

根据以上分析，采用查询方法，设计的源程序如下：

```
#include<reg51.h>
char i=0;                    //设定计数变量初值
void main()
{
TMOD=0x60;                   //定时器 1 为模式 2 计数方式
TH1=256-2;                   //计数初值为 0
TL1=256-2;
TF1=0;                       //TF1 清 0
TR1=1;                       //启动定时器 1
while(1)                     //无穷循环
  {while(! TF1);             //若 TF1 为 0，等待；若 TF1=1，说明按下开关 K4 两次
  TF1=0;                     //TF1 清 0
  ++i;                       //变量自加 1
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    P1=i;}
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 timer1_count.c。对程序进行编译、链接和调试，产生 timer1_count.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_9/ timer1_count 文件夹中。

实验 5：在下载型实验板上做以下实验：采用定时器 1 的工作方式 1 的中断计数方式，计数 T1 引脚(P3.5)上外接开关 K4 脉冲数，按 5 次 K4 后，P1 口 LED 闪烁不停。

(1)对 TMOD 寄存器赋值

使用定时器 1 的工作方式 1，应使 M1M0=01；为实现计数功能，应使 $C/\overline{T}=1$；为实现定时器 1 的运行控制，则 GATE=0。定时/计数器 0 不用，有关位设定为 0。因此 TMOD 寄存器初始化为 0x50。

(2)计数初值

由于每按 5 次 K4 计数一次，因此，计数初值为 65536−5=65531，将其转换为十六进制后为 0xfffb，因此，TH1=0xff，TL1=0xfb。

(3)对 IE 赋初值

对于本例，需要定时器 1 中断，因此，将 EA、ET1 置 1。

(4)启动定时器 1

将定时器控制寄存器 TCON 中的 TR1 设置为 1，可启动定时器 1，TR1 设置为 0，定时器 1 停止定时。

根据以上分析，采用中断方法，设计的源程序如下：

```
#include<reg51.h>
//以下是延时函数
void Delay(unsigned int i)          //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)                   //变量 i 由实际参数传入一个值,因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
}
//以下是主程序
void main()
{
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
TMOD=0x50;                    //定时器 1 为模式 1 计数方式
TH1=0xff;                     //计数初值
TL1=0xfb;
EA=1;                         //开总中断
ET1=1;                        //开定时器 1 中断
TR1=1;                        //启动定时器 1
while(1);                     //无穷循环
}
//以下是定时器 1 中断服务程序
void timer1_int(void) interrupt 3
{
    for(;;)                   //无穷循环
    {P1=0xff;                 //全灭
    Delay(500);               //延时
    P1=0;                     //全亮
    Delay(500);}              //延时
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 timer1_int. c。对程序进行编译、链接和调试，产生 timer1_int. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_9/ timer1_int 文件夹中。

实验 6：用下载型实验板做以下实验：在 P1.0 脚输出周期为 3s，占空比为 1/3(即高电平时间为 1s)的脉冲信号。用 P1.0 口外接的 LED 模拟显示(P1.0 为高电平时灯灭、低电平时灯亮，即 P1.0 口灯灭 1s，亮 2s)。

(1)对 TMOD 寄存器赋值

使用定时器 0 的方式 1，应使 M1M0=01；为实现定时功能，应使 $C/\overline{T}=0$；为实现定时器 0 的运行控制，则 GATE=0。定时/计数器 1 不用，有关位设定为 O。因此 TMOD 寄存器初始化为 0x01。

(2)计算计数初值

设定时时间为 10ms，则周期为 3s 需要 300 次中断，占空比为 1/3，高电平应为 100 次中断。

设计数初值为 X，由于下载型实验板使用 11.059MHz 晶振和工作方式 1，根据

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

$$\text{定时时间}=(2^{16}-X)\times\frac{12}{\text{晶振频率}}$$

可得
$$10\times1000=(65536-X)\times\frac{12}{11.059}$$

所以
$$X=56320$$

将 56320 转换为十六进制后为 0xdc00。其中，高 8 位为 0xdc；放入 TH0；低 8 位为 0x00；放入 TL0。

(3)对 IE 赋初值

对于本例，因需要定时器 0 中断，因此，将 IE 的 EA、ET0 位置 1。

(4)启动定时器 1

将定时器控制寄存器 TCON 中的 TR0 设置为 1，可启动定时器 0，TR0 设置为 0，定时器 0 停止定时。

根据以上分析，采用中断方法，设计的源程序如下：

```
#include<reg51.h>
sbit P10=P1^0;
//以下是主函数
unsigned  char high=100;                    //给变量 high 赋值
unsigned  int period=300;                   //给变量 period 赋值
unsigned  int time=0;                       //给变量 time 赋值
void main()
  {
    TMOD=0x01;                              //设定定时器 0 为工作方式 1
    TH0=0xdc;
    TL0=0x00;                               //设定计数脉冲
    EA=1;                                   //开中断
    ET0=1;                                  //允许定时器 0 中断
    TR0=1;                                  //启动定时器 0
    while(1);                               //无穷循环，等待中断
    }
    //以下是定时器 0 中断服务程序
void timer0_int(void) interrupt 1
{
    TH0=0xdc;
    TL0=0x00;                               //加载计数脉冲
    time++;                                 //变量 time 自加 1
    if(time==high) P10=0;                   //计数 100 次后，P10 输出低电平
    else if(time==period)
    {P10=1;                                 //计数 300 次后，P10 输出高电平
    time=0;}
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

}

重点提示 定义 period、time 变量时，要将其定义为 int 类型，不可定义为 char 类型，这是由于 char 的最大值为 256。若误将 period、time 定义为 char 类型，将出现编译可通过但不能仿真的错误执行结果。

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 period. c。对程序进行编译、链接和调试，产生 period. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_9/ period 文件夹中。

方法技巧 在以上定时程序中，中断服务程序都要进行重装计数器初值等操作。这样，在定时器溢出发出中断请求到重装完定时器初值，并在此基础上重新开始计数定时时，总有一定的时间间隔，造成定时时间增加了若干微秒。为了减少这种定时误差，就要对重装的计数初值作适当的调整。一般情况下，使定时时间缩短 7 个～10 个机器周期，对于本例，可将定时时间由 10ms 降为 9. 99ms。如果采用定时器工作方式 2(自动重装初值)，则可避免重装初值引起的误差，使定时比较精确，但工作方式 2 的计数长度为 256，定时时间太小。

实验 7：如图 9-9 所示是乐曲《八月桂花遍地开》。编写程序，在下载型实验板上进行硬件仿真实验。

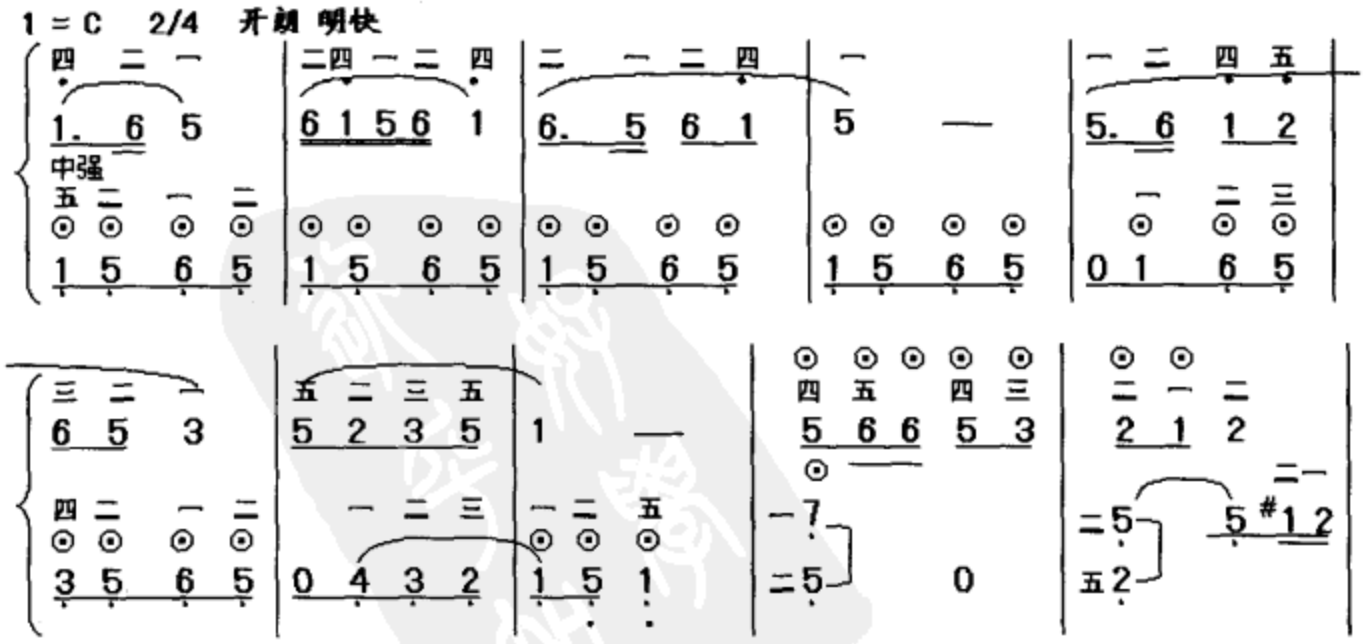


图 9-9 乐曲《八月桂花遍地开》

乐曲演奏的原理是这样的：组成乐曲的每个音符的频率值(音调)及其持续的时间(音长)是乐曲能连续演奏所需的两个基本数据，因此只要控制输出到扬声器的激励信号的频率的高低和持续的时间，就可以使扬声器发出连续的乐曲声。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

(1)音调的控制

首先来看一下怎样控制音调的高低变化。乐曲是由不同音符编制而成的。音符中有7个音名:C、D、E、F、G、A、B。它们分别唱做哆、唻、咪、法、嗦、啦、唏。声音是由空气振动产生的,每个音名,都有一个固定的振动频率,频率的高低决定了音调的高低。音乐的十二平均率规定:每两个八度音(如简谱中的中音1与高音1)之间的频率相差一倍。在两个八度音之间又可分为十二个半音,每两个半音的频率比为$\sqrt[12]{2}$。另外,音名A(简谱中的低音6)的频率为440Hz,音名B(简谱中的音7)到C(简谱中的音1)之间、E(简谱中的音3)到F(简谱中的音4)之间为半音,其余为全音。由此可以计算出简谱中从低音1至高音1之间每个音名对应的频率,如表9-3所列。

表9-3　简谱中的音名与频率的关系

音名	频率/Hz	音名	频率/Hz	音名	频率/Hz
低音1	262	中音1	523	高音1	1047
低音2	294	中音2	587	高音2	1175
低音3	330	中音3	659	高音3	1319
低音4	349	中音4	699	高音4	1397
低音5	392	中音5	784	高音5	1569
低音6	440	中音6	880	高音6	1760
低音7	494	中音7	988	高音7	1976

在下载型实验板上,P3.2引脚经过三极管驱动一个无源蜂鸣器,构成一个简单的音响电路,因此,只要有了某个音的频率数,就能产生出这个音来。现以《八月桂花遍地开》第一个音"高音1"这个音名为例来进行分析。高音1的频率数为1047Hz,则其周期为:

$T=\frac{1}{f}=\frac{1}{1047}\approx 0.96\text{ms}$,即要求P3.2输出周期为0.96ms的等宽方波。也就是说,P3.2每0.48ms高低电平要转换一次。如果调用6.7μs的延时程序(如程序中的delay()延时程序),则延时0.48ms需要调用72次,由于在程序中加了一条unsigned int a=3*m;语句,因此,调用的次数m为72/3=24(十进制),将其转换为十六进制为0x18。即"高音1"的频率常数为0x18。用同样的方法可算出其他音的频率常数。

(2)音长的控制

乐曲中的音符不单有音调的高低,还要有音的长短,如有的音要唱四分之一拍,有的音要唱二拍等。在节拍符号中,用×代表某个音的唱名,×下面无短线为4分音符,有一条短横线代表8分音符,有两条横线代表16分音符,×右边有一条短横线代表2分音符,有"."的音符为符点音符。节拍控制可以通过定时器0中断产生,若定时时间为10ms,以每拍640ms的节拍时间为例,那么,1拍需要循环调用延时子程序64次(64×10ms),转换成十六进制为0x40,即节拍常数为0x40。同理,半拍需要调用延时子程序32次(32×10ms),转换成十六进制为0x20。具体节拍常数如表9-4所列。

表9-4　节拍与调用延时子程序的关系

节拍符号	×̲̲	×̲	×̲.	×	×.	×—	×———
名称	16分音符	8分音符	8分符点音符	4分音符	4分符点音符	2分音符	全音符
拍数	1/4拍	半拍	3/4拍	1拍	1又1/2拍	2拍	4拍
节拍常数	0x10	0x20	0x30	0x40	0x60	0x80	0x100

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

乐曲中，每一音符对应着确定的频率，将每一音符的计数初值和其相应的节拍常数（调用延时程序的次数）作为一组，按顺序将乐曲中的所有常数排列成一个表，然后由查表程序依次取出，产生音符并控制节奏，就可以实现演奏效果。

此外，结束符和休止符可以分别用代码 0x00 和 0xff 来表示，若查表结果为 0x00，则表示曲子终了；若查表结果为 0xff，则产生相应的停顿效果。

(3)用定时器产生 10ms 定时

①对 TMOD 寄存器赋值。

使用定时器 0 的工作方式 1，应使 M1M0＝01；为实现定时功能，应使 $C/\overline{T}=0$；为实现定时器 0 的运行控制，则 GATE＝0。定时/计数器 1 不用，有关位设定为 0。因此 TMOD 寄存器初始化为 0x01。

②计算计数初值。

设定时时间为 10ms，计数初值为 X，由于下载型实验板使用 11.059MHz 晶振和工作方式 1，根据

$$\text{定时时间}=(2^{16}-X)\times\frac{12}{\text{晶振频率}}$$

可得

$$10\times1000=(65536-X)\times\frac{12}{11.059}$$

所以

$$X=56320$$

将 56320 转换为十六进制后为 0xdc00。其中，高 8 位为 0xdc，放入 TH0；低 8 位为 0x00，放入 TL0。

③对 IE 赋初值。

对于本例，因需要定时器 0 中断，因此，将 IE 的 EA、ET0 位置 1。

④启动定时器 1。

将定时器控制寄存器 TCON 中的 TR0 设置为 1，可启动定时器 0，TR0 设置为 0，定时器 0 停止定时。

根据以上分析，编写的源程序如下：

```
#include <reg51.h>
#include <intrins.h>
sbit P32=P3^2;
unsigned char n=0;                    //n 为节拍常数变量，全局变量
unsigned char code music[] ={
0x18, 0x30, 0x1C, 0x10,
0x20, 0x40, 0x1C, 0x10,
0x18, 0x10, 0x20, 0x10,
0x1C, 0x10, 0x18, 0x40,
0x1C, 0x20, 0x20, 0x20,
0x1C, 0x20, 0x18, 0x20,
0x20, 0x80, 0xFF, 0x20,
0x30, 0x1C, 0x10, 0x18,
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

0x20, 0x15, 0x20, 0x1C,
0x20, 0x20, 0x20, 0x26,
0x40, 0x20, 0x20, 0x2B,
0x20, 0x26, 0x20, 0x20,
0x20, 0x30, 0x80, 0xFF,
0x20, 0x20, 0x1C, 0x10,
0x18, 0x10, 0x20, 0x20,
0x26, 0x20, 0x2B, 0x20,
0x30, 0x20, 0x2B, 0x40,
0x20, 0x20, 0x1C, 0x10,
0x18, 0x10, 0x20, 0x20,
0x26, 0x20, 0x2B, 0x20,
0x30, 0x20, 0x2B, 0x40,
0x20, 0x30, 0x1C, 0x10,
0x18, 0x20, 0x15, 0x20,
0x1C, 0x20, 0x20, 0x20,
0x26, 0x40, 0x20, 0x20,
0x2B, 0x20, 0x26, 0x20,
0x20, 0x20, 0x30, 0x80,
0x20, 0x30, 0x1C, 0x10,
0x20, 0x10, 0x1C, 0x10,
0x20, 0x20, 0x26, 0x20,
0x2B, 0x20, 0x30, 0x20,
0x2B, 0x40, 0x20, 0x15,
0x1F, 0x05, 0x20, 0x10,
0x1C, 0x10, 0x20, 0x20,
0x26, 0x20, 0x2B, 0x20,
0x30, 0x20, 0x2B, 0x40,
0x20, 0x30, 0x1C, 0x10,
0x18, 0x20, 0x15, 0x20,
0x1C, 0x20, 0x20, 0x20,
0x26, 0x40, 0x20, 0x20,
0x2B, 0x20, 0x26, 0x20,
0x20, 0x20, 0x30, 0x30,
0x20, 0x30, 0x1C, 0x10,
0x18, 0x40, 0x1C, 0x20,
0x20, 0x20, 0x26, 0x40,
0x13, 0x60, 0x18, 0x20,
0x15, 0x40, 0x13, 0x40,

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
0x18, 0x80, 0x00
};                                  //格式为:频率常数,节拍常数,频率常数,节拍常数
void time0_int()   interrupt 1      //采用定时中断 0 控制节拍
  {
 TH0=0xdc;
 TL0=0x00;
 n--;
  }
void delay (unsigned char m)        //控制频率的延时程序
{
 unsigned int a=3 * m;
 while(--a);
}
void delayms(unsigned int i)        //延时子程序
{
     unsigned int j;
     for(;i>0;i--)                  //变量 i 由实际参数传入一个值,因此 i 不能赋初值
     for(j=0;j<125;j++)
     {;}
}
void main()
{
unsigned char p,m;                  //m 为频率常数变量
unsigned char i=0;
TMOD=0x01;
TH0=0xdc;
TL0=0x00;
EA=1;
ET0=1;
play:while(1)
    {
  a: p=music[i];
   if(p==0x00)
    { i=0, delayms(1000);goto play;}        //如果碰到结束符,延时 1s,回到
                                            开始再来一遍
    else if(p==0xff)
    { i=i+1;delayms(10),TR0=0; goto a;}     //若碰到休止符,延时 10ms,继续
                                            取下一音符
    else
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    {m=music[i++]; n=music[i++];}        //取频率常数和节拍常数
    TR0=1;                               //开定时器 1
    while(n! =0)
    {P32=~P32;delay(m);}                 //等待节拍完成,通过 P1 口输出
                                         音频
    TR0=0;                               //关定时器 1
    }
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 music. c。对程序进行编译、链接和调试,产生 music. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target'Target',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_9/music 文件夹中。

重点提示 在下载型实验板中,P3.2 引脚经过三极管驱动一个无源蜂鸣器,构成一个简单的音响电路,由于 P3.2 同时作为按键输入使用,为了避免按键操作对发声电路的影响,使用了 JP5 插针。若使用音响电路,需要用短路子 JP5 的两个引脚连起来,这时 P3.2 作为输出口来使用。

第三节 串行数据通信技术及实验

一、串行数据通信概述

1. 并行通信与串行通信

计算机与其外围设备之间进行数据交换称为通信。通信的基本方式可分为并行通信和串行通信两种。

并行通信是将组成数据的各位同时传送,并通过并行门(如 P1 口等)来实现,如图 9-10所示为 80C51 系列单片机与外部设备之间 8 位数据并行通信的连接方式。在并行通信中,数据传送线的根数与传送的数据位数相等,传送数据速度快,但所占用的传输线位数多。因此,并行通信适合于短距离通信。

串行通信是指数据一位一位地按顺序传送。串行通信通过串行口来实现。在全双工的串行通信中,仅需一根发送线和一根接收线,如图 9-11 所示为 80C51 单片机与外部设备之间串行通信的连接方式。串行通信可大大节省传送线路的成本,但数据传送速度慢。因此,串行通信适合于远距离通信。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

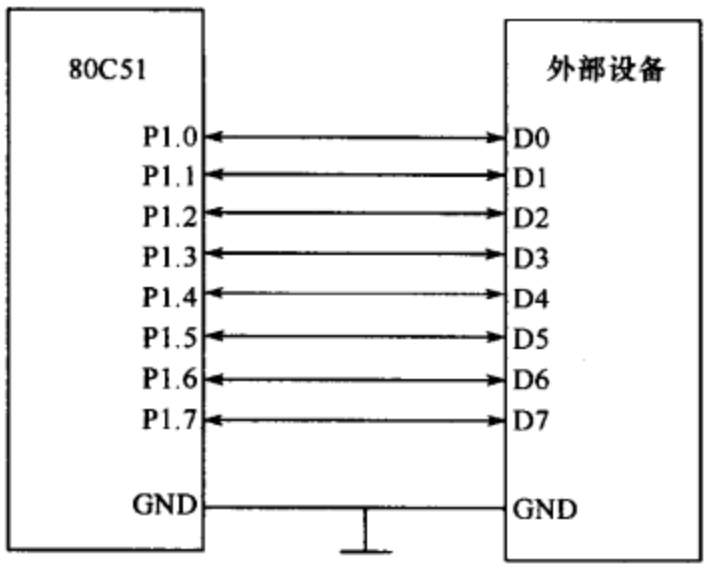


图 9-10　并行通信的连接方式

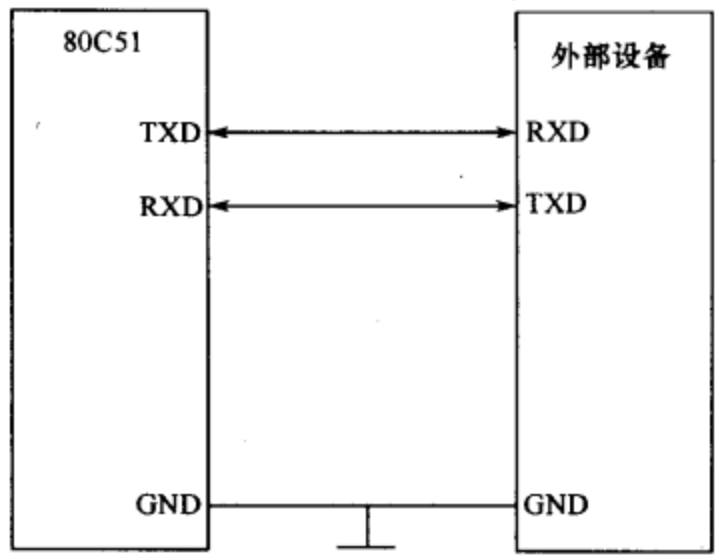


图 9-11　串行通信的连接方式

2. 串行通信的两种基本形式

串行通信根据数据传送时的编码格式不同，分为同步通信和异步通信两种方式。

(1)同步通信

在同步通信中，数据是连续传送的，即数据以数据块为单位传送。在数据开始传送前用同步字符来指示(常约定 1 个或 2 个字符)，并由时钟来实现发送端和接收端同步，即检测到规定的同步字符后，下面就连续按顺序传送或接收数据，直到数据传送结束为止。在同步传送时，数据与数据之间没有间隙，其典型格式如图 9-12 所示。

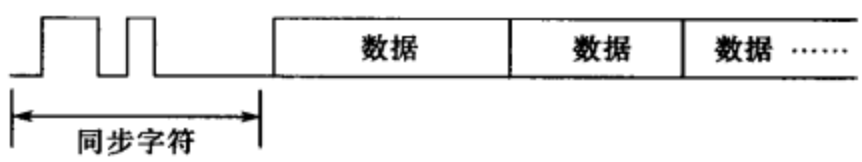


图 9-12　同步传送数据格式

在同步传送中，接收端和发送端必须有同步时钟进行严格同步。在发送时要插入同步字符，接收端检测到同步字符后，便可接收串行数据位。发送端在数据流发送过程中，若出现数据没准备好的情况，便用同步字符来填充，直到下一字符准备好。

(2)异步通信

在异步通信中，数据是不连续传送的。它以字符为单位进行传送，各个字符可以是连续传送，也可以是间断传送。每个被传送字节数据由 4 部分组成：起始位、数据位、校验位和停止位，这 4 部分在通信中称为一帧。首先是一个起始位(0)，它占用一位，用低电平表示；数据位 8 位(规定低位在前，高位在后)；奇偶校验位只占一位(可省略)；最后是停止位(1)，停止位表示一个被传送字符传送的结束，它一定是高电平。接收端不断检测传输线的状态，若连续为 1 后，下一位测到一个 0，就知道发送出一个新字符，应准备接收。由此可见，字符的起始位还被用作同步接收端的时钟，以保证以后的接收能正确进行。如图 9-13所示为异步传送数据格式。

3. 奇偶校验

为了确保传送的数据准确无误，在串行通信中，常在传送过程中进行相应的检测，避免不正确数据被误用。奇偶校验是常用的检测方法。

奇偶校验的工作原理如下：P 是特殊功能寄存器 PSW 的最低位，它的值根据累加器

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

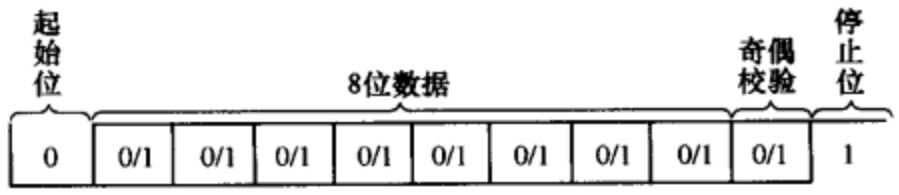


图 9-13　异步传送数据格式

A 中的运算结果而变化。如果 A 中“1”的个数为偶数，则 P=0；如果为奇数，则 P=1。如果在进行串行通信时，把 A 的值（数据）和 P 的值（代表所传送数据的奇偶性）同时传送，那么接收到数据后，也对数据进行一次奇偶校验。如果校验的结果相符（校验后 P=0，而传送过来的数据位也等于 0；或者校验后 P=1，而接收到的校验位也等于 1），就认为接收到的数据是正确的。反之，如果对数据校验的结果是 P=0，而接收到的校验位等于 1 或者相反，那么就认为接收到的数据是错误的。

4. 串行通信的传输速率

串行通信的传输速率用波特率表示。波特率定义为：每秒发送二进制数码的位数，单位为“位/秒”，记作“波特”。

例如，在同步通信中传送数据速度为 450 字符/秒，每个字符又包含 10 位，则波特率为：450 字符/秒×10 位/字符=4500 位/秒(b/s)=4500 波特，一般串行通信的波特率在(50～9600)波特之间。

5. 单工、半双工与全双工通信

在通信线路上按数据传输方向划分有单工、半双工和双工通信方式。

(1)单工通信

单工通信指传送的信息始终是同一方向，而不能进行反向传送设备无发送权。

(2)半双工通信

半双工通信是指信息流可在两个方向上传输，但同一时刻只能有一个站发送，两个方向上的数据传送不能同时进行。

(3)全双工通信

全双工通信是指同时可以作双向通信，两个既可同时发送、接收，又可同时接收、发送。

单工通信、半双工通信和全双工通信示意图如图 9-14 所示。

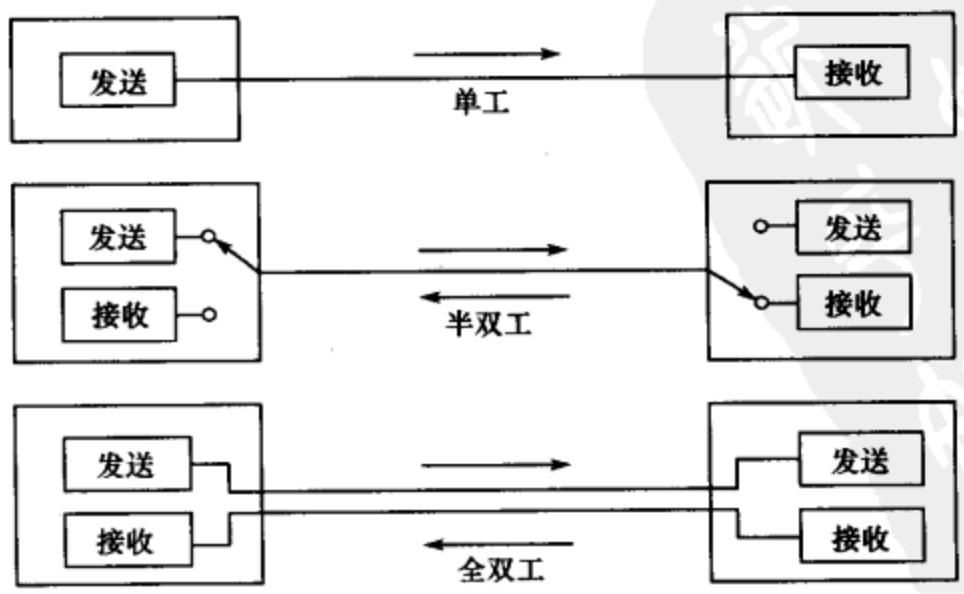


图 9-14　单工通信、半双工通信和全双工通信示意图

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

6. 串行通信总线标准接口

标准接口是指明确定义若干信号线，使接口电路标准化、通用化的接口。借助串行通信标准接口，不同类型的数据通信设备可以很容易实现它们之间的串行通信连接。采用标准接口后，能很方便地把各种计算机、外部设备、单片机等有机地连接起来，进行串行通信。标准异步串行通信接口有 RS-232C、RS-422、RS-423 和 RS-485 等。其中 RS-232C 是由美国电子工业协会(EIA)正式公布的、在异步串行通信中应用最广的标准总线，它包括了按位串行传输的电气和机械方面的规定，适合于短距离或带调制解调器的通信场合。

RS-232C 中的 RS 是英文“推荐标准”的缩写，232 为标识号，C 表示修改次数。RS-232C 总线标准规定了 21 个信号和 25 个引脚，包括一个主通道和一个辅助通道，在多数情况下主要使用主通道。对于一般双工通信，仅需几条信号线就可实现，包括一条发送线、一条接收线和一条地线。

RS-232C 标准规定的数据传输速率为 50b/s、75b/s、100b/s、150b/s、300b/s、600b/s、1200b/s、2400b/s、4800b/s、9600b/s、19200b/s。驱动器允许有 2500pF 的电容负载，通信距离将受此电容限制。信号传输速率为 20kb/s 时，最大传输距离为 15m。传输距离短的另一原因是 RS-232C 属单端信号传送，存在共地噪声和不能抑制共模干扰等问题，因此一般用于短距离通信。

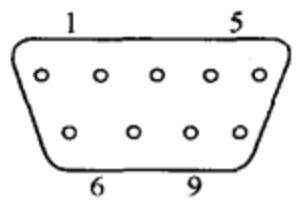


图 9-15　RS－232C 串口引脚排列图

图 9-15 是计算机 9 芯串口引脚排列图，表 9-5 是计算机 9 芯串口引脚信号功能。

表 9-5　计算机 9 芯串口引脚信号功能

脚号	信号名称	方向	信 号 功 能
1	DCD	PC 机←对方	PC 机收到远程信号(载波检测)
2	RXD	PC 机←对方	PC 机接收数据
3	TXD	PC 机→对方	PC 机发送数据
4	DTR	PC 机→对方	PC 机准备就绪
5	GND	—	信号地
6	DSR	PC 机←对方	对方准备就绪
7	RTS	PC 机→对方	PC 机请求接收数据
8	CTS	PC 机←对方	双方已切换到接收状态(清除发送)
9	RI	PC 机←对方	通知 PC 机，线路正常(振铃指示)

由于 RS-232C 是早期(1969 年)为促进公用电话网络进行数据通信而制定的标准，其逻辑电平对地是对称的，与 TTL、MOS 逻辑电平完全不同。逻辑 0 电平规定为＋5V～＋15V 之间，逻辑 1 电平为－5V～－15V 之间，因此，RS-232C 与 TTL 电平连接必须经过电平转换。目前，比较常用的方法是直接选用 232 芯片，图 9-16 为 80C51 单片机串行口电平转换电路。

RS-232C 由于发送器和接收器之间具有公共信号地，不能使用双端信号，因此，共模噪声会耦合到信号系统中，这是迫使 RS-232C 使用较高传输电压的主要原因。即使如此，该标准的信号传输速率也只能达到 20Kb/s，而且最大距离仅 15m。只有在这种条件

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

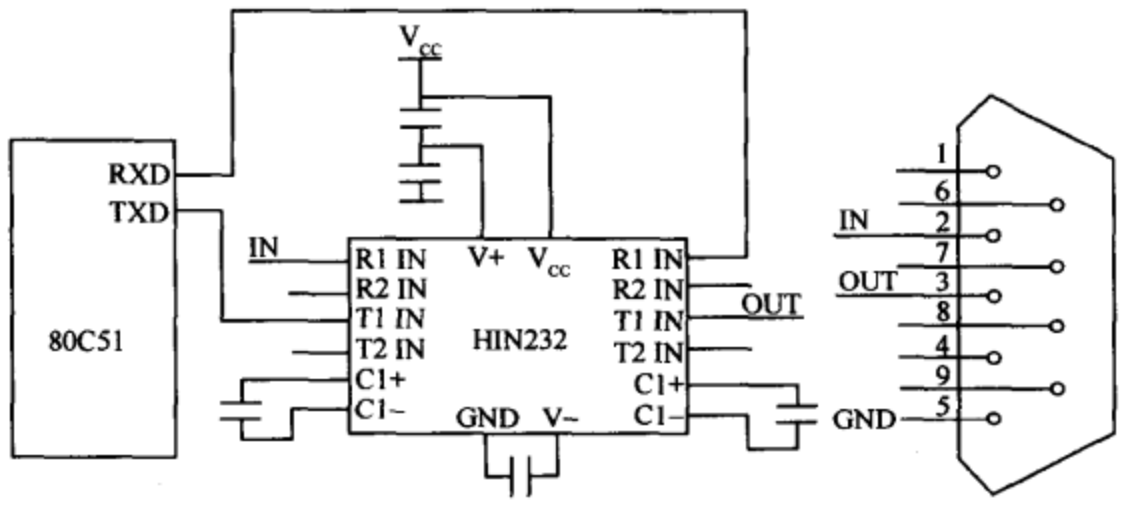


图 9-16　80C51 单片机串行口转换电路

下才能可靠地进行数据传输。

二、串行口的基本结构

串行口电路也称为通用异步收发器(UART)。从原理上说,一个 UART 应包括发送器电路、接收器电路和控制电路。80C51 单片机的 UART 已集成在其中,构成一个全双工串行口,这个口既可以用于网络通信,也可以实现串行异步通信,还可以作为同步移位寄存器使用。

80C51 的串行口通过引脚 RXD(P3.0,串行口数据接收端)和引脚 TXD(P3.1,串行口数据发送端)与外部设备进行串行通信。如图 9-17 所示为 80C51 单片机内部串行口结构示意图。

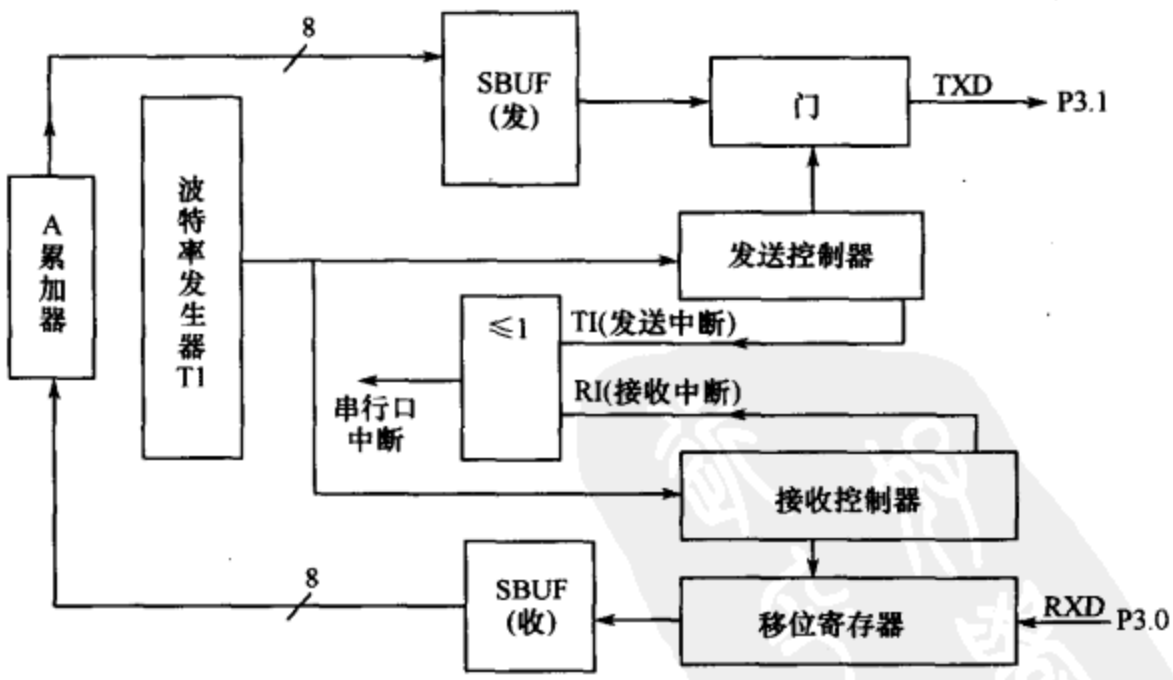


图 9-17　单片机内部串行口结构示意图

图中共有两个串行口缓冲寄存器(SBUF),一个是发送寄存器,一个是接收寄存器,以便 80C51 能以全双工方式进行通信。串行发送时,从片内总线向发送 SBUF 写入数据;串行接收时,从接收 SBUF 向片内总线读出数据。它们都是可寻址的寄存器,但因为发送与接收不能同时进行,所以给这两个寄存器赋以同一地址 99H。

在接收方式下,串行数据通过引脚 RXD(P3.0)进入,由于在接收寄存器之前还有移

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

位寄存器，从而构成了串行接收的双缓冲结构，以避免在数据接收过程中出现帧重叠错误，即在下一帧数据来时，前一帧数据还没有读走。

在发送方式下，串行数据通过引脚 TXD(P3.1)发出。与接收数据情况不同，发送数据时，由于 CPU 是主动的，不会发生帧重叠错误，因此发送电路就不需双重缓冲结构，这样可以提高数据发送速度。

三、串行通信控制寄存器

串行口的通信由 3 个特殊功能寄存器对数据的接收和发送进行控制。它们分别是串行口控制寄存器 SCON、电源控制寄存器 PCON 和中断允许控制寄存器 IE。

1. 串行口控制寄存器 SCON

串行口控制寄存器 SCON 地址为 98H，位地址 9FH～98H，具体格式如下：

位地址	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
位名称	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

在前面学习中断时，已对 SCON 的 RI 和 TI 位进行了简要分析，为便于读者对 SCON 有一个全面的了解，下面对 SCON 各位的控制功能一同进行分析。

(1)SM0、SM1——串行口工作方式选择位

SM0、SM1 对应的 4 种通信方式如表 9-6 所列(表中 f_{osc}为晶振频率)。

表 9-6　串行口工作方式

SM0　SM1	工作方式	功　能	波特率
0　0	工作方式 0	8 位同步移位方式	$f_{osc}/12$
0　1	工作方式 1	10 位 UART	可变
1　0	工作方式 2	11 位 UART	$f_{osc}/32$ 或 $f_{osc}/64$
1　1	工作方式 3	11 位 UART	可变

(2)SM2——多机通信控制位

该位为多机通信控制位，主要用于工作方式 2 和工作方式 3。

当 SM2 为 0 时，则接收到的第 9 位数据(RB8)无论是 0 还是 1，都将接收到的数据装入 SBUF 中，在接收完当前帧后，产生中断申请，RI 置位。

当 SM2 为 1 时，则只有当接收到的第 9 位数据 RB8 为 1，才将接收到的数据装入 SBUF 中，在接收完当前帧后，产生中断申请，RI 置位。若接收到的第 9 位数据 RB8 为 0，则接收到的前 8 位数据丢弃，且不产生中断申请。

在工作方式 0 时，SM2 必须为 0。

(3)REN——允许接收位

由软件置位或清 0，只有当 REN＝1 时，才允许接收，它相当于串行接收的开关；若 REN＝0 时，则禁止接收。

在串行通信过程中，如果满足 REN＝1 且 RI＝1，则启动一次接收过程，一帧数据就装入接收缓冲器 SBUF 中。

(4)TB8——发送数据位 8

在工作方式 2 和工作方式 3 时，TB8 的内容是要发送的第 9 位数据，其值由用户通过软件设置。在双机通信时，TB8 一般作为奇偶校验位使用，在多机通信中，常以 TB8 位的

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

状态表示主机发送的是地址帧还是数据帧。

在工作方式 1 和工作方式 0 中，该位未用。

(5)RB8——接收数据位 8

RB8 是接收数据的第 9 位，在工作方式 2 和工作方式 3 中，接收数据的第 9 位数据放在 RB8 中，它可能是约定的奇偶校验位，也可能是地址/数据标志等。

在工作方式 1 中，RB8 存放的是接收的停止位。

在工作方式 0 中，该位未用。

(6)TI——发送中断标志

当工作方式 0 时，发送完第 8 位数据后，该位由硬件置 1，在其他方式下，于发送停止位之前，由硬件置 1，因此 TI=1，表示帧发送结束，其状态既可供软件查询使用，也可请求中断。TI 位必须由软件清 0。

(7)RI——接收中断标志

当方式 0 时，接收完第 8 位数据后，该位由硬件置 1，在其他方式下，当接收到停止位时，该位由硬件置 1，因此 RI=1，表示帧接收结束。其状态既可供软件查询使用，也可以请求中断。RI 位也必须由软件清 0。

2. 电源控制寄存器 PCON

PCON 主要是为 CHMOS 型单片机 80C51 的电源控制而设置的专用寄存器。单元地址为 87H，不能位寻址。其格式如下：

位号	D7	D6	D5	D4	D3	D2	D1	D0
位符号	SMOD	—	—	—	GF1	GF0	PD	ID

电源控制寄存器 PCON 中，与串行口工作有关的仅有它的最高位 SMOD，SMOD 称为串行口的波特率倍增位。当 SMOD=1 时，波特率加倍。系统复位时，SMOD=0。

3. 中断允许控制寄存器 IE

中断允许控制寄存器 IE 在前面已作了简要介绍，其中，与串行通信有关的位有 ES 位。ES 为串行中断允许位，ES=9，禁止串行中断；ES=1，允许串行中断。

归纳总结 为便于读者对中断、定时/计数器和串行通信有关控制寄存器 TCON、IE、IP、TMOD、SCON 和 PCON 有一个全面的认识，下面列表(见表 9-7)对照进行说明。

表 9-7 与中断、定时/计数器和串行通信有关的寄存器功能一览表

寄存器	位符号	功能	备 注
定时器 控制寄存器 TCON(88H) (可位寻址)	TF0(TF1)	计数溢出标志位	溢出时由硬件自动置 1 使用查询方式查询有效后用软件清 0 使用中断方式转向中断服务程序时由硬件自动清 0
	TR0 (TR1)	定时器运行控制位	由软件进行设置 TR0(TR1)=1，启动定时/计数器工作 TR0(TR1)=0，停止定时/计数器工作
	IE0(IE1)	外中断请求标志位	有外中断时，由硬件自动置 1 中断响应完成后由硬件自动清 0
	IT0(IT1)	外中断请求触发方式控制位	由软件进行设置 IT0(IT1)=1，脉冲触发方式 IT0(IT1)=0，电平触发方式

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

（续）

寄存器	位符号	功能	备　注
中断允许控制寄存器 IE(0A8H) (可位寻址)	EA	中断总允许控制位	由软件进行设置 EA=0,中断总禁止 EA=1,中断总允许
	EX0(EX1)	外中断允许控制位	由软件进行设置 EX0(EX1)=0,禁止外中断 EX0(EX1)=1,允许外中断
	ET0(ET1)	定时中断允许控制位	由软件进行设置 ET0(ET1)=0,禁止定时中断 ET0(ET1)=1,允许定时中断
	ES	串行中断允许控制位	由软件进行设置 ES=0,禁止串行中断 ES=1,允许串行中断
中断优先控制寄存器 IP(0B8H) (可位寻址)	PX0(PX1)	外中断优先级设定位	由软件进行设置 PX0(PX1)=1,优先级高 PX0(PX1)=0,优先级低
	PT0(PT1)	定时中断优先级设定位	由软件进行设置 PT0(PT1)=1,优先级高 PT0(PT1)=0,优先级低
	PS	串行中断优先级设定位	由软件进行设置 PS=1,优先级高 PS=0,优先级低
定时器工作方式控制寄存器 TMOD(89H) (不可位寻址)	M1 M0	工作方式选择位	由软件在字节中设置 M1 M0=0 0,工作方式 0 M1 M0=0 1,工作方式 1 M1 M0=1 0,工作方式 2 M1 M0=1 1,工作方式 3
	GATE	门控位	由软件在字节中设置 GATE=0,由 TR0(TR1)启动定时器 GATE=1,由外中断请求信号启动定时器
	C/$\overline{T}$	定时或计数方式选择位	由软件在字节中设置 C/$\overline{T}$=0,定时工作方式 C/$\overline{T}$=1,计数工作方式
串行口控制寄存器 SCON(98H) (可位寻址)	TI(RI)	串行口发送(接收)中断请求标志位	帧发送(接收)完由硬件自动置 1 转向中断服务程序后用软件清 0
	SM0 SM1	串行口工作方式选择位	由软件进行设置 SM0 SM1=0 0,工作方式 0 SM0 SM1=0 1,工作方式 1 SM0 SM1=1 0,工作方式 2 SM0 SM1=1 1,工作方式 3

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

（续）

寄存器	位符号	功能	备　注
串行口控制寄存器 SCON(98H) （可位寻址）	SM2	多机通信控制位	由软件进行设置 主要用于工作方式 2 和工作方式 3 SM2＝1，只有接收到的第 9 位数据为 1， 才将接收到的前 8 位送入 SUBF，并置位 RI 产生中断 SM2＝0，不论第 9 位数据为 1 还是 0，都将接收到的前 8 位送入 SUBF，并置位 RI 产生中断
	REN	允许接收位	由软件进行设置 REN＝0，禁止接收 REN＝1，允许接收
	TB8	发送数据位 8	由软件进行设置 在工作方式 2、3 时，存放要发送的第 9 位数据
	RB8	接收数据位 8	在工作方式 2、3 时，存放接收到的第 9 位数据
电源控制及波特率选择寄存器 PCON(87H) （不可位寻址）	SMOD	串行口波特率倍增位	由软件在字节中设置 SMOD＝1 时，串行口波特率加倍 SMOD＝0 时，串行口波特率不加倍

四、串行口工作方式

80C51 单片机串行口有 4 种工作方式，分别为工作方式 0、工作方式 1、工作方式 2 和工作方式 3，由串行口控制寄存器 SCON 中最高两位 SM0、SM1 的状态，通过软件设置来决定选择何种工作方式。其中有 8 位、10 位和 11 位为一帧的数据传送格式。

1. 工作方式 0

工作方式 0 以 8 位数据为一帧进行传输，不设起始位和停止位，先发送或接收最低位。其一帧数据格式如下：

……	D0	D1	D2	D3	D4	D5	D6	D7	……

(1)发送与接收

使用工作方式 0 实现数据的移位输入/输出时，实际上是把串行口变成为并行口使用。串行口作为并行输出口使用时，要有“串入并出”的移位寄存器（例如 CD4094、74LS164 等）配合，图 9-18 是采用 CD4094 的连接示意图。

数据预先写入串行口数据缓冲寄存器，然后从串行口 RXD 端在移位时钟脉冲 TXD 的控制下逐位移入 CD4094，当 8 位数据全部移出后，SCON 寄存器的发送中断标志 TI 被自动置 1。其后主程序就可用中断或查询的方法，通过设置 STB 状态的控制，把 CD4094 的内容并行输出。

如果把能实现“并入串出”功能的移位寄存器（例如 CD4014、74LS165 等）与串行口配合使用，就可以把串行口变为并行输入口使用。图 9-19 是采用 CD4014 的连接示意图。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

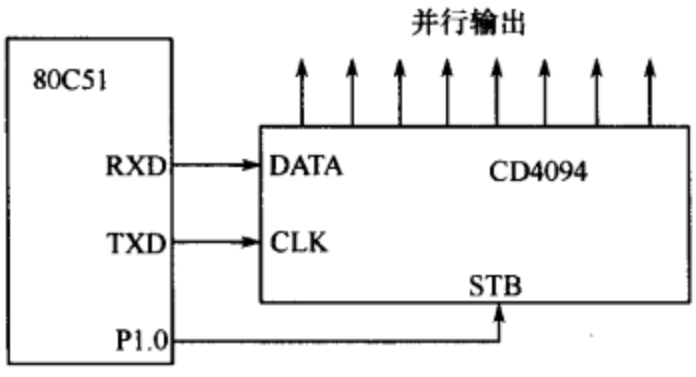


图 9-18　串行口与 CD4094 配合

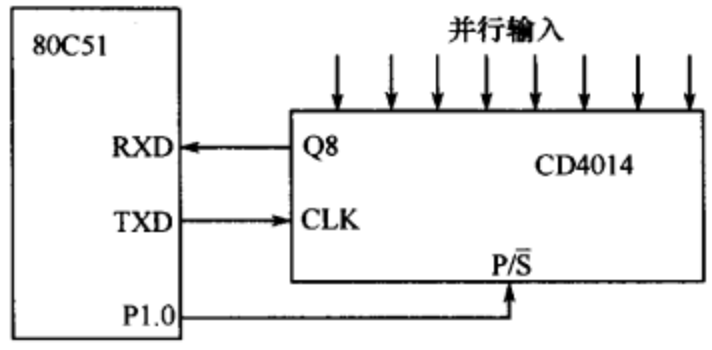


图 9-19　串行口与 CD4014 配合

CD4014 移出的串行数据同样经 RXD 端串行输入，还是由 TXD 端提供移位时钟脉冲。8 位数据串行接收需要有允许接收的控制，具体由 SCON 寄存器的 REN 位实现。REN=0，禁止接收；REN=1，允许接收。当软件置位 REN 时，即开始从 RXD 端输入数据（低位在前），当接收到 8 位数据时，置位接收中断标志 RI。

从以上可以看出，在工作方式 0 下，串行口为 8 位同步移位寄存器输入/输出方式，这种方式不适合用于两个 80C51 单片机芯片之间的直接数据通信，但可以通过外接移位寄存器来实现单片机的接口扩展。

(2)波特率的设定

工作方式 0 时，移位操作（串入或串出）的波特率是固定的，如图 9-29 所示。波特率为单片机晶振频率的 1/12，如晶振频率以 f_{osc} 表示，则波特率为 $f_{osc}/12$。按此波特率也就是一个机器周期进行一次移位，如 $f_{osc}=12MHz$，则波特率为 1M(b/s)，即 1μs 移位一次。

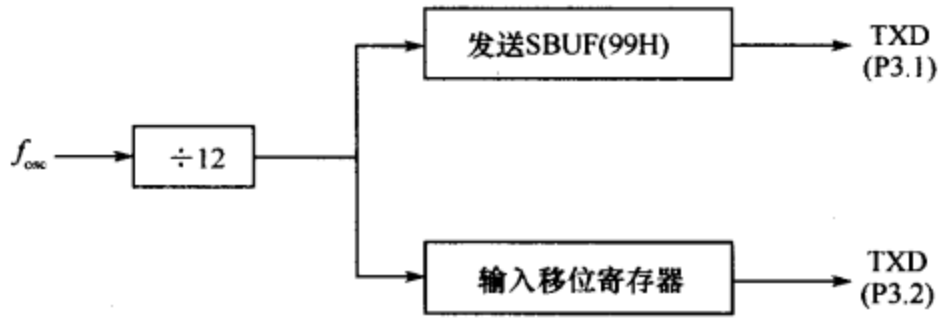


图 9-20　工作方式 0 波特率产生电路

2. 工作方式 1

工作方式 1 以 10 位数据为一帧进行传输，设有 1 个起始位(0)、8 个数据位，1 个停止位(1)，其一帧数据格式如下：

起始	D0	D1	D2	D3	D4	D5	D6	D7	停止

(1)发送与接收

工作方式 1 为 10 位异步通信接口，TXD 和 RXD 分别用于发送与接收数据。收发一帧数据为 10 位，数据位是先低位，后高位。

发送时，数据从 TXD(P3.0)端输出，当 TI=0，执行数据写入发送缓冲器 SBUF 指令时，就启动了串行口数据的发送操作。

启动发送后，串行口自动在起始位清 0，而后是 8 位数据和一位停止位 1，一帧数据为 10 位。数据依次从 TXD 端发出，一帧数据发送完毕，使 TXD 输出线维持在 1 状态下，并将 SCON 寄存器的 TI 置 1，以便查询数据是否发送完毕或作为发送中断申请信号。TI

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

必须由软件清 0。

接收时，数据从 RXD(P3.0)端输入，SCON 的 REN 位应处于允许接收状态(REN=1)。在此前提下，串行口采样 RXD 端，当采样到从 1 向 0 的状态跳变时，就认定是接收到起始位。随后在移位脉冲的控制下，把接收到的数据位移入接收寄存器中。直到停止位到来之后把停止位送入 RB8 中，并置位中断标志位 RI，通知 CPU 从 SBUF 取走接收到的一个字符。

(2)波特率的设定

工作方式 0 的波特率是固定的，但工作方式 1 的波特率则是可变的，如图 9-21 所示。

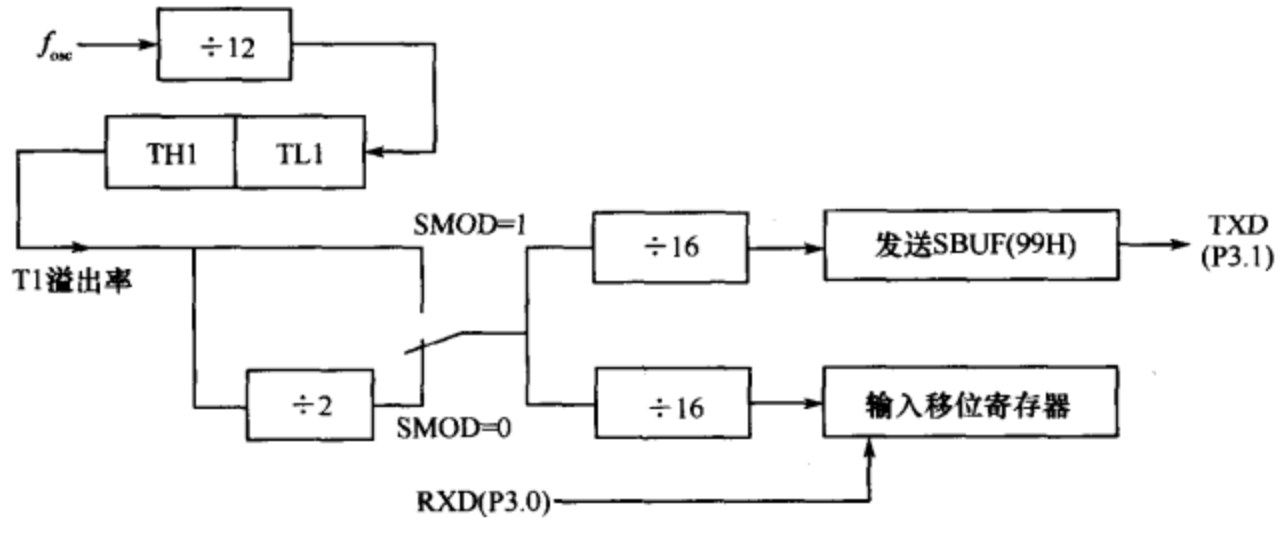


图 9-21　工作方式 1 的波特率产生电路

以定时器 T1 作波特率发生器使用，其值由定时器 1 的计数溢出率来决定，其公式为：

$$\text{波特率}=\frac{2^{\text{smod}}}{32}\times\text{定时器 T1 溢出率}$$

其中，T1 溢出率为一次定时时间的倒数，即

$$\text{定时器 T1 溢出率}=\frac{1}{(2^M-X)\times\dfrac{12}{f_{osc}}}=\frac{f_{osc}}{(2^M-X)\times 12}$$

上式中，X 为计数初值，M 由定时器 T1 的工作方式所决定，即 $M=8$、13 或 16，当定时器 1 作波特率发生器使用时，一般选用工作方式 2。之所以选择工作方式 2，是因为它具有自动加载功能，可避免通过程序反复装入初值所引起的定时误差，使波特率更加稳定。因此，对于定时器 T1 的工作方式 2，定时器 T1 的溢出率又可简化为：

$$\text{定时器 T1 溢出率}=\frac{f_{osc}}{(256-X)\times 12}$$

此时，波特率为：

$$\text{波特率}=\frac{2^{\text{smod}}}{32}\times\frac{f_{osc}}{(256-X)\times 12}=\frac{2^{\text{smod}}\times f_{osc}}{384\times(256-X)}$$

因此，计数初值 X 为：

$$X=256-\frac{2^{\text{smod}}\times f_{osc}}{384\times\text{波特率}}$$

例如，设两机通信的波特率为 2400 波特，若 $f_{osc}=12\text{MHz}$，串行口工作在方式 1，用定时器 T1 作波特率发生器，选定时器工作在方式 2(要禁止 T1 中断，以免产生不必要的中断带来频率误差)。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

若 SMOD=1，则计数初值 X 为：

$$X=256-\frac{2^{smod}\times f_{osc}}{384\times 波特率}=256-\frac{2\times 12\times 10^{-6}}{384\times 2400}\approx 230=0xe6$$

若 SMOD=0，则计数初值 X 为：

$$X=256-\frac{2^{smod}\times f_{osc}}{384\times 波特率}=256-\frac{1\times 12\times 10^{-6}}{384\times 2400}\approx 243=0xf3$$

3. 工作方式 2

工作方式 2 是 11 位为一帧的串行通信方式，即 1 个起始位、9 个数据位和 1 个停止位。其帧格式为：

起始	D0	D1	D2	D3	D4	D5	D6	D7	D8	停止

(1)发送和接收

在工作方式 2 下，字符还是 8 个数据位，只不过增加了一个第 9 个数据位(D8)，而且其功能由用户确定，是一个可编程位。

在发送数据时，应预先在 SCON 的 TB8 位中把第 9 个数据位的内容准备好。

发送数据(D0～D7)向 SBUF 写入，而 D8 位的内容由硬件电路从 TB8 中直接送到发送移位寄存器的第 9 位，并以此来启动串行发送。一个字符帧发送完毕后，将 TI 位置 1，其他过程与工作方式 1 相同。

工作方式 2 的接收过程也与工作方式 1 基本类似，所不同的只在第 9 数据位上，串行口把接收到的前 8 个数据位送入 SBUF，而把第 9 数据位送入 RB8。

(2)波特率的设定

工作方式 2 的波特率是固定的，且有两种，如图 9-22 所示。

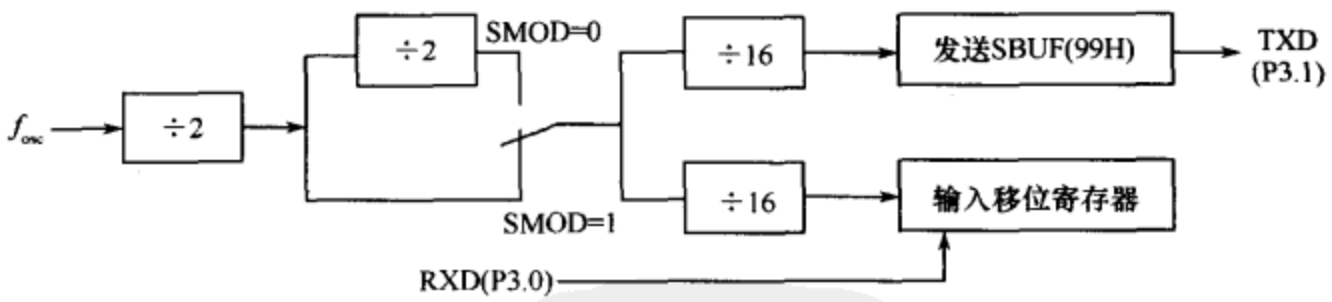


图 9-22　工作方式 2 的波特率产生电路

工作方式 2 的波特率与 PCON 寄存器中 SMOD 位的值有关。当 SMOD=0 时，波特率为 f_{osc} 的 1/64；当 SMOD=1 时，波特率等于 f_{osc} 的 1/32。

4. 工作立式 3

工作方式 3 是 11 位为一帧的串行通信方式，其通信过程与工作方式 2 完全相同，所不同的仅是波特率，工作方式 2 的波特率只有固定的两种，而工作方式 3 的波特率则可由用户根据需要设定，其设定方法与工作方式 1 相同，即通过设置定时器 1 的初值来设置波特率。

重点提示　根据以上介绍可知，串行通信的工作方式 0 和工作方式 1 的波特率是固定的，不需要计算计数初值，而串行通信的工作方式 2 和工作方式 3，则需要计算定时器的计数初值(一般采用定时器的工作方式 2)，以便设置波特率。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

五、实验演练

实验 1:用 AT89C51 实验板做以下实验:让 PS 机通过串口调试软件 comdebug 软件每隔 0.5s 接到单片机发送的字母“A”,设波特率为 1200b/s。

①选用串行口的工作方式 1,SCON 的 SM0 SM1=01,即 SCON=0x40。

②选用定时器 T1 的方式作为波特率发生器使用,则 TMOD=0x20。由于波特率为 1200b/s,则计数初值为:

$$X=256-\frac{2^{smod}\times f_{osc}}{384\times 波特率}$$

这里,SMOD=0(PCON=0),晶振为 12MHz,波特率为 1200b/s

所以 $$X=256-\frac{1\times 12\times 10^{-6}}{384\times 2400}=230=0xe6$$

即 TH1=TL1=0xe6

根据以上分析,编写的源程序如下:

```
#include <AT89X51.h>
void main(void)
{
  unsigned int i;
  SCON=0x40;                    //串口方式 1
  PCON=0;                       //SMOD=0
    REN=1;                      //允许接收
    TMOD=0x20;                  //定时器 1 定时方式 2
    TH1=0xe6;                   //12MHz 1200 波特率
    TL1=0xe6;
    TI=1;
    TR1=1;                      //启动定时器
start: for(i=0;i<50000;i++);    //延时 0.5s
    SBUF='A';
    goto start;
    }
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 com_1.c。对程序进行编译、链接,产生 com_1.hex 目标文件。

②取下 AT89C51 实验板上的 CPU,插入 RF-810 编辑器,对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接,将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件,将波特率调整为 1200b/s,串口选择为 COM1,并打开串口。

⑤打开 AT89C51 实验板电源,串口调试软件开始接收字母 A,如图 9-23 所示。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

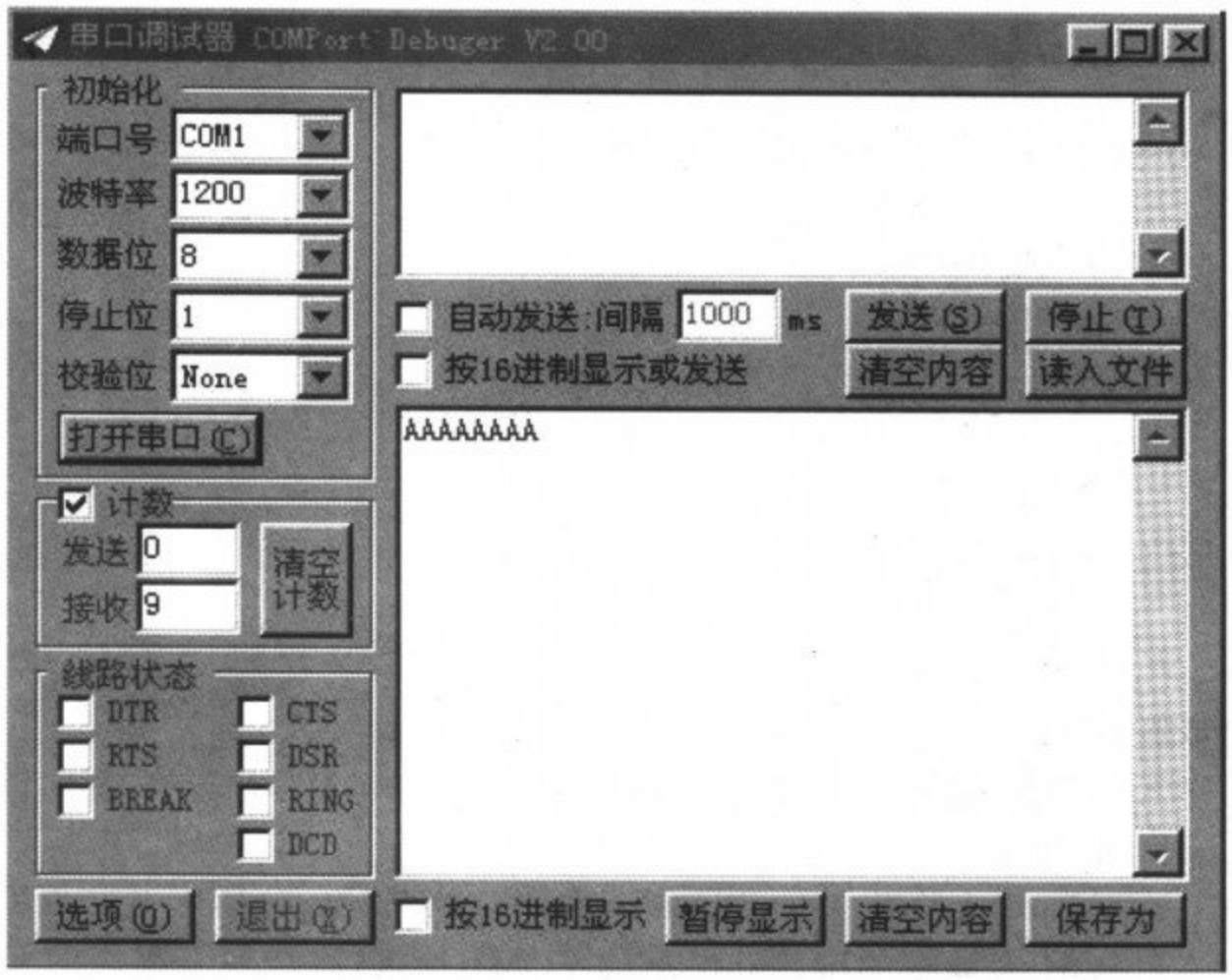


图 9-23　接收字母 A

该实验程序在附赠光盘的 ch_9/com_1 文件夹中。

实验 2:用 AT89C51 实验板做以下实验:让 PC 机通过串口调试软件 comdebug 软件每隔 0.5s 接到单片机发送的字符串“80C51”,设波特率为 1200b/s。

源程序如下:

```
#include <AT89X51.h>
#define uchar unsigned   char
#define uint unsigned int
uchar   idata trdata[10]={'8','0','C','5','1',0x0d,0x0a,0x00};
                                        //0x0d,0x0a 为回车换行的 ASCII 代码
main()
{
  uchar i;
  uint j;
  SCON=0x40;                            //串口方式 1
  PCON=0;                               //SMOD=0
  REN=1;                                //允许接收
  TMOD=0x20;                            //定时器 1 定时方式 2
  TH1=0xe6;                             //12MHz 1200 波特率
  TL1=0xe6;
  TR1=1;                                //启动定时器
  while(1)
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    {
      i=0;
      while(trdata[i]! =0x00)
      {
        SBUF=trdata[i];
        while(TI==0);
        TI=0;
        i++;
      }
      for (j=0;j<50000;j++);          //延时 0.5s
    }
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 com_2. c。对程序进行编译、链接，产生 com_2. hex 目标文件。

②取下 AT89C51 实验板上的 CPU，插入 RF-810 编辑器，对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接，将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开串口调试软件 comdebug 软件，将波特率调整为 1200b/s，串口选择为 COM1，并打开串口。

⑤打开 AT89C51 实验板电源，串口调试软件开始接收字符串 80C51，如图 9-24 所示。

图 9-24　接收字符串

该实验程序在本书附赠光盘的 ch_9/com_2 文件夹中。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

第十章　单片机实用接口C语言编程技术

在单片机开发工作中，根据用户的不同要求，单片机应用系统常常需要配接LED显示器、键盘、LCD显示、A/D(模/数)转换器和D/A(数/模)转换器等外围设备，接口技术就是解决单片机与外围设备之间相互联系的问题。本章将通过一些实例和实验，对单片机开发中的实用接口技术进行分析和介绍。

第一节　LED显示器接口

一、8段LED显示器的结构及原理

1. LED显示器的结构

LED是发光二极管的简称，其PN结是用某些特殊的半导体材料(如磷砷化镓)做成的，当外加正向电压时，可以将电能转换成光能，从而发出清晰悦目的光线。如果将多个LED管排列好并封装在一起，就成为LED显示器。LED发光二极管和常用的8段LED显示器(也称LED数码管)的结构如图10-1所示。

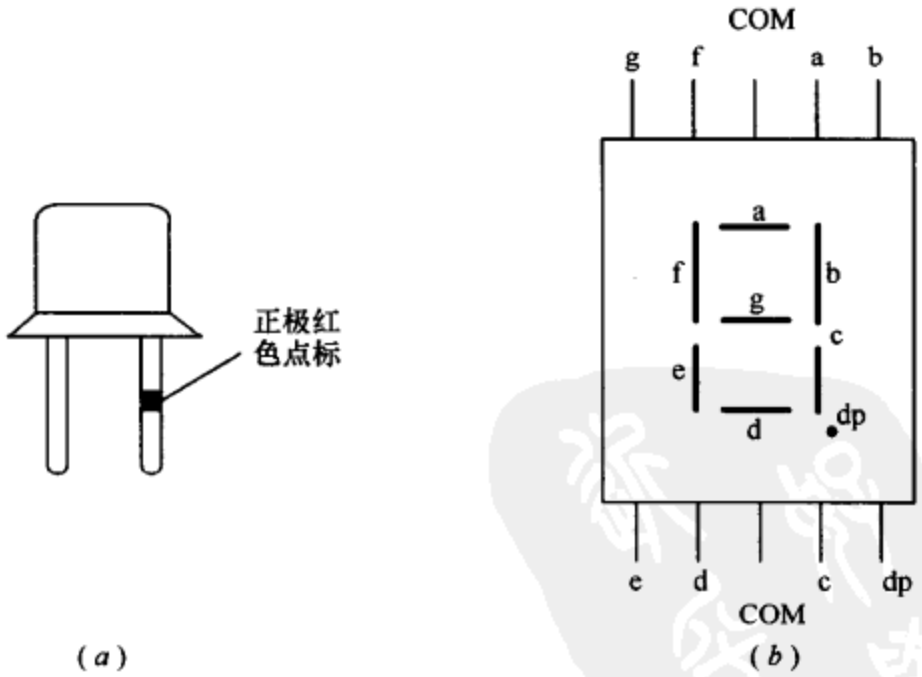


图10-1　LED发光二极管和LED显示器

(a)发光二极管；(b)数码管。

图中，LED显示器内部是8只发光二极管，分别记为a、b、c、d、e、f、g、dp，其中除dp制成圆形用以表示小数点外，其余7只全部制成条形，并排列成如图所示的“8”字形状。每只发光二极管都有一根电极引到外部引脚上，而另外一根电极全部连接在一起，引到外引脚，称为公共极(COM)。

图10-2是LED显示器的电路原理图。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

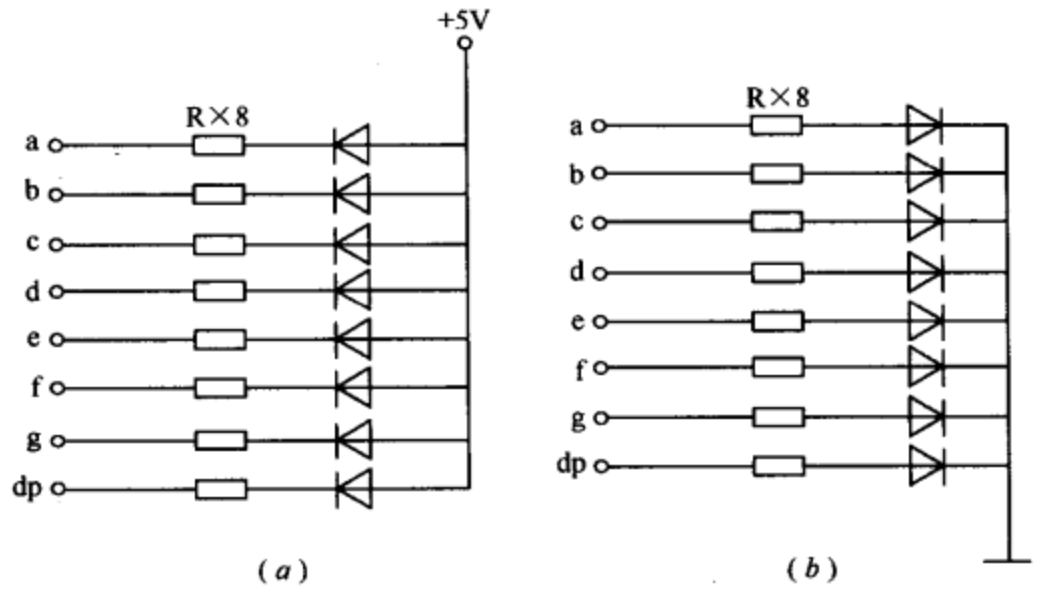


图 10-2　共阳和共阴 LED 显示器电路原理图

(a)共阳极；(b)共阴极。

图(a)是把各个发光二极管的阳极都连在一起，从 COM 端引出，阴极分别从其他 8 根引脚引出，称为共阳结构。使用时，公共阳极接＋5V，这样，阴极端输入低电平的发光二极管就导通点亮，而输入高电平的段则不能点亮。

图(b)是把各个发光二极管的阴极都接在一起，从 COM 端引出，阳极分别从其他 8 根引脚引出，称为共阴结构。使用时，公共阴极接地，这样，阳极端输入高电平的发光二极管就导通点亮，而输入低电平的段则不能点亮。

在购买和使用 LED 显示器时，必须说明是共阴还是共阳结构。

2. LED 显示器的原理

根据 LED 显示器结构可知，如果希望显示“8”字，那么除了“dp”管不要点亮以外，其余管全部点亮。同理，如果要显示“1”，那么，只需 b、c 两个发光二极管点亮。对于共阳结构，就是要把公共端 COM 接到电源正极，而 b、c 两个负极分别经过一个限流电阻后接低电平；对于共阴结构，就是要把公共端 COM 接低电平(电源负极)，而 b、c 两个正极分别经一个限流电阻后接到高电平。接照同样的方法分析其他显示数和字形码，如表 10-1 所列。

表 10-1　8 段 LED 数码管段位与显示字形码的关系

显示	共阳									共阴								
	dp	g	f	e	d	c	b	a	十六进制数	dp	g	f	e	d	c	b	a	十六进制数
0	1	1	0	0	0	0	0	0	0C0H	0	0	1	1	1	1	1	1	3FH
1	1	1	1	1	1	0	0	1	0F9H	0	0	0	0	0	1	1	0	06H
2	1	0	1	0	0	1	0	0	0A4H	0	1	0	1	1	0	1	1	5BH
3	1	0	1	1	0	0	0	0	0B0H	0	1	0	0	1	1	1	1	4FH
4	1	0	0	1	1	0	0	1	99H	0	1	1	0	0	1	1	0	66H
5	1	0	0	1	0	0	1	0	92H	0	1	1	0	1	1	0	1	6DH
6	1	0	0	0	0	0	1	0	82H	0	1	1	1	1	1	0	1	7DH
7	1	1	1	1	1	0	0	0	0F8H	0	0	0	0	0	1	1	1	07H
8	1	0	0	0	0	0	0	0	80H	0	1	1	1	1	1	1	1	7FH

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

（续）

显示	共阳									共阴								
	dp	g	f	e	d	c	b	a	十六进制数	dp	g	f	e	d	c	b	a	十六进制数
9	1	0	0	1	0	0	0	0	90H	0	1	1	0	1	1	1	1	6FH
A	1	0	0	0	1	0	0	0	88H	0	1	1	1	0	1	1	1	77H
B	1	0	0	0	0	0	1	1	83H	0	1	1	1	1	1	0	0	7CH
C	1	1	0	0	0	1	1	0	0C6H	0	0	1	1	1	0	0	1	39H
D	1	0	1	0	0	0	0	1	0A1H	0	1	0	1	1	1	1	0	5EH
E	1	0	0	0	0	1	1	0	86H	0	1	1	1	1	0	0	1	79H
F	1	0	0	0	1	1	1	0	8EH	0	1	1	1	0	0	0	1	71H
H	1	0	0	0	1	0	0	1	89H	0	1	1	1	0	1	1	0	76H
L	1	1	0	0	0	1	1	1	0C7H	0	0	1	1	1	0	0	0	38H
P	1	0	0	0	1	1	0	0	8CH	0	1	1	1	0	0	1	1	73H
U	1	1	0	0	0	0	0	1	0C1H	0	0	1	1	1	1	1	0	3EH
Y	1	0	0	1	0	0	0	1	91H	0	1	1	0	1	1	1	0	6EH
灭	1	1	1	1	1	1	1	1	0FFH	0	0	0	0	0	0	0	0	00H

重点提示 这种规定和定义并非是一成不变的，在实际应用中，为了减少走线交叉，便于电路板布线，设计者可自行定义 8 段 LED 显示器的引脚符号，并根据其工作原理编制相应的“段码表”以及设计与之相匹配的连接电路。

二、LED 显示器的显示方式

点阵式显示器件不仅可以显示数字，还可显示各种符号、文字甚至图形，但点阵式显示电路较复杂，仅使用硬件电路难以实现，必须在单片机（如 8051）的控制下配以相应的软件才能实现。下面主要以 8 段 LED 显示器为例来介绍数码显示的基本方法和相关电路。

1. 静态显示方法

所谓静态显示，就是当显示某一个数字时，代表相应笔划的发光二极管恒定发光，例如 8 段数码管的 a、b、c、d、e、f 笔段亮时显示数字“0”；b、c 亮时显示“1”；a、b、d、e、g 亮时显示“2”等等。

图 10-3 是 LED 显示器静态显示电路。每位数码管的公共端 COM 接在一起接正电压（共阳极显示器）或接地（共阴极显示器）。段选线分别通过限流电阻与段驱动电路连接。限流电阻的阻值根据驱动电压和 LED 的额定电流确定。

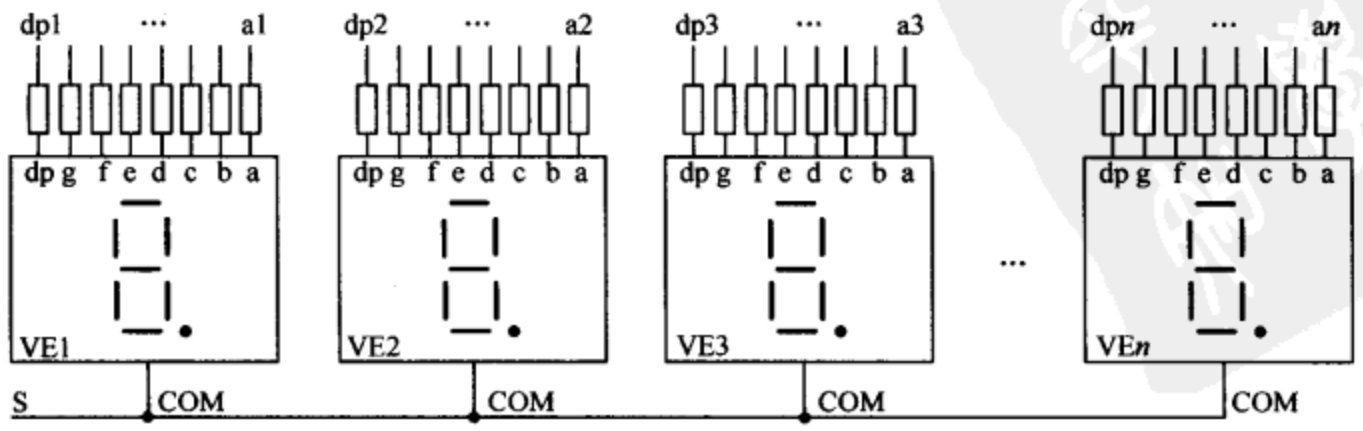


图 10-3 静态显示电路

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

静态显示的优点是显示稳定，在驱动电流一定的情况下显示的亮度高，缺点是使用元器件较多（每一位都需要一个驱动器，每一段都需要一个限流电阻），连接线多。

2. 动态显示方法

上面介绍的静态显示方法的最大缺点是使用元件多、引线多、电路复杂，而动态显示使用的元件少、引线少、电路简单。仅从引线角度考虑，静态显示从显示器到控制电路的基本引线数为“段数×位数”，而动态显示从显示器到控制电路的基本引线数为“段数＋位数”。以 6 位显示为例，动态显示时的基本引线数为 7＋6＝13（无小数点）或 8＋6＝14（有小数点），而静态显示的基本引线数为 7×6＝42（无小数点）或 8×6＝48（有小数点）。在实际应用中，显示器与主电路往往不会安装在同一块电路板上，有时甚至有一段距离，因此静态显示的引线数大多会给实际安装、加工工艺带来困难。

动态显示所显示的若干位数是逐位轮流显示的，周而复始不断循环，只要轮流的速度足够快（每秒轮流 50 次以上），由于人眼“视觉暂留”的物性，感觉不到显示器的闪动，所看到的是连续显示一组数字。图 10-4（a）是三位 LED 显示器采用动态显示方法的接线图，它把 3 个 LED 显示器的各个段对应并联，而各位的公共极（本图采用共阴 LED，公共极是发光二极管的阴极）受到独立控制，控制信号分别为 S1、S2、S3，其时序关系如图（b）所示。

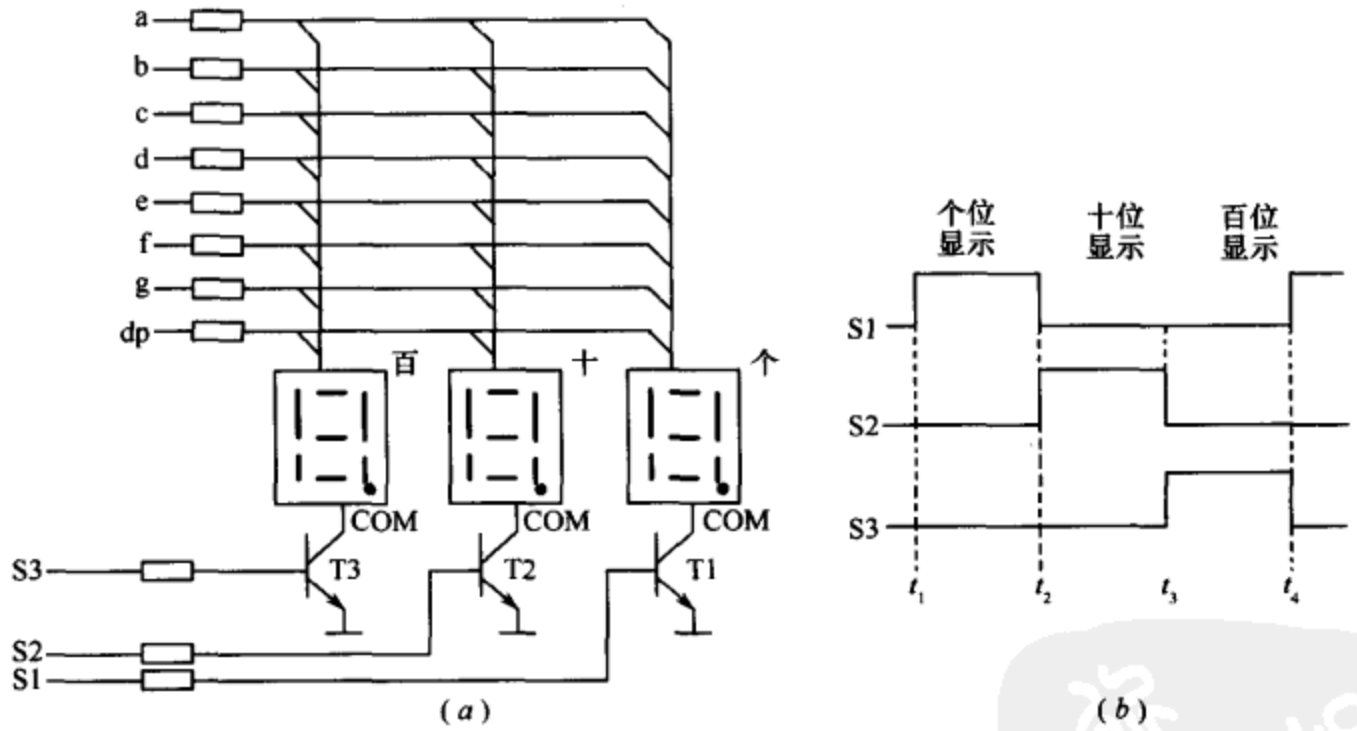


图 10-4　动态显示电路

在 t_1～t_2 期间，单片机输出个位段码加到各个 LED 数码管的阳极 dp～a；在这期间，控制信号 S1 为高电平，三极管 T1 饱和导通，个位 LED 的阴极接地，使个位 LED 显示器相应的字段点亮，显示个位数；而控制信号 S2、S3 为低电平，故三极管 T2、T3 截止，使十位、百位 LED 的阴极不能接向低电平，因此十位、百位的灯不亮。在 t_2～t_3 期间，单片机输出“十位”的段码加到各个 LED 数码管的阳极 dp～a；相应地使控制信号 S2 为高电平，而 S1、S3 为低电平，故只有三极管 T2 饱和导通，而 T1、T3 截止，即只有“十位”LED 点亮。同理，在 t_3～t_4 期间，只有“百位”LED 点亮。此规律周而复始不断循环，就使得个位、十位、百位 3 个 LED 依次轮流循环显示，这就是动态扫描显示的原理，控制信号 S1、S2、S3 称为“位扫描”信号。从以上分析可以看到，动态扫描显示方式可以十分有效地减

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

少元件、减少引线，尤其是在显示位数较多时，其优越性就更为突出，因此在单片机显示电路中，大多采用这种方式。

三、实验演练

实验 1：用 AT89C51 实验板做以下实验：利用其中的一只 LED 显示器循环显示 0～9，时间间隔为 1s。有关电路如图 10-5 所示。

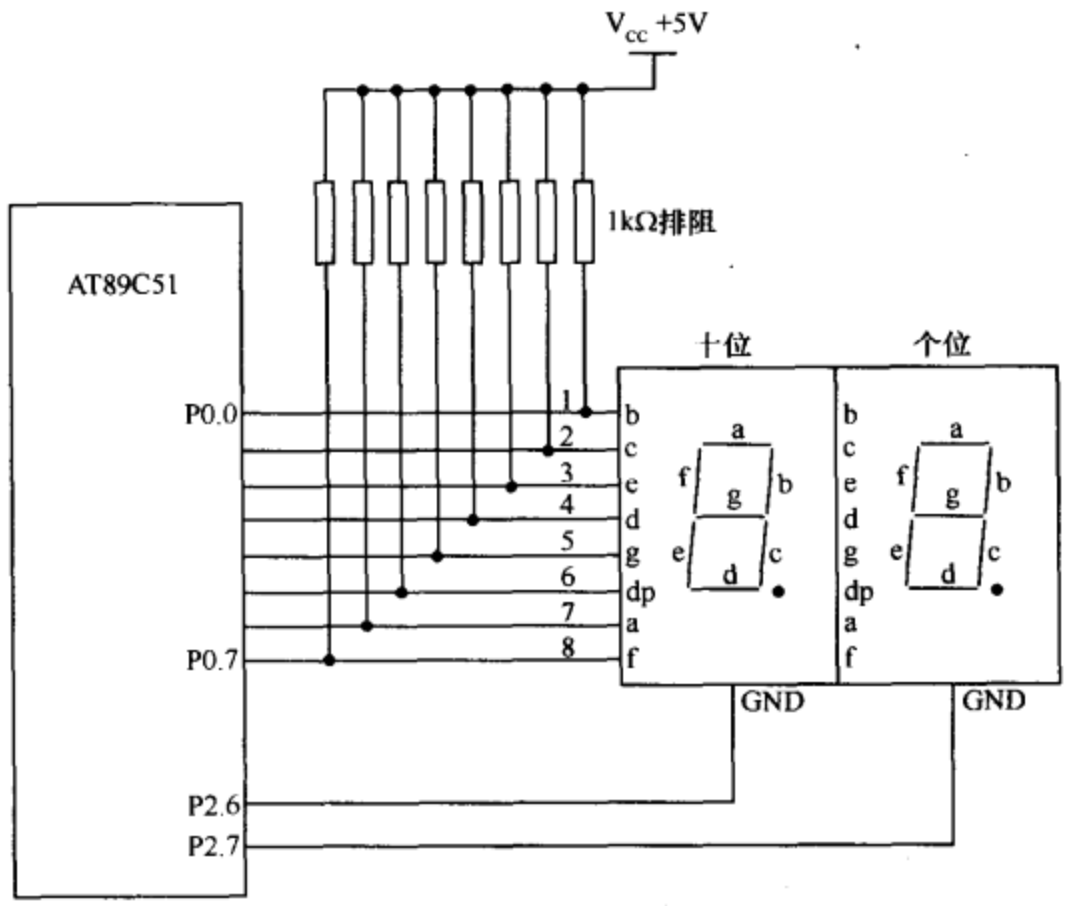


图 10-5　AT89C51 实验开发板数码管电路

图中，P0 口和 P2 口的部分(P2.6、P2.7)引脚构成了两位 LED 数码管驱动电路，数码管采用了共阴型。即对应的 a、b、c、d、e、f、dp 是二极管的正极，所有二极管的负极连在一起，构成公共端。+5V 通过 1kΩ 的排阻直接给数码管的 8 个段位供电，P2.6 和 P2.7 分别控制数码管的十位和个位的供电，当 P2.6 或 P2.7 变成低电平时，相应的位可以吸入电流。

这里，以个位 LED 数码管循环显示 0～9 为例进行实验，源程序如下：

```
#include<reg51.h>
sbit P26=P2^6;
sbit P27=P2^7;
//以下是延时函数
void Delay(unsigned int i)      //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)               //变量 i 由实际参数传入一个值,因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
}
//以下是主程序
void main()
{
  unsigned char i;
  unsigned char code tab[]={0xcf,0x03,0x5d,0x5b,0x93,0xda,0xde,0x43,0xdf,0xdb};
while(1)
  {
    for(i=0;i<=9;i++)
    {P0=tab[i];                    //将段码表送 P0 口
      P26=1;                       //P2.6 置 1,十位 LED 数码管熄灭
      P27=0;                       //P2.7 清 0,个位 LED 数码管点亮
      Delay(1000);}                //延时 1s
  }
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 disp_1.c。对程序进行编译、链接,产生 disp_1.hex 目标文件。

②取下 AT89C51 实验板上的 CPU,插入 RF-810 编辑器,对 CPU 进行编程。

③将 AT89C51 实验板的串口和电脑的 COM1 连接,将烧写有程序的 CPU 插入 AT89C51 实验板的插座中。

④打开 AT89C51 实验板电源,观察实验现象。

该实验程序在本书附赠光盘的 ch_10/disp_1 文件夹中。

重点提示 AT89C51 实验开发板上的段码表与前面介绍的段码表不一致,这是因为该 LED 段与 P0 口的接法不一致造成的。LED 显示器的各段与 P0 口的对应关系如下所示:

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
f	a	dp	g	d	e	c	b

从对应关系中可以看出,当显示 P0 口输出的数为 0CFH(1100 1111)时,LED 数码管中对应的 fadecb 为高电平,因此,LED 数码管显示为 0;同理,可分析出数字 1、2……9 的段码表。

实验 2:用下载型实验板做以下实验:利用其中的一只 LED 显示器循环显示 0～9,时间间隔为 1s。有关电路如图 10-6 所示。

下载型实验板显示器共 6 位,LED 的段控接 P0 口,位控接 P2 口的 P2.2～P2.7。P0 口作段控口输出时,由于其内部是漏极开路的,因此要外接 1kΩ 左右的上拉电阻(图中未画出),而 P2 口由于其内部已有上拉电阻,无需外接。

下载型实验板的 LED 是共阳的,要点亮某段,必须在位控端加高电平,段控端加低电平。一般来说,LED 上通有 5mA～30mA 的电流即可点亮,因此,可计算段控端的限流电

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

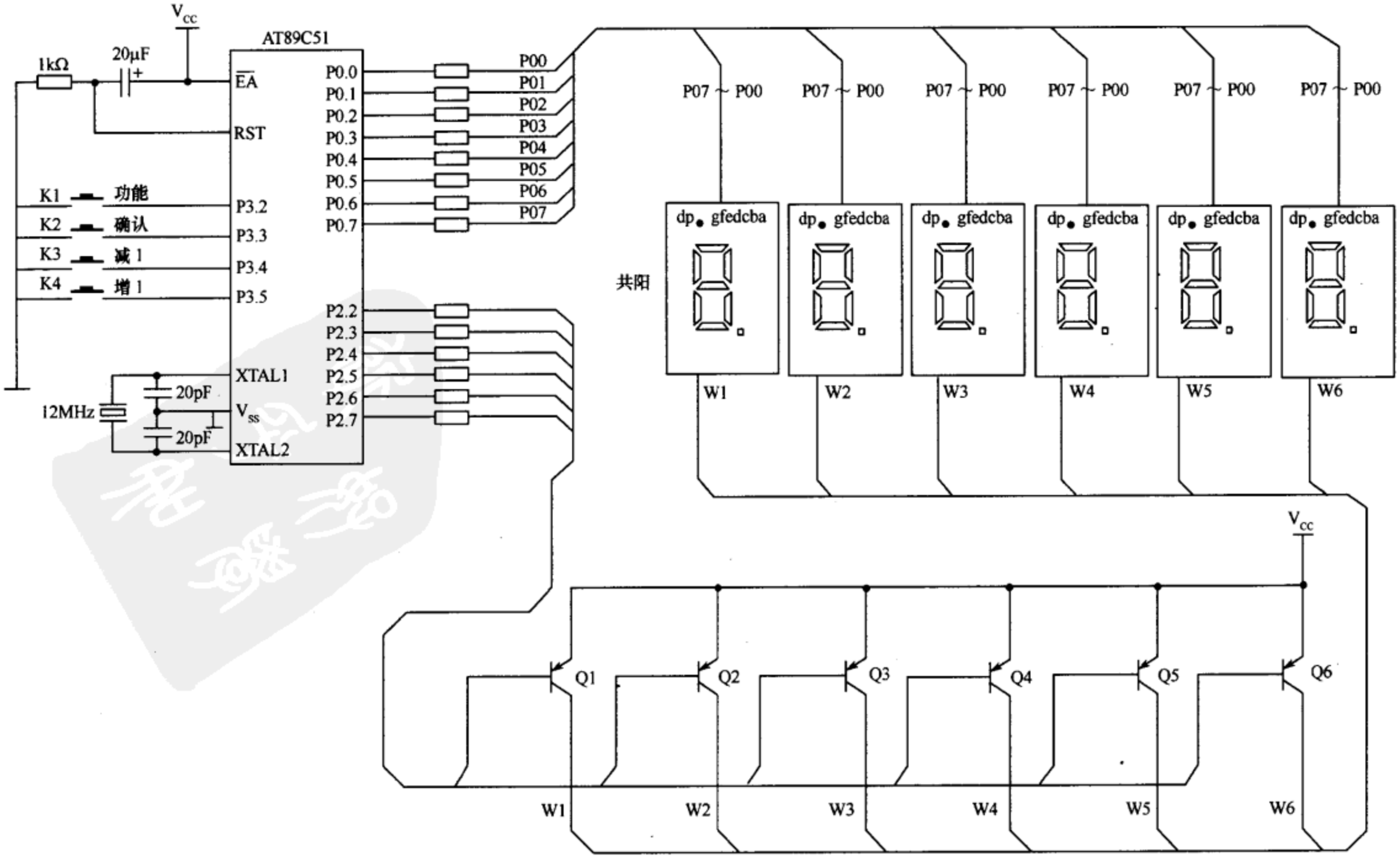


图10-6　下载型实验板数码管电路原理图

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

阻(R1～R8)为(5－1.5)V/(5～30)mA,式中的 1.5V,是发光二极管的每段压降。表 10-2、表 10-3 是根据图 10-6 所示电路原理图列出的字形码。

需要说明的是,由于位控端加有三极管反相器,因此,P2.2～P2.7 的相应脚为低电平时,LED 的位控端才为高电平。

表 10-2　根据数码管连接方法写出的字形码(小数点不亮)

显示	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	字形码
	dp	g	f	e	d	c	b	a	
0	1	1	0	0	0	0	0	0	0C0H
1	1	1	1	1	1	0	0	1	0F9H
2	1	0	1	0	0	1	0	0	0A4H
3	1	0	1	1	0	0	0	0	0B0H
4	1	0	0	1	1	0	0	1	99H
5	1	0	0	1	0	0	1	0	92H
6	1	0	0	0	0	0	1	0	82H
7	1	1	1	1	1	0	0	0	0F8H
8	1	0	0	0	0	0	0	0	80H
9	1	0	0	1	0	0	0	0	90H
灭	1	1	1	1	1	1	1	1	0FFH

表 10-3　根据数码管连接方法写出的字形码(小数点亮)

显示	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	字形码
	dp	g	f	e	d	c	b	a	
0	0	1	0	0	0	0	0	0	40H
1	0	1	1	1	1	0	0	1	79H
2	0	0	1	0	0	1	0	0	24H
3	0	0	1	1	0	0	0	0	30H
4	0	0	0	1	1	0	0	1	19H
5	0	0	0	1	0	0	1	0	12H
6	0	0	0	0	0	0	1	0	2H
7	0	1	1	1	1	0	0	0	78H
8	0	0	0	0	0	0	0	0	00H
9	0	0	0	1	0	0	0	0	10H
灭	0	1	1	1	1	1	1	1	7FH

若以最高位 LED 数码管循环显示 0～9,不显示小数点,则实验源程序如下:

```
#include<reg51.h>
sbit P22=P2^2;
sbit P23=P2^3;
sbit P24=P2^4;
sbit P25=P2^5;
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
sbit P26=P2^6;
sbit P27=P2^7;
//以下是延时函数
void Delay(unsigned int i)      //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)               //变量 i 由实际参数传入一个值,因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
}
//以下是主程序
void main()
{
  unsigned char i;
  unsigned char code tab[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
 while(1)
 {
    for(i=0;i<=9;i++)
    {P0=tab[i];                 //将段码表送 P0 口
      P27=0;                    //P2.7 清 0,LED 数码管点亮
      P26=1;                    //P2.6 置 1,LED 数码管熄灭
      P25=1;                    //P2.7 置 1,LED 数码管熄灭
      P24=1;                    //P2.7 置 1,LED 数码管熄灭
      P23=1;                    //P2.7 置 1,LED 数码管熄灭
      P22=1;                    //P2.7 置 1,LED 数码管熄灭
      Delay(1000);}             //延时 1s
  }
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 disp_2.c。对程序进行编译、链接和调试,产生 disp_2.hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target ',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_10/disp_2 文件夹中。

实验 3:用下载型实验板中的 LED 显示器显示“050906”。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

因为要显示6位不同的数字，所以必须用动态扫描的方法来实现，先第1位显示2ms，再第2位显示2ms……最后第6位显示2ms，则显示6位共需12ms，即每秒扫描80多次，因为人眼的视觉残留效应，就可以看到6位不同的数字稳定显示。

源程序如下：

```
#include<reg51.h>
//以下是延时函数
void Delay(unsigned int i)      //延时程序,i是形式参数
{
    unsigned int j;
    for(;i>0;i--)               //变量i由实际参数传入一个值,因此i不能赋初值
    for(j=0;j<125;j++)
    {;}
}
//以下是主程序
void main()
{
  unsigned char code tab[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,
  0x90};//段码表
  unsigned char code tab_bit[]={0x7f,0xbf,0xdf,0xef,0xf7,0xfb};//位码表
 while(1)
{
  P0=tab[0];              //将数字0的段码表送P0口
  P2= tab_bit[0];         //P2.7=0,第6位数码管显示0
  Delay(2);               //延时2ms
  P0=tab[5];              //将数字5的段码表送P0口
  P2= tab_bit[1];         //P2.6=0,第5位数码管显示5
  Delay(2);               //延时2ms
  P0=tab[0];              //将数字0段码表送P0口
  P2=tab_bit[2];          //P2.5=0,第4位数码管显示0
  Delay(2);               //延时2ms
  P0=tab[9];              //将数字9的段码表送P0口
  P2=tab_bit[3];          //P2.4=0,第3位数码管显示9
  Delay(2);               //延时2ms
  P0=tab[0];              //将数字0的段码表送P0口
  P2=tab_bit[4];          //P2.3=0,第2位数码管显示0
  Delay(2);               //延时2ms
  P0=tab[6];              //将数字6的段码表送P0口
  P2=tab_bit[5];          //P2.2=0,第1位数码管显示6
  Delay(2);               //延时2ms
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    }
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 disp_3. c。对程序进行编译、链接和调试，产生 disp_3. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_10/disp_3 文件夹中。

实验 4：用下载型实验板中的 LED 显示器显示“123456”。

本实验和实验 3 要求基本一致，只是显示的数字不同而已。下面采用编写显示函数的方法编写源程序，如下所示：

```
#include<reg51.h>
unsigned char code tab[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,
0x90};//段码表
unsigned char code tab_bit[]={0x7f,0xbf,0xdf,0xef,0xf7,0xfb};//位码表
//以下是延时函数
void Delay(unsigned int i)      //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)               //变量 i 由实际参数传入一个值,因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
}
//以下是显示函数
void display(unsigned char a, unsigned char num)   //a 为显示位数,num 为待显示的数字
    {
    switch(a)
      { case 6: P0=tab[num],P2= tab_bit[0];break;   //第 6 位显示数字 num
      case 5: P0=tab[num],P2= tab_bit[1]; break;    //第 5 位显示数字 num
      case 4: P0=tab[num],P2= tab_bit[2]; break;    //第 4 位显示数字 num
      case 3: P0=tab[num],P2= tab_bit[3]; break;    //第 3 位显示数字 num
      case 2: P0=tab[num],P2= tab_bit[4]; break;    //第 2 位显示数字 num
      case 1: P0=tab[num],P2= tab_bit[5]; break;    //第 1 位显示数字 num
      default: break;}
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
}
//以下是主程序
void main()
{
    while(1)
    {
    display(6,1);              //第 6 位显示数字 1
    Delay(2);                  //延时 2ms
    display(5,2);              //第 5 位显示数字 2
    Delay(2);                  //延时 2ms
    display(4,3);              //第 4 位显示数字 3
    Delay(2);                  //延时 2ms
    display(3,4);              //第 3 位显示数字 4
    Delay(2);                  //延时 2ms
    display(2,5);              //第 2 位显示数字 5
    Delay(2);                  //延时 2ms
    display(1,6);              //第 1 位显示数字 6
    Delay(2);                  //延时 2ms
  }
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 disp_4. c。对程序进行编译、链接和调试，产生 disp_4. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_10/disp_4 文件夹中。

实验 5：用下载型实验板中的 LED 显示器交替显示“123456”和“050906”，显示时间间隔为 1s。

根据要求，编写的源程序如下：

```
#include<reg51. h>
unsigned char code tab[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,
0x90};//段码表
unsigned char code tab_bit[]={0x7f,0xbf,0xdf,0xef,0xf7,0xfb};//位码表
//以下是延时函数
void Delay(unsigned int i)      //延时程序,i 是形式参数
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
{
    unsigned int j;
    for(;i>0;i--)              //变量 i 由实际参数传入一个值,因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
}
//以下是显示函数
void display(unsigned char a, unsigned char num)  //a 为显示位数,num 为待显示
的数字
    {
    switch(a)
      {
        case 6: P0=tab[num],P2= tab_bit[0];break;  //第 6 位显示数字 num
        case 5: P0=tab[num],P2= tab_bit[1]; break; //第 5 位显示数字 num
        case 4: P0=tab[num],P2= tab_bit[2]; break; //第 4 位显示数字 num
        case 3: P0=tab[num],P2= tab_bit[3]; break; //第 3 位显示数字 num
        case 2: P0=tab[num],P2= tab_bit[4]; break; //第 2 位显示数字 num
        case 1: P0=tab[num],P2= tab_bit[5]; break; //第 1 位显示数字 num
        default: break;
      }
    }
//以下是主程序
void main()
{
  while(1)
    {
      unsigned char i;
      for(i=0;i<=80;i++) //循环显示 80 次,约延时 2ms×6×80=960ms(约
                         1s)
      {display(6,1);     //第 6 位显示数字 1
      Delay(2);          //延时 2ms
      display(5,2);      //第 5 位显示数字 2
      Delay(2);          //延时 2ms
      display(4,3);      //第 4 位显示数字 3
      Delay(2);          //延时 2ms
      display(3,4);      //第 3 位显示数字 4
      Delay(2);          //延时 2ms
      display(2,5);      //第 2 位显示数字 5
      Delay(2);          //延时 2ms
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
        display(1,6);              //第 1 位显示数字 6
        Delay(2);}                 //延时 2ms
        for(i=0;i<=80;i++)         //循环显示 80 次,约延时 2ms×6×80=960ms(约 1s)
        {display(6,0);             //第 6 位显示数字 0
        Delay(2);                  //延时 2ms
        display(5,5);              //第 5 位显示数字 5
        Delay(2);                  //延时 2ms
        display(4,0);              //第 4 位显示数字 0
        Delay(2);                  //延时 2ms
        display(3,9);              //第 3 位显示数字 9
        Delay(2);                  //延时 2ms
        display(2,0);              //第 2 位显示数字 0
        Delay(2);                  //延时 2ms
        display(1,6);              //第 1 位显示数字 6
        Delay(2);}                 //延时 2ms
    }
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 disp_5. c。对程序进行编译、链接和调试,产生 disp_5. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target ',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_10/disp_5 文件夹中。

第二节　键 盘 接 口

键盘是单片机十分重要的输入设备,是实现人机对话的纽带。键盘是由一组规则排列的按键组成,一个按键实际上就是一个开关元件,即键盘是一组规则排列的开关。根据按键结构、原理的不同,它主要分为两类:一类是触点式开关按键,如机械开关式等;第二类是无触点开关按键,如磁感应按键等。前者造价低,后者寿命长。目前,单片机应用系统中最常见的是触点式开关按键。

一、键盘的工作原理

1. 键盘的特性

键盘是由一组按键开关组成的。通常,按键所用开关为机械弹性开关,这种开关一般

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

为常开型。平时(按键不按下时),按键的触点是断开状态,按键被按下时,它们才闭合。由于机械触点的弹性作用,一个按键开关从开始接上至接触稳定要经过一定的弹跳时间,即在这段时间里连续产生多个脉冲,在断开时也不会一下子断开,存在同样的问题,按键抖动信号波形如图 10-7 所示。

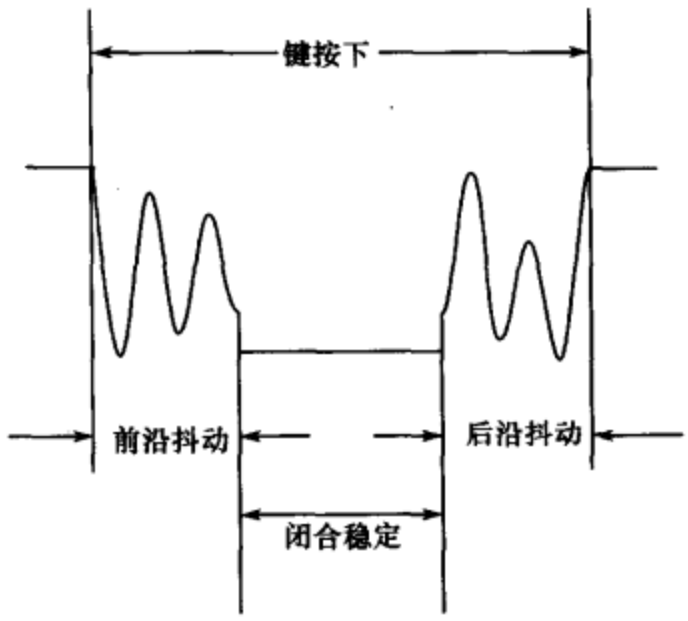


图 10-7　按键抖动信号波形

从波形图可以看出,按键开关在闭合及断开的瞬间,均伴随有一连串的抖动。抖动时间的长短由按键的机械特性决定,一般为 5ms～10ms,而按键的稳定闭合期的长短则是由操作人员的按键动作决定的,一般为十分之几秒的时间。

2. 按键的确认

按键的确认就是判别按键是否闭合,反映在电压上就是和按键相连的引脚呈现出高电平或低电平。如果高电平表示断开的话,那么低电平则表示闭合,所以通过检测电平的高低状态,便可确认按键是否按下。

3. 按键抖动的消除

因为机械开关存在抖动问题,为了确保 CPU 对一次按键动作只确认一次按键,必须消除抖动的影响。消除按键的抖动,通常有硬件、软件两种消除方法。在键数较少时,可用硬件消抖,而当键数较多时,采用软件消抖。

(1)硬件消抖

常用的硬件消抖有双稳态消抖和滤波消抖电路两种。

如图 10-8 是双稳态消抖电路。图中用两个与非门构成一个基本 RS 触发器。当按键未按下时,因 a=0,b=1,输出 Q 为 1;当按键按下时,因按键的机械性能,使按键因弹性抖动而产生瞬时不闭合(抖动跳开 b),当开关 K 没有稳定到达 b 时,因与非门 2 输出为 0 反馈到与非门 1 的输入端,封锁了与非门 1,双稳态电路的状态不会改变,输出保持为

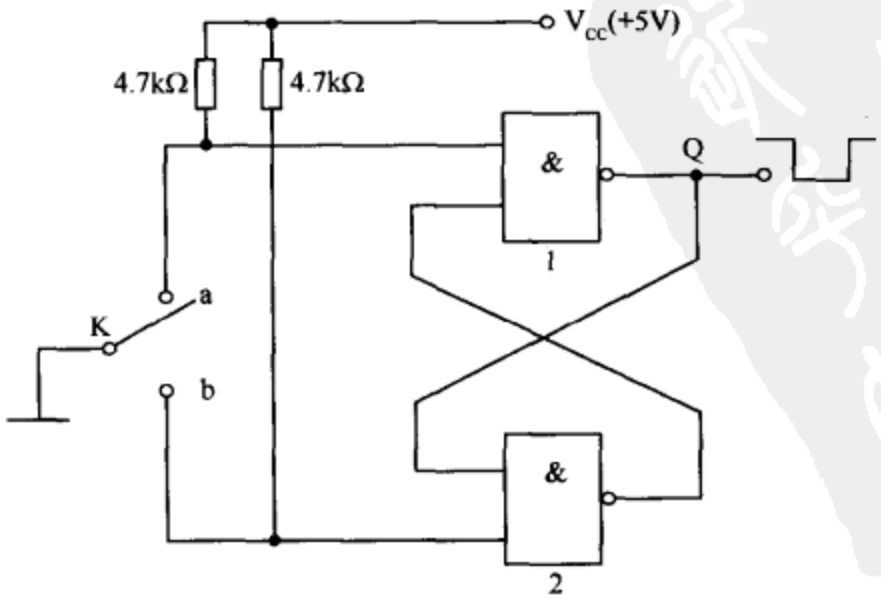


图 10-8　双稳态消抖电路

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

1，不会产生抖动的波形。当开关 K 稳定到达 b 时，因 a=1，b=0，从而使 Q=0，状态产生翻转。当松开开关 K，在开关未稳定达到 a 端时，因 Q=0，所以封锁了与非门 2，从而消除了后沿的抖动，使 Q=0 不变。只有当开关稳定地达到 a 端后，输出才重新返回到原状态。也就是说，即使开关输出的电压波形是抖动的，但经双稳态电路之后，其输出为正规的矩形方波，不会出现“毛刺”的现象。

如果利用 RC 积分电路，来吸收机械按键开关产生的干扰脉冲，则这种电路称为滤波消抖电路，如图 10-9 所示。当 K 未按下时，电容两端电压为 0，与非门输出为 1。当 K 按下时，由于 C 两端电压不能突变，即使在接触过程中出现抖动，只要适当选取 R1、R2 和 C 的值，使 C 两端的充电电压波动不超过门的开启电压（TTL 为 0.8V 左右），门的输出将不会改变；同样，K 在断开过程中，即使出现抖动，由于 C 两端电压不能突变，而要经过 R2 放电，只要 C 两端的放电电压波动不超过门的关闭电压，门的输出也不会改变。所以，从上面分析可看出，只要正确选取 R1、R2 和 C 构成的时间常数，就能保证 C 由稳态电压充电到开启电压或放电到关闭电压的延迟时间大于或等于 10ms，达到消除按键抖动的目的。

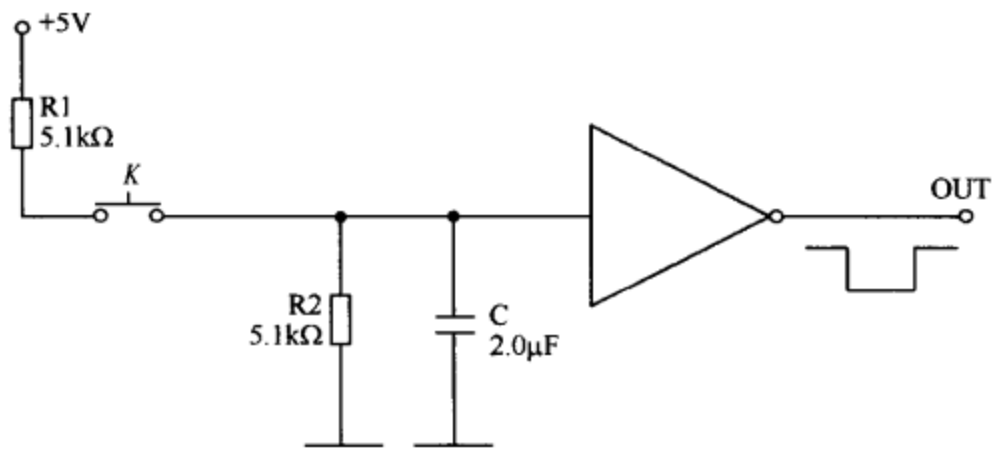


图 10-9　滤波消除抖动电路

(2)软件消抖

单片机应用系统中，常用软件方法来消除抖动，即检测出键闭合后执行一个延时程序，产生 5ms～10ms 的延时，以避开按键按下去的抖动时间，待信号稳定之后再进行键查询，如果仍保持闭合状态电平，则确认为真正有键按下。一般情况下，不对按键释放的后沿进行处理。

二、键盘与单片机的连接

单片机中的键盘可分为独立式、编码式、串行口扩展式和矩阵式（行列式）等几类。

1. 独立式按键

独立式按键就是各按键相互独立、每个按键各接一根输入线，一根输入线上的按键是否按下不会影响其他输入线上的工作状态。因此，通过检测输入线的电平状态，可以很容易判断哪个按键被按下了。独立式按键电路配置灵活，软件结构简单。但每个按键需占用一根输入口线，在按键数量较多时，输入口浪费大，电路结构显得很繁杂，故此种键盘适用于按键较少或操作速度较高的场合。下面介绍几种独立式按键的接口。

图 10-10 是直接与单片机的 I/O 口线相接的独立式按键接口工作电路，通过读 I/O 口，判定各 D0 口线的电平状态，即可识别出按下的按键。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

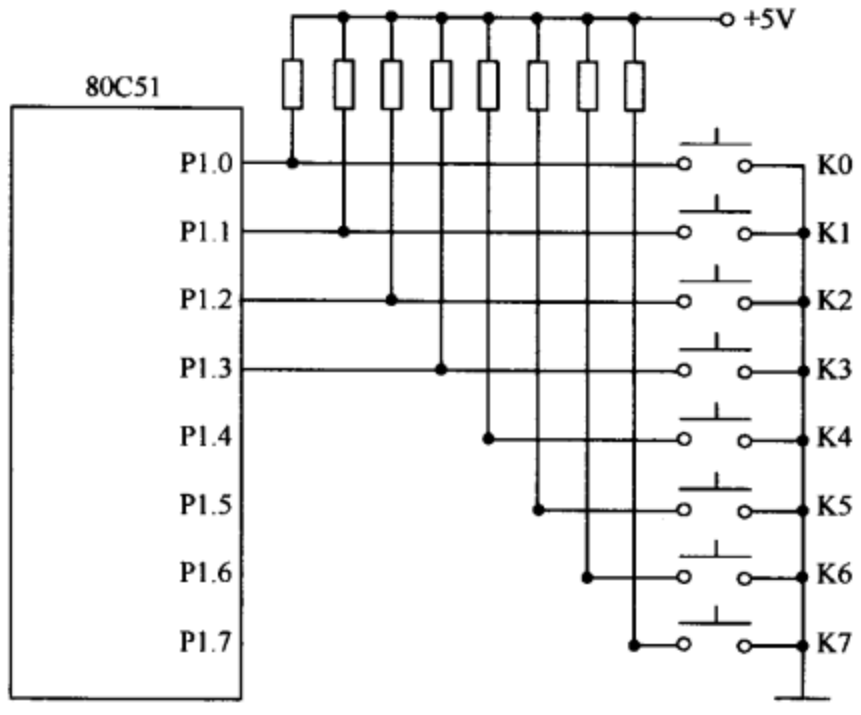


图 10-10　独立式按键

上述独立式按键电路中，各按键开关均采用了上拉电阻，这是为了保证在按键断开时，各 I/O 口有确定的高电平，当然，如果输入口线内部已有上拉电阻，则外电路的上拉电阻可省去。采用查询式方式编程的方法是：先逐位查询每条 I/O 口线的输入状态，如某一条 I/O 口线输入为低电平，则可确认该 I/O 口线所对应的按键已按下，然后，再转向该键的功能处理程序。

2. 编码式按键

采用编码式按键接口可提高单片机 I/O 口线的利用率。当按键较多，I/O 口线不够用时，可采用这种接口方式。图 10-11 是采用 74LS148 编码式按键接口电路。

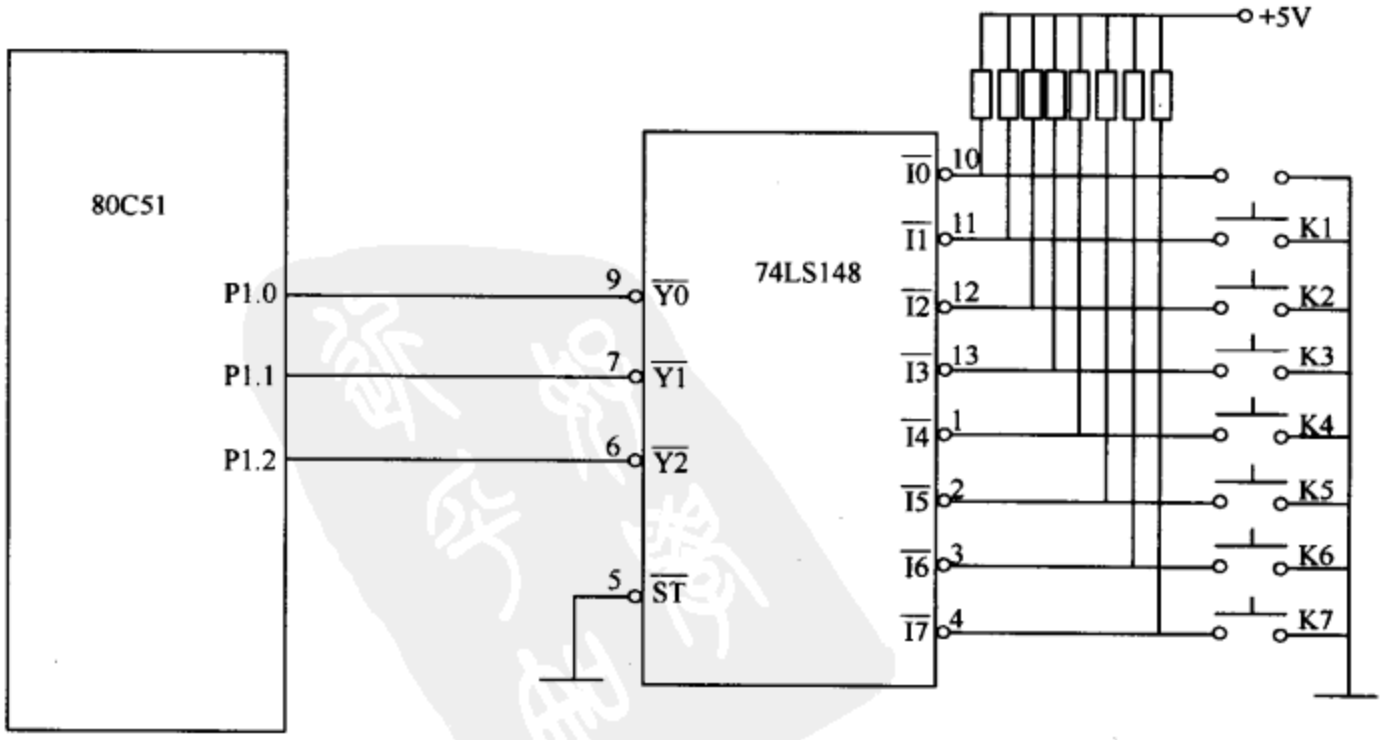


图 10-11　采用 74LS148 编码式按键接口电路

74LS148 是将 8 条数据线编码为 3 条数据线的 8—3 线优先编码器。它把占有 8 条线的按键状态变为用 3 位二进制方式进行传送。74LS148 的引脚功能如图 10-12 所示。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

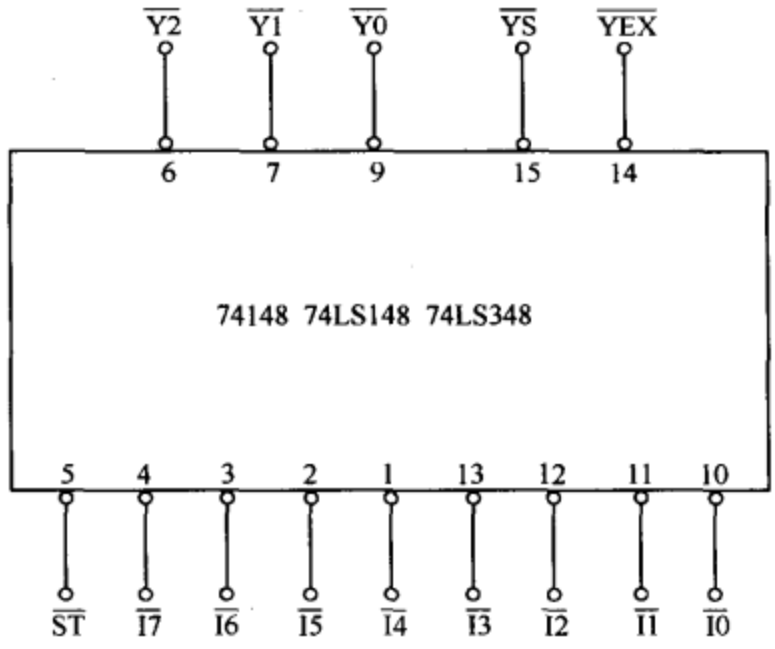


图 10-12　74LS148 的引脚功能图

74LS148 的$\overline{I0}$～$\overline{I7}$为输入脚，低电平有效；$\overline{Y2}$、$\overline{Y1}$、$\overline{Y0}$为输出脚，共可输出 8 组二进制码，亦为低电平有效。输入$\overline{I0}$～$\overline{I7}$中，优先权排列顺序为$\overline{I7}$(最高)……$\overline{I0}$(最低)。例如，当器件工作时，若$\overline{I7}=0$($\overline{I7}$权限最高)，则不论$\overline{I6}$～$\overline{I0}$的状态如何，输出$\overline{Y2Y1Y0}=000$；当$\overline{I7}=1$时，则观察$\overline{I6}$是否为有效信号(即是否为“0”)，若$\overline{I6}=0$，那么输出$\overline{Y2Y1Y0}=001$，而不论$\overline{I5}$～$\overline{I0}$的状态如何……74LS148 编码器的真值如表 10-4 所列。

表 10-4　74LS148 编码器真值表

输　入									输　出				
$\overline{ST}$	$\overline{I7}$	$\overline{I6}$	$\overline{I5}$	$\overline{I4}$	$\overline{I3}$	$\overline{I2}$	$\overline{I1}$	$\overline{I0}$	$\overline{Y2}$	$\overline{Y1}$	$\overline{Y0}$	$\overline{Y_{EX}}$	Y_S
1	×	×	×	×	×	×	×	×	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	×	×	×	×	×	×	×	0	0	0	0	1
0	1	0	×	×	×	×	×	×	0	0	1	0	1
0	1	1	0	×	×	×	×	×	0	1	0	0	1
0	1	1	1	0	×	×	×	×	0	1	1	0	1
0	1	1	1	1	0	×	×	×	1	0	0	0	1
0	1	1	1	1	1	0	×	×	1	0	1	0	1
0	1	1	1	1	1	1	0	×	1	1	0	0	1
0	1	1	1	1	1	1	1	0	1	1	1	0	1

74LS148 除了 8 根输入线和 3 根输出线外，还有 3 根信号线。其中$\overline{ST}$是输入允许控制端，称为输入使能端，当$\overline{ST}=0$ 时编码器工作，$\overline{ST}=1$ 时编码器不工作，所有输出均为 1。Y_S及$\overline{Y_{EX}}$为输出信号，用以指示输入信号的情况，当器件不工作($\overline{ST}=1$)时，$Y_S=1$，$\overline{Y_{EX}}=1$；当器件工作($\overline{ST}=0$)时，若$\overline{I0}$～$\overline{I7}$中有无信号输入，则 $Y_S=1$，否则 $Y_S=0$。

3. 串行口扩展式按键

80C51 单片机应用系统中，如果并行 I/O 口不够，而串行口又没有其他用处时，则可用来扩展并行 I/O 口，从而节省单片机的硬件资源。80C51 单片机内部的串行口在工作方式 0 状态下，使用移位寄存器芯片可以扩展一个或多个 8 位并行 I/O 口，下面以

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

74LS165 为例，通过串行口来扩展键盘电路，有关电路如图 10-13 所示。

8 位单向移位寄存器 74LS165 为并行(串行)输入/串行输出寄存器，它能在一个信号的控制下并行置入一个字节的数据，然后在时钟脉冲的作用下逐位移出，也能使数据从另外一个引脚串行输入。74LS165 引脚功能如图 10-14 所示。

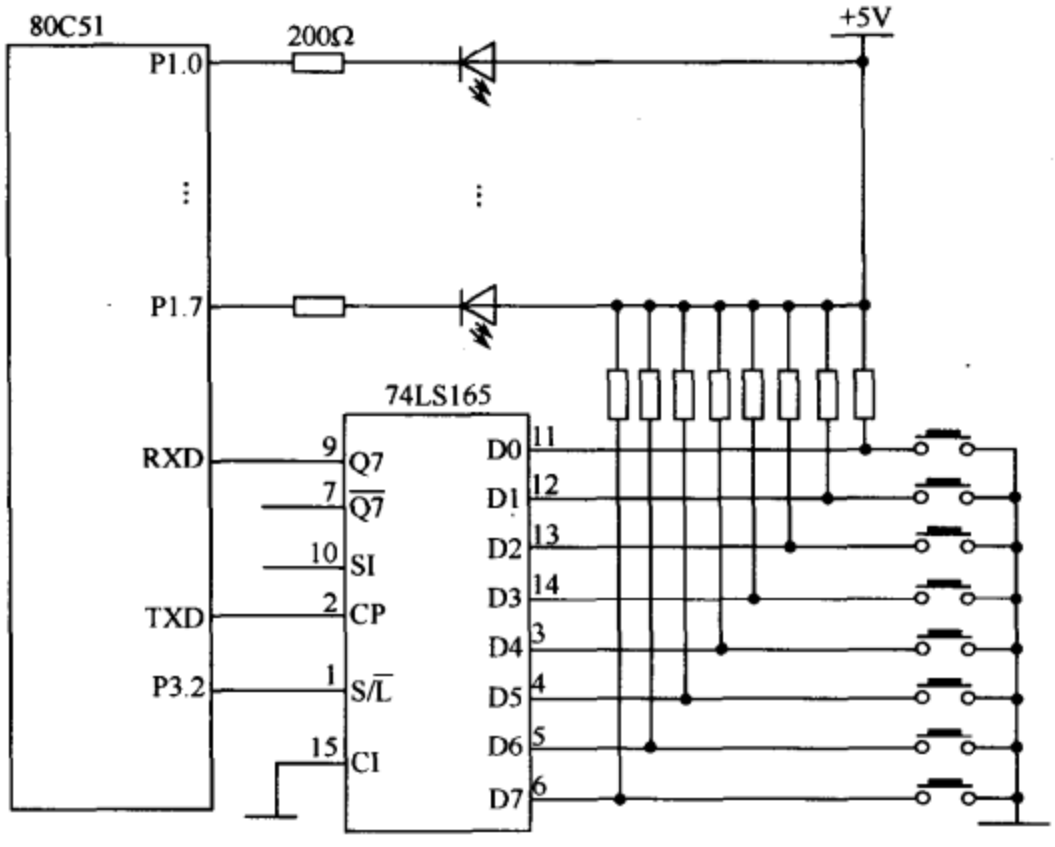


图 10-13　串行口扩展键盘电路

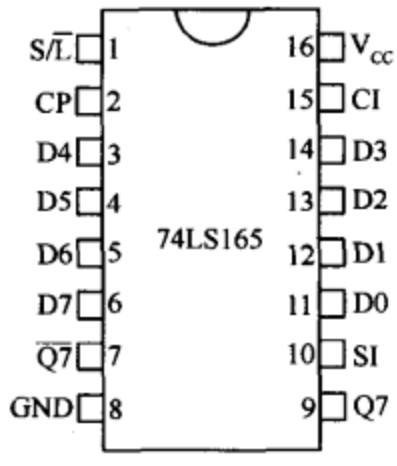


图 10-14　并行输入/串行输出移位寄存器 74LS165 管脚功能

图中，D0～D7 为一个字节的并行数据输入端。S/$\overline{L}$(Shift/Load)：控制信号输入端，该引脚为高电平时具有移位功能，为低电平时，将 D0～D7 端的数据输入到内部保存。CP 为时钟信号(即移位脉冲)输入端，当 S/$\overline{L}$ 端为高电平时，CP 端的每一次正跳变，都会使已收入内部的数据(D0～D7)从 Q7 端移出一位，移位的顺序是 D7 最先从 Q7 端移出，D0 最后从 Q7 端移出。CI(Clock Inhibit)为时钟脉冲禁止端，当该引脚为高电平时，时钟信号(移位脉冲)不能进入，因而也就不可能移位，正常工作时必须接低电平。SI(Serial Input)为串行数据输入端，该器件也能从该端接收串行数据并从 Q7 端移出。

8 位单向移位寄存器 74LS165 状态如表 10-5 所列。

表 10-5　8 位单向移位寄存器 74LS165 状态表

S/$\overline{L}$	CI	CP	SI	D0～D7	$Q0^{n+1}$	$Q1^{n+1}$	$Q7^{n+1}$	说明
0	×	×	×	D0～D7	D0	D1	D7	并行输入
1	0	0	×	×	$Q0^n$	$Q1^n$	$Q7^n$	保持
1	0	↑	1	×	1	$Q0^n$	$Q6^n$	输入一个 1
1	0	↑	0	×	0	$Q0^n$	$Q6^n$	输入一个 0
1	1	×	×	×	$Q0^n$	$Q1^n$	$Q7^n$	保持

图 10-13 所示的电路中，TXD(P3.1)为移位脉冲输出端，与 74LS165 的移位脉冲输

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

入端 CP 相连；RXD(P3.0)作为串行输入端，与 74LS165 的串行输出端 Q7 相连；P3.2 用来控制 74LS165 的移位与置入。

4. 矩阵式(行列式)按键

为了减少键盘与单片机接口时所占用 I/O 线的数目，在按键数较多时，通常将键盘排列成行列矩阵形式，如图 10-15 是一种 3 行×4 列的矩阵式键盘电路。

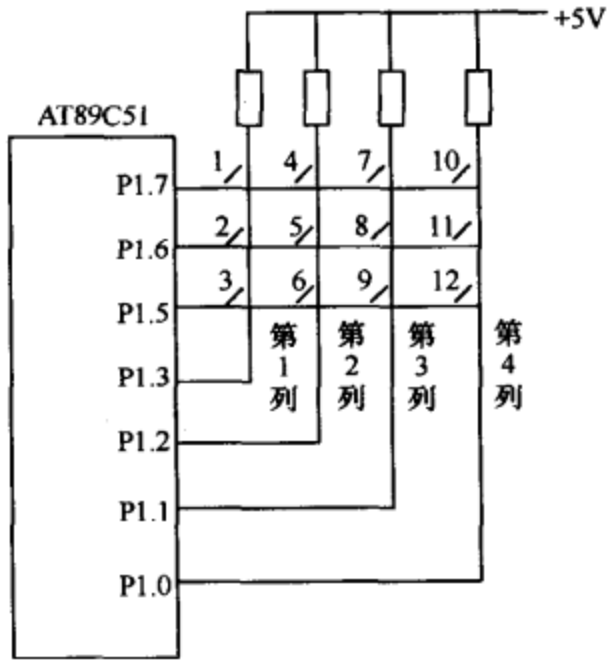


图 10-15　矩阵式键盘电路

键盘的列线一端经电阻接＋5V 电源，另一端接单片机的输入口；各行线的一端接单片机的输出口，另一端悬空。每一行线(水平线)与列线(垂直线)的交叉处不相通，而是通过一个按键来连通。利用这种矩阵结构，只需 n 条行线和 m 条列线，即可组成具有 $n\times m$ 个按键的键盘。由此可见，在需要的键位数比较多时，一般需要采用矩阵式键盘。

①判断有无键被按下。键被按下时，与此键相连的行线与列线将导通，而列线电平在无键按下时处于高电平。显然，如果让所有行线处于高电平，那么键按下与否都不会引起列线电平的状态变化，所以只有让所有行线处于低电平，当有键按下时按键所在列电平将被拉成低电平，根据此列电平的变化，便能判定此列一定有键被按下。

矩阵式结构的键盘比独立式的要复杂一些，判断有无键被按下也要复杂一些。图 10-15中，列线通过电阻接正电源，并将列线所接的单片机的 I/O 口作为输入端；而行线所接的 I/O 口则作为输出端。为了判断有无键被按下，可先经输出口向所有行线输出低电平，然后再输入各列线状态。若列线状态皆为高电平，则表明无键按下；若列线状态中有低电平，则表明有键被按下。

②判断按键是否真的被按下。当判断出有键被按下之后，用软件延时的方法延时 10ms，再判断键盘的状态，如果仍认为有键被按下，则认为确实有键按下，否则，当作键抖动处理。

③判断哪一个按键被按下。当判断出哪一列中有键被按下时，可根据 P1 口的数值来确定哪一个键被按下。

④等待键释放。键释放之后，可以根据键码转相应的键处理子程序，进行数据的输入或命令的处理。

例：编写 3 行×4 列按键程序，并用 LED 显示器显示出来。

实验源程序如下：

```
#include<reg51.h>      //若本行改为#include<AT89X51.h>，可省略以下几行
                       sbit 语句
sbit P1_3=P1^3;        //定义列线
sbit P1_2=P1^2;
sbit P1_1=P1^1;
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
sbit P1_0=P1^0;
sbit P2_1=P2^0;          //定义第 1 位 LED 显示器位码线
unsigned char code table[]=
  {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,
  0x90,0x88,0x83,0xA7,0xA1,0x86,0x8E};   //0～F 显示段码表
void KeyScan();                          //键盘扫描函数
void Delay10ms(unsigned char time);      //10ms 延时函数
void Dispaly(unsigned char k);           //显示屏显示函数
unsigned char key,temp;                  //定义变量
//以下是按键扫描函数(子程序)
void KeyScan()
{
  P1=0xFF;                   //P1 口置高电平
  P1_3=0;                    //第 1 列置低电平
  temp=P1;                   //将 P1 暂存于 temp
  temp&=0xF0;                //temp 与 0xF0 相与后得到行线状态送 temp
  if(temp ! =0xF0)           //若行线状态 temp 不等于 0xf0,说明第 1 列有键
                             按下
  {
    Delay10ms(1);            //延时 10ms
    temp=P1;                 //将 P1 送 temp
    temp&=0xF0;              //temp 与 0xF0 相与后得到行线状态,再送到
                             temp
    if(temp ! =0xF0)         //若行线状态 temp 不等 0xf0,说明第 1 列的确有
                             键按下
    {                        //若第 1 列有键按下,再判断是第 1 列的哪一个
                             键被按下
      temp=P1;               //将 P1 送 temp
      temp&=0xF0;            //temp 与 0xF0 相与后得到行线状态送 temp
      switch(temp)           //求行状态的值
      {case 0x70:key=1;break; //temp=0x70,说明 P1_7=0,即第 1 号键被按下
      case 0xB0:key=2;break;  //temp=0xB0,说明 P1_6=0,即第 2 号键被按下
      case 0xD0:key=3;break;} //temp=0xD0,说明 P1_5=0,即第 3 号键被按下
      Dispaly(key);          //调显示子程序,显示键值
    }
  }                          //若行线状态 temp 等于 0xf0,说明第 1 列无键按
                             下,再判断第 2 列
  P1=0xFF;                   //P1 口置高电平
  P1_2=0;                    //第 2 列置低电平
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
temp=P1;
temp&=0xF0;                    //取出行状态 temp
if(temp! =0xF0)                //若行状态 temp 不等于 0xf0,说明第 2 列有键被
                               按下
{
  Delay10ms(1);                //延时 10ms
  temp=P1;
  temp&=0xF0;                  //再取出行状态 temp
  if(temp ! =0xF0)             //若 temp 不等于 0xf0,说明第 2 列的确有键被按
                               下
  {
    temp=P1;
    temp&=0xF0;                //取出行状态
    switch(temp)               //求行状态的值
    {case 0x70:key=4;break;    //temp=0x70,说明 P1_7=0,即第 4 号键被按下
    case 0xB0:key=5;break;     //temp=0xB0,说明 P1_6=0,即第 5 号键被按下
    case 0xD0:key=6;break;}    //temp=0xD0,说明 P1_5=0,即第 6 号键被按下
    Dispaly(key);              //显示键值
  }
}
```

```
P1=0xFF;
P1_1=0;                        //第 3 列置低电平
temp=P1;
temp&=0xF0;
if(temp ! =0xF0)
{
  Delay10ms(1);
  temp=P1;
  temp&=0xF0;
  if(temp ! =0xF0)
  {
    temp=P1;
    temp&=0xF0;
    switch(temp)
    switch(temp)               //求行状态的值
    {case 0x70:key=7;break;    //temp=0x70,说明 P1_7=0,即第 7 号键被按下
    case 0xB0:key=8;break;     // temp=0xB0,说明 P1_6=0,即第 8 号键被按下
    case 0xD0:key=9;break;}    //temp=0xD0,说明 P1_5=0,即第 9 号键被按下
  Dispaly(key);                //调显示子程序,显示键值
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    }
}
P1=0xFF;
P1_0=0;                          //第 4 列置低电平
temp=P1;
temp&=0xF0;
if(temp ! =0xF0)
{
   Delay10ms(1);
   temp=P1;
   temp&=0xF0;
   if(temp ! =0xF0)
   {
      temp=P1;
      temp&=0xF0;
      switch(temp)
      switch(temp)               //求行状态的值
      {case 0x70:key=10;break;  //temp=0x70,说明 P1_7=0,即第 10 号键被按下
      case 0xB0:key=11;break;   //temp=0xB0,说明 P1_6=0,即第 11 号键被按下
      case 0xD0:key=12;break;}  //temp=0xD0,说明 P1_5=0,即第 12 号键被按下
         Dispaly(key);          //调显示子程序,显示键值
      }
   }
}
//以下是 10ms 延时函数(子程序)
void Delay10ms(unsigned char time)
{
   unsigned char a,b,c;
   for(a=0;a<time;a++)
   for(b=0;b<10;b++)
   for(c=0;c<120;c++)
   ;
}
//以下是显示函数(子程序)
void Dispaly(unsigned char k)
{
   P0=table[k];                  //将显示数据送 P0 口
   P2_1=0;                       //位码 P2_1 置 0,让第 1 位 LED 显示器显示键值
}
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

以下是主程序

```
void main()
{
while(1)
  {
  KeyScan();                    //调键盘扫描函数
  }
}
```

三、键盘的工作方式

键盘的工作方式有 3 种，即程序控制扫描、定时扫描和中断扫描方式。

1. 程序控制扫描方式

程序控制扫描方式是指单片机在空闲时，才调用键盘扫描子程序，并反复地扫描键盘，直到用户从键盘上输入命令或数据，而在执行键入命令或处理键入数据过程中，CPU 将不再响应键入要求，直到 CPU 重新扫描键盘为止。它主要包含以下内容：

①判别有无键按下。

②键扫描取得闭合键的行、列值。

③计算键值。

④判断闭合键是否释放，如果没有释放则等待；如果闭合键释放，将闭合键的键号保存，并转去执行该闭合键的功能。

前面介绍的几种按键程序均采用了这种扫描方式。

2. 定时扫描方式

定时扫描方式就是每隔一定时间对键盘扫描一次，它利用单片机内部的定时器产生一定时间(例如 10ms)的定时，当定时时间到就产生定时器溢出中断，CPU 响应中断后对键盘进行扫描，并在有键按下时识别出该键执行响应的键功能程序。

3. 中断扫描方式

键盘工作在程序控制扫描方式时，当无键按下时 CPU 要不间断地扫描键盘，直到有键按下为止。如果 CPU 要处理的事情很多，这种工作方式将不能适应。定时扫描方式只要定时时间到，CPU 就去扫描键盘，工作效率有了进一步的提高。但这两种方式常使 CPU 处于空扫状态，而中断扫描方式下，CPU 可以一直处理自己的工作，直到有键闭合时发出中断申请，CPU 响应中断，执行相应的中断服务程序，才对键盘进行处理，从而提高了 CPU 的工作效率。图 10-16 为中断扫描方式键盘接口电路。

该键盘由单片机的 P1 口构成 4×4 矩阵键盘。键盘的列线和 P1 口的高 4 位 P1.4～P1.7 相接，键盘的行线和 P1 口的低 4 位 P1.0～P1.3 相接。P1.4～P1.7 经过与门和单片机的外中断$\overline{\text{INT0}}$相接。当有键按下时，$\overline{\text{INT0}}$变为低电平，向 CPU 发出中断申请，若 CPU 开放外部中断 0，则 CPU 响应外部中断请求，执行相应的中断服务程序。为了在中断服务中不因按键的识别错误再引起中断，因此，在中断服务程序中，首先应关闭中断，再进行键扫描、消抖动、识别按错误键等工作，具体处理方法同程序控制扫描方式。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

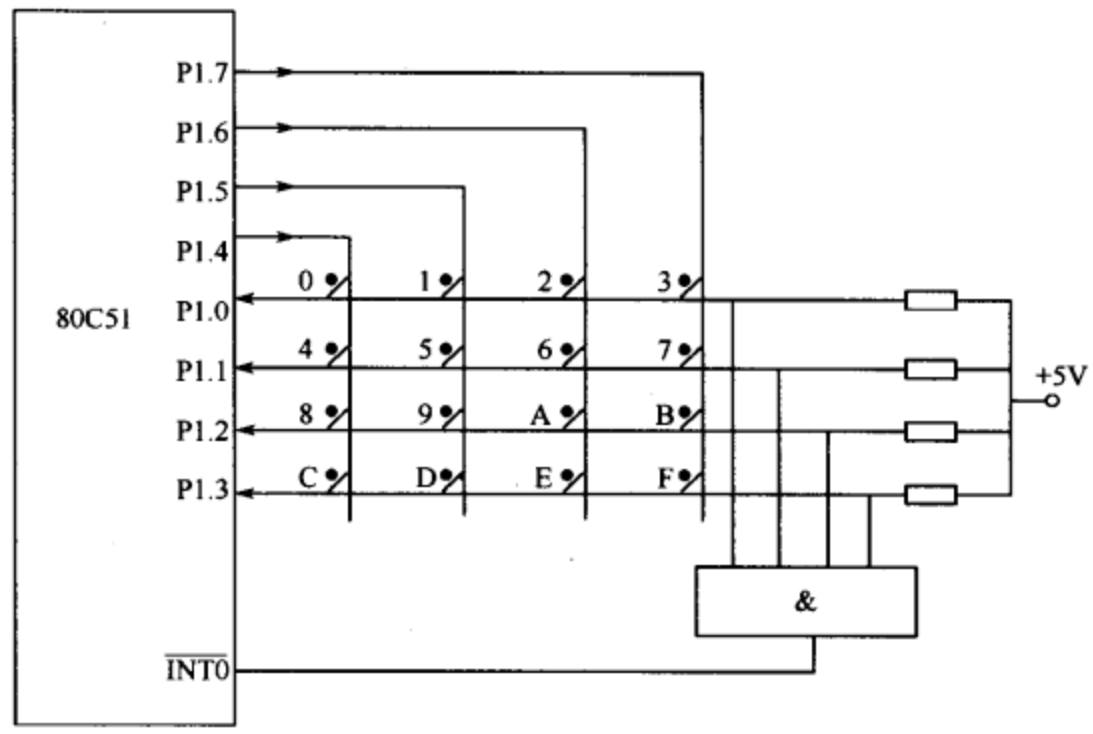


图 10-16　中断扫描方式键盘接口电路

四、实验演练

实验:用下载型实验板做以下实验:按 K1(P3.2),松开后 P1 口 LED 闪 1 次;按 K2(P3.3),松开后 LED 闪 2 次;按 K3(P3.4),松开后 LED 闪 3 次;按 K4(P3.5),松开后 LED 闪 4 次。

实验源程序如下:

```
#include<AT89X51.h>
void main()
{
void flashLed(unsigned char cnt);    //闪烁函数
void Delay(unsigned char t);         //延时函数
flashLed(2);                         //开机复位后,LED 先闪两次,表明单片机正常
start:
while(1)
{
if(P3_2==0)                          //若开关 K1 按下
  {Delay(50);                        //延时
   while(P3_2)                       //若开关 K1 抬起,P3_2=1,则往下执行,否
                                     则,等待松开
   {flashLed(1);                     //闪烁一次
   goto start;}}                     //跳出去,否则会一直闪烁
  if(P3_3==0)                        //若开关 K2 被按下
   {Delay(50);                       //延时
   while(P3_3)                       //等待松开
   {flashLed(2);                     //闪烁两次
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
        goto start;}}                    //跳出去,否则会一直闪烁
        if(P3_4==0)                      //若 K3 被按下
          {Delay(50);
          while(P3_4)                    //等待松开
          {flashLed(3);                  //闪烁三次
           goto start;}}                 //跳出去,否则会一直闪烁
        if(P3_5==0)                      //若 K4 被按下
          {Delay(50);
           while(P3_5)                   //等待松开
           {flashLed(4);                 //闪烁四次
           goto start;}}                 //跳出去,否则会一直闪烁
      }
}
void flashLed(unsigned char cnt)         //闪烁子程序
{
  unsigned char a;
  for(a=1;a<=cnt;a++)
  {
   P1=0x00;                              //LED 点亮
   Delay(100);
   P1=0xff;                              //LED 熄灭
   Delay(100);
  }
}
void Delay(unsigned char t)              //延时子程序
{
  unsigned char i,j,k;
  for(i=0;i<=t;i++)
  for(j=0;j<=10;j++)
  for(k=0;k<=120;k++)
 ;
}
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 key. c。对程序进行编译、链接和调试,产生 key. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target ',在出现的窗口中设置为硬件仿真。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_10/key 文件夹中。

第三节　LCD 显示器接口

液晶显示器(LCD)由于体积和质量小、功耗低等优点,应用十分广泛,如电子表、传真机、复印机以及电脑中。

从液晶显示器的显示内容来分,可分为段式、字符式和点阵式 3 种。其中字符式液晶显示器以其价廉、显示内容丰富、美观、无须定制、使用方便等特点,成为 LED 的理想替代品。一般初学者由字符型 LCD 入手比较简单,学完之后,再进一步控制图案型 LCD 模块就十分方便。

一、字符型液晶显示器概述

字符型 LCD 专门用于显示数字、字母、图形符号及少量自定义符号。这类显示器均把液晶显示控制器、驱动器、字符存储器等做在一块板上,再与液晶屏一起组成一个显示模块,称为 LCM,因此,这类显示器安装与使用都十分简单和方便。

字符型 LCD 是由若干个 5×7 或 5×11 等点阵字符位组成。每一个点阵字符位都可以显示一个字符。目前市面上常用的有 16 字×1 行、16 字×2 行、20 字×2 行和 40 字×2 行等的字符模块组,它们虽然显示的字数各不相同,但输入/输出接口都相同。

图 10-17 是 16 字×2 行(下称 16×2)液晶显示模块的外形,其接口引脚有 16 只,管脚功能如表 10-6 所列。

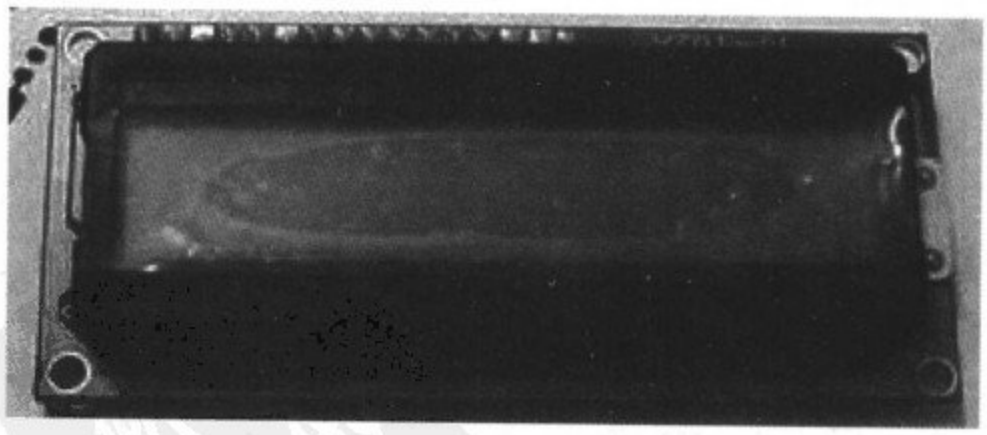


图 10-17　16×2 液晶显示模块外形

表 10-6　字符型液晶显示接口功能

编号	符号	功　能	编号	符号	功　能
1	V_{SS}	电源地	9	D2	数据 2
2	V_{DD}	电源正极	10	D3	数据 3
3	VLCD	液晶显示偏压信号	11	D4	数据 4
4	RS	数据/命令选择	12	D5	数据 5
5	R/W	读/写选择	13	D6	数据 6
6	E	使能信号	14	D7	数据 7
7	D0	数据 0	15	BLA	背光源正极
8	D1	数据 1	16	BLK	背光源负极

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

表 10-6 中，V_{SS}为电源地，V_{DD}接 5V 正电源，VLCD 为液晶显示器对比度调整端，接正电源时对比度最弱，接地时对比度最高。对比度过高时会产生“鬼影”，使用时可以通过一个 10kΩ 的电位器调整对比度。RS 为寄存器选择，高电平时选择数据寄存器，低电平时选择指令寄存器。R/W 为读写信号线，高电平时进行读操作，低电平时进行写操作。E 端为使能端，当 E 端由高电平跳变成低电平时，液晶模块执行命令。D0～D7 为 8 位双向数据线。BLA、BLK 用于带背光的模块，不带背光的模块这两个管脚悬空不接。

二、字符显示模块内部结构

目前大多数字符显示模块的控制器都采用型号为 HD44780 的集成电路作控制器。HD44780 是集控制器、驱动器于一体，专用于字符显示控制驱动集成电路。HD44780 是字符型液晶显示控制器的代表电路。

1. HD44780 集成电路的特点

①HD44780 不仅可作为控制器，而且还具有驱动 40×16 点阵液晶像素的能力，并且 HD44780 的驱动能力可通过外接驱动器扩展 360 列驱动。

②HD44780 的显示缓冲区及用户自定义的字符发生器 CGRAM 全部内藏在芯片内。

③HD44780 接口数据传输可为 8 位数据和 4 位数据传输两种方式。

④HD44780 具有简单而功能较强的指令集，可实现字符移动、闪烁等显示功能。

2. HD44780 的工作原理

HD44780 的内部电路如图 10-18 所示。

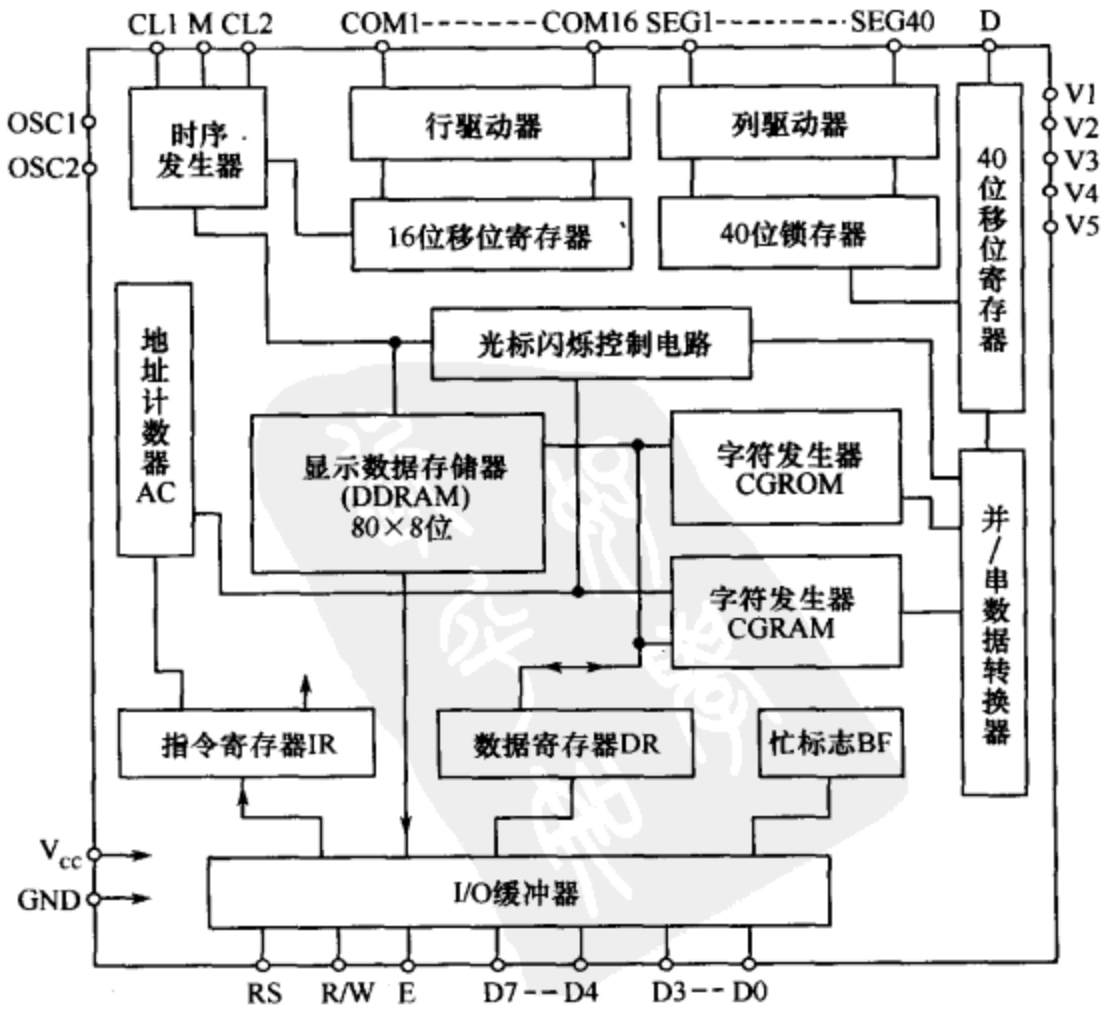


图 10-18　HD44780 的内部电路

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

(1)显示数据存储器 DDRAM

DDRAM 用来存放要 LCD 显示的数据，只要将标准的 ASCII 码送入 DDRAM，内部控制电路会自动将数据传送到显示器上，例如要 LCD 显示字符 A，则只须将 ASCII 码 41H 存入 DDRAM 即可。DDRAM 有 80 字节空间，共可显示 80 个字(每个字为 1 个字节)，其存储器地址和实际显示位置的排列顺序的对应关系如图 10-19 所示。

	0	1	2	3	……	12	13	14	15	-->显示位置
第一行-->	00	01	02	03	…	0C	0D	0E	0F	-->DDRAM地址

(*a*)

	0	1	2	3	……	15	16	17	18	19	-->显示位置
第一行-->	00	01	02	03	…	0F	10	11	12	13	-->DDRAM地址
第二行-->	40	41	42	43	…	4F	50	51	52	53	-->DDRAM地址

(*b*)

	0	1	2	3	…	15	16	17	18	19	-->显示位置
第一行-->	00	01	02	03	…	0F	10	11	12	13	-->DDRAM地址
第二行-->	40	41	42	43	…	4F	50	51	52	53	-->DDRAM地址
第三行-->	14	15	16	17	…	23	23	25	26	27	-->DDRAM地址
第四行-->	54	55	56	57	…	63	64	65	66	67	-->DDRAM地址

(*c*)

图 10-19　存储器地址与实际显示位置的排列顺序对应关系

(*a*)16(字)×1(行)显示地址；(*b*)20(字)×2(行)显示地址；(*c*)20(字)×4(行)显示地址。

图 10-19(*a*)为 16 字×1 行的 LCM，它的地址从 00H 到 0FH；图 10-19(*b*)为 20 字×2 行的 LCM，第一行的地址从 00H 到 13H，第二行的地址从 40H 到 53H；图 10-19(*c*)为 20 字×4 行的 LCM，第一行的地址从 00H 到 13H，第二行的地址从 40H 到 53H，第三行的地址从 14H 到 27H，第四行的地址从 54H 到 67H。

(2)字符发生器 CGROM

HD44780 内藏的字符发生存储器(CGROM)，存储了 160 个不同的点阵字符图形，如表 10-7 所列。这些字符有：阿拉伯数字、英文字母的大小写、常用的符号和日文假名等，每一个字符都有一个固定的代码。例如字符码 41H 为 A 字符，若要在 LCD 中显示 A，就是将 A 的代码 41H 写入 DDRAM 中，同时电路到 CGROM 中将 A 的字形点阵数据找出来，显示在 LCD 上，这样就能看到字母 A。

(3)字符发生器 CGRAM

CGRAM 是供使用者储存自行设计的特殊造型的造型码 RAM，CGRAM 共有 512bit (64 字节)。一个 5×7 点矩阵字形占用 8×8bit，所以 CGRAM 最多可存 8 个造型。

(4)指令寄存器 IR

指令寄存器 IR 负责储存单片机要写给 LCM 的指令码。当单片机要发送一个命令到 IR 时，必须要控制 LCM 的 RS、R/W 及 E 这三个引脚，当 RS 及 R/W 引脚信号为 0，E 引脚信号由 1 变为 0 时，就会把在 D0～D7 引脚上的数据送入 IR。

(5)数据寄存器 DR

数据寄存器 DR 负责储存单片机要写到 CGRAM 或 DDRAM 的数据，或储存单片机要从 CGRAM 或 DDRAM 读出的数据，因此 DR 可视为一个数据缓冲区，它也是由 LCM

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

表 10-7　内藏字符发生器存储的字符

高4位 / 低4位	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)															
xxxx0001	(2)															
xxxx0010	(3)															
xxxx0011	(4)															
xxxx0100	(5)															
xxxx0101	(6)															
xxxx0110	(7)															
xxxx0111	(8)															
xxxx1000	(1)															
xxxx1001	(2)															
xxxx1010	(3)															
xxxx1011	(4)															
xxxx1100	(5)															
xxxx1101	(6)															
xxxx1110	(7)															
xxxx1111	(8)															

的 RS、R/W 及 E 3 个引脚来控制。当 RS 及 R/W 引脚信号为 1，E 引脚信号为 1 时，LCM 会将 DR 内的数据由 D0～D7 输出，以供单片机读取；当 RS 引脚信号为 1，R/W 引脚信号为 0，E 引脚信号由 1 变为 0 时，就会把在 D0～D7 引脚上的数据存入 DR。

(6)忙碌标志信号 BF(忙标志)

BF 的功能是告诉单片机，LCM 内部是否正忙着处理数据。当 BF＝1 时，表示 LCM 内部正在处理数据，不能接受单片机送来的指令或数据。LCM 设置 BF 的原因为单片机处理一个指令的时间很短，只需几微秒左右，而 LCM 需要 40μs～1.64ms 的时间，所以单片要机要写数据或指令到 LCM 之前，必须先查看 BF 是否为 0。

(7)地址计数器 AC

AC 的工作是负责计数写到 CGRAM、DORAM 数据的地址，或从 DDRAM、CGRAM

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

读出数据的地址。使用地址设定指令写到 IR 后，则地址数据会经过指令解码器，再存入 AC。当单片机从 DDRAM 或 CGRAM 存取资料时，AC 依照单片机对 LCM 的操作而自动修改它的地址计数值。

方法技巧 HD44780 的特点是读写操作由使能信号 E 完成，不操作时为低电平，操作时产生一个正脉冲，在读信号且 E 为高电平时，控制器将所需的数据送入数据总线上，供单片机读取；在写操作时，E 信号的下降沿处将数据总线上的数据写入控制器接口部的寄存器内。HD44780 对读/写操作的识别是判断 R/W 信号端上的电平状态，R/W=1 为读操作选择，R/W=0 为写操作选择。RS 信号是 HD44780 识别数据总线上的数据是属于指令代码还是属于显示数据。RS=0 选通指令寄存器通道，数据总线传输的是指令代码或标志位；RS=1 选通数据寄存器通道，数据总线传输的是显示数据或自定义字符的字模数据。接口部信号逻辑功能组合如表 10-8 所列。

表 10-8 HD44780 接口部信号逻辑功能组合表

RS	R/W	E	D7～D0	功 能
0	0	下降沿	输入态	写指令代码
0	1	高电平	输出态	读忙标志和 AC 码
1	0	下降沿	输入态	写数据
1	1	高电平	输出态	读数据

三、字符型液晶控制器的指令

用单片机来控制 LCM，方式十分简单，LCM 其内部可以看成两组寄存器，一个为指令寄存器 IR，一个为数据寄存器 DR，由 RS 引脚来控制。所有对指令寄存器或数据寄存器的存取均需检查 LCM 内部的忙标志 BF，此标志用来告知 LCM 内部正在工作，并不允许接收任何的控制命令。而此位的检查可以令 RS=0，用读取 D7 来加以判断，当 D7 为 0 时，才可以写入指令或数据寄存器。LCM 控制指令共有 11 组，以下分别介绍。

1. 清屏

清屏指令格式如下：

控制信号		控 制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1

指令代码为 01H，将 DDRAM 数据全部填入“空白”的 ASCII 代码 20H，执行此指令将清除显示器的内容，同时光标移到左上角。

2. 光标归位

光标归位指令格式如下：

控制信号		控 制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	×

指令代码为 02H，地址计数器 AC 被清 0，DDRAM 数据不变，光标移到左上角。×

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

表示可以为 0 或 1。

3. 输入方式设置

输入方式设置指令格式如下：

控制信号		控 制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1	I/D	S

该指令用来设置光标、字符移动的方式，具体情况如表 10-9 所列。

表 10-9　光标和字符移动方式的设置

状 态 位		指令代码	功　能
I/D	S		
0	0	04H	光标左移 1 格，AC 值减 1，字符全部不动
0	1	05H	光标不动，AC 值减 1，字符全部右移 1 格
1	0	06H	光标右移 1 格，AC 值加 1，字符全部不动
1	1	07H	光标不动，AC 值加 1，字符全部左移 1 格

4. 显示开关控制

显示开关控制指令格式如下：

控制信号		控 制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	D	C	B

指令代码为 08H～0FH。该指令控制字符、光标及闪烁的开与关，有 3 个状态位 D、C、B，这 3 个状态位分别控制着字符、光标和闪烁的显示状态。

D 是字符显示状态位。当 D=1 时为开显示，D=0 时为关显示。注意关显示仅是字符不出现，而 DDRAM 内容不变。这与清屏指令不同。

C 是光标显示状态位。当 C=1 时为光标显示，C=0 时为光标消失。光标为底线形式(5×1 点阵)，光标的位置由地址指针计数器 AC 确定，并随其变动而移动。当 AC 值超出了字符的显示范围，光标将随之消失。

B 是光标闪烁显示状态位。当 B=1 时，光标闪烁；B=0 时，光标不闪烁。

5. 光标、字符位移

光标、字符位移指令的格式如下：

控制信号		控 制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	S/C	R/L	×	×

执行该指令将产生字符或光标向左或向右滚动一个字符位。如果定时间隔地执行该指令，将产生字符或光标的平滑滚动，具体情况如表 10-10 所列。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

表 10-10　光标、字符位移的设置

状态位		指令代码	功　能
S/C	R/L		
0	0	10H	光标左滚动
0	1	14H	光标右滚动
1	0	18H	字符左滚动
1	1	1CH	字符右滚动

6. 功能设置

功能设置指令格式如下：

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	DL	N	F	0	0

该指令用于设置控制器的工作方式，有 3 个参数 DL、N 和 F。它们的作用是：

①DL 用于设置控制器与计算机的接口形式。接口形式体现在数据总线长度上。DL=1，设置数据总线为 8 位长度，即 D7～D0 有效。DL=0，设置数据总线为 4 位长度，即 D7～D4 有效。在该方式下，8 位指令代码和数据将按先高 4 位、后低 4 位的顺序分两次传输。

②N 用于设置显示的字符行数。N=0 为一行字符行，N=1 为两行字符行。

③F 用于设置显示字符的字体。F=0 为 5×7 点阵字符体，F=1 为 5×10 点阵字符体。

7. CGRAM 地址设置

CGRAM 地址设置指令格式如下：

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	A5	A4	A3	A2	A1	A0

该指令将 6 位的 CGRAM 地址写入地址指针计数器 AC 内，随后，单片机对数据的操作是对 CGRAM 的读/写操作。

8. DDRAM 地址设置

DDRAM 地址设置指令格式如下：

控制信号		控制代码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	A6	A5	A4	A3	A2	A1	A0

该指令将 7 位的 DDRAM 地址写入地址指针计数器 AC 内，随后，单片机对数据的操作是对 DDRAM 的读/写操作。

9. 读 BF 及 AC 值

读 BF 及 AC 指令的格式如下：

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

控制信号		控 制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0

LCD 的忙标志 BF 用以指示 LCD 目前的工作情况，当 BF＝1 时，表示正在做内部数据的处理，不接受单片机送来的指令或数据；当 BF＝0 时，则表示已准备接收命令或数据。当程序读取此数据的内容时，D7 表示忙标志，而另外 D6～D0 的值表示 CGRAM 或 DDRAM 中的地址，至于是指向哪一地址，则根据最后写入的地址设定指令而定。

10. 写数据到 CGRAM 或 DDRAM

写数据到 CGRAM 或 DDRAM 的指令格式如下：

控制信号		控 制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	0								

先设定 CGRAM 或 DDRAM 地址，再将数据写入 D7～D0 中，以使 LCD 显示出字形。也可将使用者自创的图形存入 CGRAM。

11. 从 CGRAM 或 DDRAM 读取数据

从 CGRAM 或 DDRAM 读取数据的指令格式如下：

控制信号		控 制 代 码							
RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	1								

先设定 CGRAM 或 DDRAM 地址，再读取其中的数据。

四、字符显示实验演练

实验 1：在下载型实验板上带有 LCD 接口，可直接与字符型液晶相连。实验板上数据线被连到 P0 口，P2.5 接 RS 端，P2.6 接 RW 端，P2.7 接 E 端，如图 10-20 所示。下面

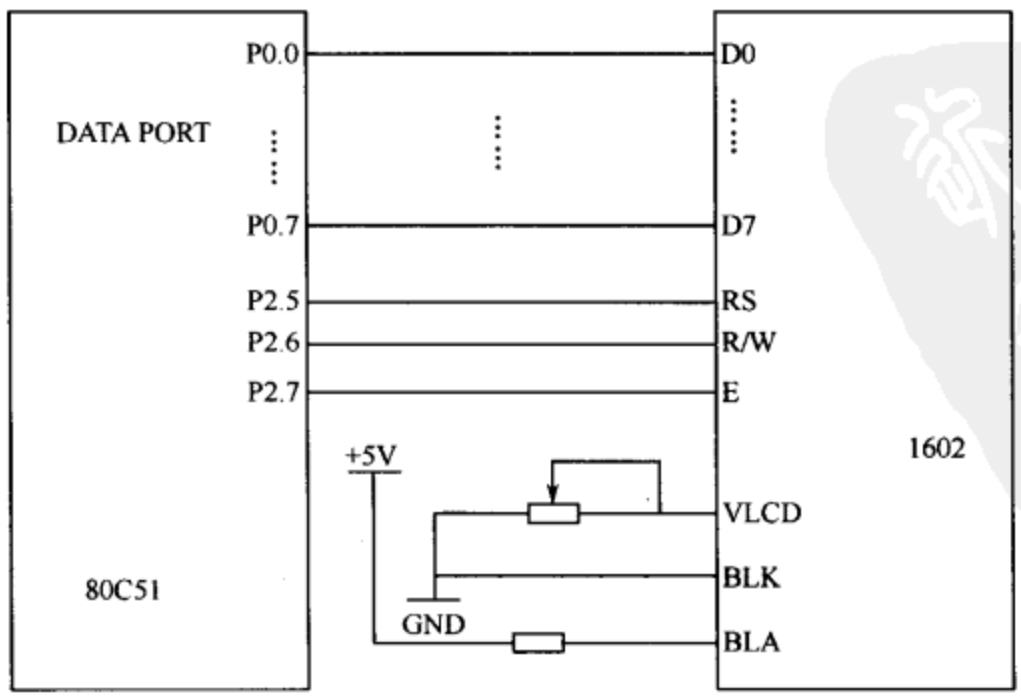


图 10-20　字符型液晶显示器与单片机的连接

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

采用下载型实验板和 1602 液晶显示模块做如下实验：在液晶模块的第 2 行第 1 个字符的位置显示字母“A”。

实验源程序如下：

```
#include <reg51.h>
#include <intrins.h>                    //内部函数,包含_nop_()空函数指令
#define uchar unsigned char
#define uint unsigned int
sbit RS=P2^5;                           //定义 LCD 模块控制引脚
sbit RW=P2^6;
sbit E=P2^7;
#define DataPort P0                     //定义 LCD 模块数据端口
#define Busy 0x80                       //忙标志,D7 位为 1 说明忙
uchar Xpos;                             //列方向地址指针
uchar Ypos;                             //行方向地址指针
//以下是短延时函数
void Delay5Ms(void)
 {
    uint i=5552;
    while(i--);
 }
//以下是读忙状态函数
//功能:在正常读写操作之前检测 LCD 模块的忙状态,
//D7 = 0:LCD 控制器空闲;D7 = 1:LCD 控制器忙
void CheckBusy( void )
 {
    DataPort = 0xff;                    //P1 口写 1
    RS=0;                               //选择指令寄存器
    RW=1;                               //选择读模式
    _nop_();
    E=1;                                //使能 LCD
    _nop_();
    _nop_();
    while( DataPort & Busy ); //若 D7=1,则 DataPort&Busy=1,说明忙,等待
    E=0;                                //若 D7=0,说明不忙,令 E=0
  }
//以下是写入 IR 命令函数
//功能:向 LCD 模块写入命令字符 CMD
//若 AttribC=1,检测忙信号;若 AttribC=0,不检测忙信号
void WriteIR(uchar CMD,uchar AttribC)
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
{
    if (AttribC) CheckBusy();   //若 AttribC =1,则检测忙信号,否则,顺序执行
    RS=0;                       //选择指令寄存器
    RW=0;                       //选择写模式
    _nop_();
    DataPort=CMD;               //将命令送数据端口
    _nop_();
    E=1;                        //使能 LCD
    _nop_();
    _nop_();
    E=0;                        //禁止 LCD
  }
//以下是写入 DDR 寄存器函数
//功能:在当前光标位置显示一个字符
void WriteDDR( char c )
 {
    CheckBusy();                //检测忙信号
    RS=1;                       //选择数据寄存器
    RW=0;                       //选择写模式
    _nop_();
    DataPort=c;                 //将显示字符送往数据口
    _nop_();
    E=1;                        //写使能
    _nop_();
    _nop_();
    E=0;                        //禁止 LCD
  }
//以下是光标定位函数
void LcdPos(uchar Xpos,uchar Ypos)
  {
    uchar tmp;                  //定义 tmp 为指令码
      Xpos&=0x0f;               //16xx 型液晶的范围是 0~15
      Ypos&=0x01;               //Y 的范围是 0~1
      tmp=Xpos;
      if(Ypos==1)
      tmp|=0xc0;                //若 Ypos 为 1(显示第 2 行),地址码+0xc0
      tmp|=0x80;                //若 Ypos 为 0(显示第 1 行),地址码+0x80
      WriteIR(tmp,0);
  }
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
//以下是初始化函数
//功能:向 LCD 模块写入不同命令,完成必要的初始化过程
void LcdReset( void )
  {
    WriteIR( 0x38, 0);          //功能设置指令,8 位接口,显示 2 行,5×7 字符
    WriteIR( 0x38, 1);          //设置显示模式(以后均检测忙信号)
    WriteIR( 0x08, 1);          //显示开关控制指令,显示关闭
    WriteIR( 0x01, 1);          //清屏指令,将 DDRAM 数据全部填入"空白"
    WriteIR( 0x06, 1);          //输入方式设置指令,字符不动,光标自动右移一格
    WriteIR( 0x0f, 1);          //显示开关控制指令,显示器开,光标开,光标闪烁
  }
//以下是在指定行列显示字符函数
void WriteChar(uchar Xpos,uchar Ypos,char c)
  {
    LcdPos(Xpos,Ypos);
    WriteDDR(c);
  }
//以下是主函数
void main(void)
  {
    LcdReset();                 //LCD 模块初始化
    Delay5Ms();                 //延时等待复位
    WriteChar(0,1,'A');         //Xpos=0(第 1 列),Ypos=1(第 2 行),显示字符 A
    for(;;)
    {;}
  }
```

实验步骤如下:

①打开 Keil 软件,输入上面的程序,保存为 lcd1. c。对程序进行编译、链接和调试,产生 lcd1. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC),将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target',在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_10/lcd1 文件夹中。

实验 2:在下载型实验板做以下实验,从 LCD 的第 2 行第 1 列开始显示"80C51"。

实验源程序如下:

```
#include <reg51. h>
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
#include <intrins.h>                    //内部函数,包含_nop_()空函数指令
#define uchar unsigned char
#define uint unsigned int
sbit RS=P2^5;                           //定义 LCD 模块控制引脚
sbit RW=P2^6;
sbit E=P2^7;
#define DataPort P0                     //定义 LCD 模块数据端口
#define Busy 0x80                       //忙标志,D7 位为 1 说明忙
uchar Xpos;                             //列方向地址指针
uchar Ypos;                             //行方向地址指针

//以下是短延时函数
void Delay5Ms(void)
  {
    uint i=5552;
    while(i--);
  }
//以下是读忙状态函数
//功能:在正常读写操作之前检测 LCD 模块的忙状态,
//D7=0:LCD 控制器空闲;D7=1:LCD 控制器忙
void CheckBusy( void )
  {
    DataPort = 0xff;                    //P1 口写 1
    RS=0;                               //选择指令寄存器
    RW=1;                               //选择读模式
    _nop_();
    E=1;                                //使能 LCD
    _nop_();
    _nop_();
    while(DataPort & Busy);             //若 D7=1,则 DataPort&Busy=1,说明忙,等待
    E=0;                                //若 D7=0,说明不忙,令 E=0
  }
//以下是写入 IR 寄存器命令函数
//功能:向 LCD 模块写入命令字符 CMD
//若 AttribC=1,检测忙信号;若 AttribC=0,不检测忙信号
void WriteIR(uchar CMD,uchar AttribC)
 {
    if(AttribC) CheckBusy();            //若 AttribC=1,则检测忙信号,否则,顺序执行
    RS=0;                               //选择指令寄存器
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    RW=0;                          //选择写模式
    _nop_();
    DataPort=CMD;                  //将命令送数据端口
    _nop_();
    E=1;                           //使能 LCD
    _nop_();
    _nop_();
    E=0;                           //禁止 LCD
  }
//以下是写入 DDR 寄存器函数
//功能:在当前光标位置显示一个字符
void WriteDDR(char c)
 {
    CheckBusy();                   //检测忙信号
    RS=1;                          //选择数据寄存器
    RW=0;                          //选择写模式
    _nop_();
    DataPort=c;                    //将显示字符送往数据口
    _nop_();
    E=1;                           //写使能
    _nop_();
    _nop_();
    E=0;                           //禁止 LCD
  }
//以下是光标定位函数
void LcdPos(uchar Xpos,uchar Ypos)
  {
    uchar tmp;                     //定义 tmp 为指令码
        Xpos&=0x0f;                //16xx 型液晶的范围是 0~15
        Ypos&=0x01;                //Y 的范围是 0~1
        tmp=Xpos;
        if(Ypos==1)
        tmp|=0xc0;                 //若 Ypos 为 1(显示第 2 行),地址码+0xc0
        tmp|=0x80;                 //若 Ypos 为 0(显示第 1 行),地址码+0x80
        WriteIR(tmp,0);
  }
//以下是初始化函数
//功能:向 LCD 模块写入不同命令,完成必要的初始化过程
void LcdReset( void )
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    {
      WriteIR(0x38, 0);          //功能设置指令,8 位接口,显示 2 行,5×7 字符
      WriteIR(0x38, 1);          //设置显示模式(以后均检测忙信号)
      WriteIR(0x08, 1);          //显示开关控制指令,显示关闭
      WriteIR(0x01, 1);          //清屏指令,将 DDRAM 数据全部填入"空白"
      WriteIR(0x06, 1);          //输入方式设置指令,字符不动,光标自动右移一格
      WriteIR(0x0f, 1);          //显示开关控制指令,显示器开,光标开,光标闪烁
    }
  //以下是在指定行列显示字符函数
  void WriteChar(uchar Xpos,uchar Ypos,char c)
    {
        LcdPos(Xpos,Ypos);
        WriteDDR(c);
    }
//以下是显示字符串函数
void WriteString(uchar Xpos,uchar Ypos,char s[])
  {
    uchar p=0;
        for(;;)
          {
     WriteChar(Xpos,Ypos,s[p]);
          p++;
          if(s[p]==0)
          break;
          if(++Xpos>=15)          //每行最多显示 16 个字符
          {Xpos=0;                //如果一行显示不完,则转到下一行或上一
                                  行的第一个字符
           Ypos^=1;}              //如果一行显示不完,则转到下一行或上一
                                  行显示
        }
  }
//以下是主函数
void main(void)
  {
    uchar s[]="80c51";
    LcdReset();                   //LCD 模块初始化
    Delay5Ms();                   //延时等待复位
    WriteString(0,1,s);           //Xpos=0(第 1 列),Ypos=1(第 2 行),显
                                  示显示字符串
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    for(;;)
    {;}
}
```

实验步骤如下：

①打开 Keil 软件，输入上面的程序，保存为 lcd2. c。对程序进行编译、链接和调试，产生 lcd2. hex 目标文件。

②将 MON51 硬件仿真器和 PC 机连接好。

③取下下载型实验板的 CPU(SST89E554RC)，将 MON51 仿真器仿真头插入到实验板的 CPU 位置。

④点击菜单 Project→Option for Target 'Target '，在出现的窗口中设置为硬件仿真。

⑤按 Ctrl+F5 进入调试状态。

⑥按 F5 进行全速执行。观察实验现象。

该实验程序在本书附赠光盘的 ch_10/lcd2 文件夹中。

第四节　I^2C 总线接口

80C51 单片机本身不具有 I^2C 接口，但是，通过用软件进行模拟，可以挂接具有 I^2C 接口的芯片。

一、I^2C 总线及其软件包

串行扩展总线在单片机系统中的应用是目前单片机技术发展的一种趋势。在目前比较流行的几种串行扩展总线中，I^2C 总线以其严格的规范和众多带 I^2C 接口的外围器件而获得广泛应用。I^2C 总线是 PHILIPS 公司推出的芯片间串行传输总线，它由两根线组成，一根是串行时钟线(SCL)，一根是串行数据线(SDA)。主控器利用串行时钟线发出时钟信号，利用串行数据线发送或接收数据。I^2C 总线由主控器电路引出，凡具有 I^2C 接口的电路(受控器)都可以挂接在 I^2C 总线上，主控器通过 I^2C 总线对受控器进行控制。

随着 I^2C 总线研究的深入，I^2C 总线已经广泛应用于视/音频领域、IC 卡行业和一些家电产品中，在智能仪器、仪表和工业测控领域也越来越多地得到应用。

1. I^2C 总线的特点

I^2C 总线的广泛应用是同它卓越的性能和简便的操作方法分不开的。I^2C 总线的特点主要表现在以下几个方面：

①硬件结构上具有相同的硬件接口界面。I^2C 总线系统中，任何一个 I^2C 总线接口的外围器件，不论其功能差别有多大，都是通过串行数据线(SDA)和串行时钟线(SCL)连接到 I^2C 总线上。这一特点给用户在设计应用系统中带来了极大的便利性。用户不必理解每个 I^2C 总线接口器件的功能如何，只需将器件的 SDA 和 SCL 引脚连到 I^2C 总线上，然后对该器件模块进行独立的电路设计，从而简化了系统设计的复杂性，提高了系统抗干扰的能力。

②总线接口器件地址具有很大的独立性。每个 I^2C 接口芯片具有唯一的器件地址，由于不能发出串行时钟信号而只能作为从器件使用。各器件之间互不干扰，相互之间不

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

能进行通信，各个器件可以单独供电。单片机与 I^2C 器件之间的通信是通过独一无二的器件地址来实现的。

③软件操作的一致性。由于任何器件通过 I^2C 总线与单片机进行数据传送的方式是基本一样的，这就决定了 I^2C 总线软件编写的一致性。

④PHILIPS 公司在推出 I^2C 总线的同时，也为 I^2C 总线制订了严格的规范，如：接口的电气特性、信号时序、信号传输的定义等。规范的严密性，结构的独立性和硬、软件接口界面的一致性，极大地方便了 I^2C 总线设计的模块化和规范化，伴随而来的是用户在使用 I^2C 总线时的“傻瓜”化。

2. I^2C 总线数据的传输规则

①在 I^2C 总线上的数据线 SDA 和时钟线 SCL 都是双向传输线，它们的接口各自通过一个上拉电阻接到电源正端。当总线空闲时，SDA 和 SCL 必须保持高电平。为了使总线上所有电路的输出能完成一个线“与”的功能，各接口电路的输出端必须是开路漏极或开路集电极。

②进行数据传送时，在时钟信号高电平期间，数据线上的数据必须保持稳定；只有时钟线上的信号为低电平期间，数据线上的高电平或低电平才允许变化，如图 10-21 所示。

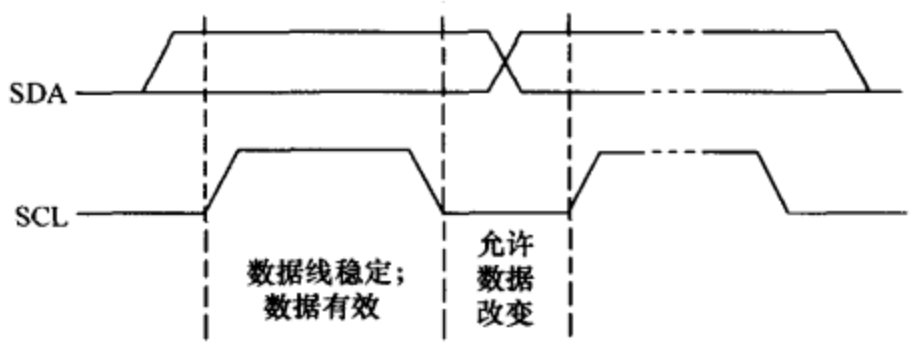


图 10-21　数据的有效性

③在 I^2C 总线的工作过程中，当时钟线保持高电平期间，数据线由高电平向低电平变化定义为起始信号(S)，而数据线由低电平向高电平的变化定义为一个终止信号(P)，如图 10-22 所示，起始信号和终止信号均由主控器产生。

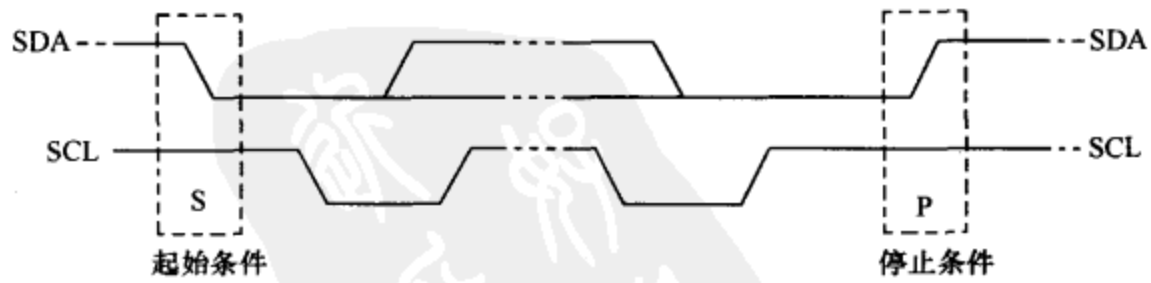


图 10-22　起始和停止条件

④I^2C 总线传送的每一字节均为 8 位，但每启动一次总线，传输的字节数没有限制，由主控器发送时钟脉冲及起始信号、寻址字节和停止信号，受控器件必须在收到每个数据字节后做出响应，在传送一个字节后的第 9 个时钟脉冲位，受控器输出低电平作为应答信号。此时，要求发送器在第 9 个时钟脉冲位上释放 SDA 线，以便受控器送出应答信号，将 SDA 线拉成低电平，表示对接收数据的认可，应答信号用 ACK 或 A 表示，非应答信号用 $\overline{ACK}$ 或 $\overline{A}$ 表示，当确认后，主控器可通过产生一个停止信号来终止总线数据传输。I^2C 总线数据传输示意图如图 10-23 所示。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

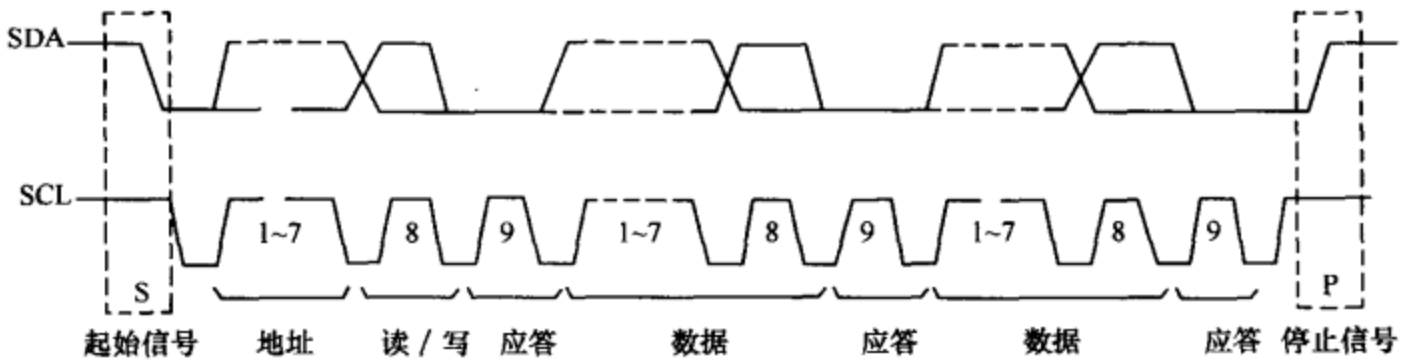


图 10-23　I²C 总线数据传输示意图

需要说明的是，当主控器接收数据时，在最后一个数据字节，必须发送一个非应答位，使受控器释放 SDA 线，以便主控器产生一个停止信号来终止总线数据传输。

3. I²C 总线数据的读写格式

总线上传送数据的格式是指为被传送的各项有用数据安排的先后顺序，这种格式是人们根据串行通信的特点，传送数据的有效性、准确性和可靠性而制定的。另外，总线上数据的传送还是双向的，也就是说主控器在指令操纵下，既能向受控器发送数据（写入），也能接收受控器中某寄存器中存放的数据（读取），所以传送数据的格式有“写格式”与“读格式”之分。

（1）写格式

I²C 总线数据的写格式如图 10-24 所示。

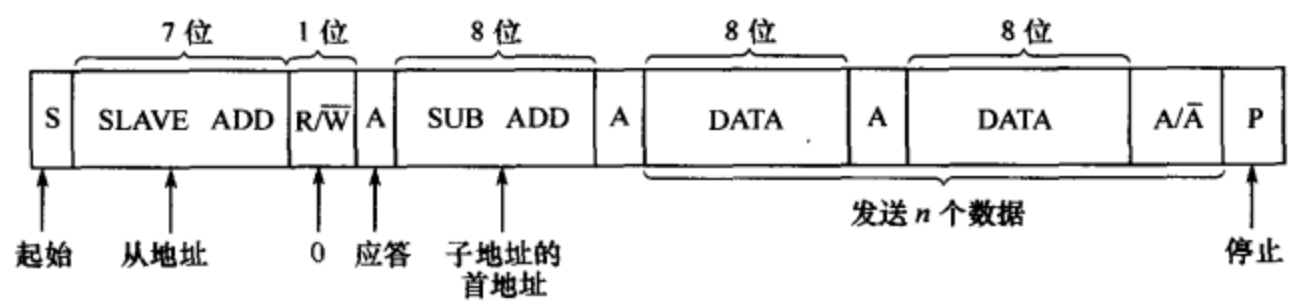


图 10-24　I²C 总线数据的写格式

写格式是指主控器向受控器发送数据，工作过程是：先由主控器发出启动信号（S），随后传送一个带读/写（R/$\overline{W}$）标记的从地址（SLAVE ADD）字节，从地址只有 7bit 长，第 8 位是读/写位（R/$\overline{W}$），用来确定数据传送的方向，对于写格式，R/$\overline{W}$ 应为“0”，表示主控器将发送数据给受控器，接着传送第二个字节，即从地址的子地址（SUB ADD），若受控器有多字节的控制项目，该子地址是指首（第一个）地址，因为子地址在受控器中都是按顺序编制的，这就便于某受控器的数据一次传送完毕；接着才是若干字节的控制数据的传送，每传送一个字节的地址或数据后的第 9 位是受控器的应答信号，数据传送的顺序要靠主控器中程序的支持才能实现，数据发送完毕后，由主控器发出停止信号（P）。

（2）读格式

读格式如图 10-25 所示。

与写格式不同，读格式首先要找到读取数据的受控器的地址，包括从地址和子地址，所以格式中在启动读之前，用写格式发送受控器，再启动读格式，不过前 3 个应答信号因为是指向受控器，所以应由受控器发出；然后，所有数据字节的应答信号因为是指向主控器，因此由主控器发出。不过最后的 $\overline{A}=1$。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

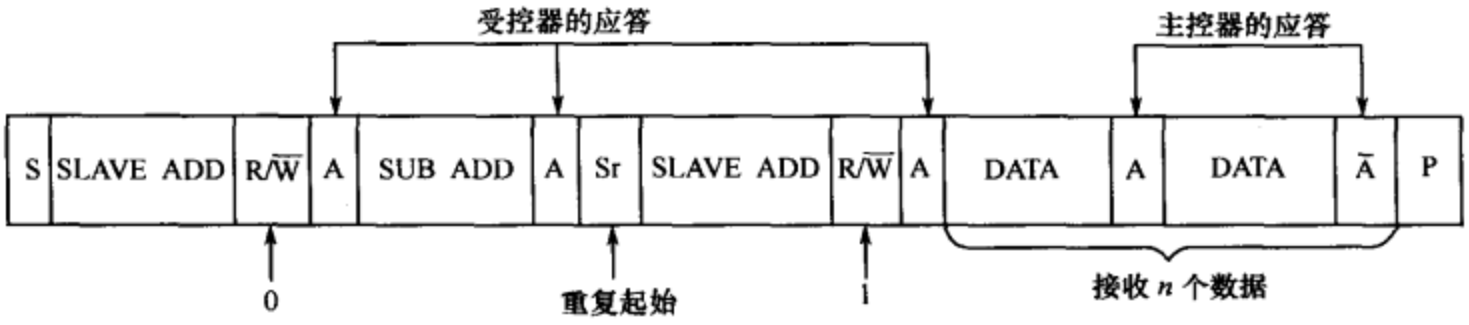


图 10-25　受控器向主控器发送数据(读格式)

重点提示　在设置众多受控器中，为了将控制数据可靠地传送给指定的受控 IC，必须使每一块 IC 编制一个地址码，通常称为从地址，显然从地址不能在不同的 IC 间重复使用。主控器发送寻址字节时，总线上所有受控器都将寻址字节中的 7 位地址与自己的从地址相比较，如果两者相同，则该器件就是被寻址的受控器(从器件)，受控器内部的 n 个数据地址(子地址)的首地址由子地址数据字节指出，I^2C 总线接口内部具有子地址指针自动加 1 功能，所以主控器不必一一发送 n 个数据字节的子地址。

二、I^2C 总线串行存储器 AT24Cxx

一个单片机系统中．存储器起着非常重要的作用。常用的 89C51 或 52 系列单片机有 128 或 256 字节的内部数据存储区(RAM)。在一个规模较小、数据量不大的应用系统中，可不必外部扩充数据存储区，单片机内部数据存储区的容量已经够用。当一个单片机应用系统要处理的数据不仅很多而且又非常重要时，通常的做法是通过单片机总线外部扩充数据存储芯片，早期的数据存储芯片主要有 61xx、62xx 系列，如 62256 便有 32KB 字节容量。这些芯片虽然与单片机接口方便，但也有自身的缺陷：即系统掉电后保存在数据存储区内部的数据会丢失，对于某些对数据要求非常严格的系统而言，这个问题往往是致命的。近来年，随着半导体技术的不断发展，陆续出现了一些新的数据存储芯片，比较典型的有基于 I^2C 总线接口的 24 系列、基于 SPI 总线的 25 系列以及并行总线接口的 28 系列、29 系列和内部带锂电池的 NVRAM 等新型数据存储芯片。这些芯片的共同特点是：芯片掉电后数据不会丢失，数据往往可以保存几年甚至几十年。一般情况下，如果系统要求数据处理实时性高，并且容量较大，可以采用并行总线协议接口的芯片，如 28 系列、29 系列，或者是新型的 NVRAM 和铁电 FRAM 存储芯片。如果系统要求数据处理实时性较低，也可采用串行总线协议的存储芯片，典型的有 24xx 和 25xx 系列的芯片。这些芯片一般采用 I^2C、SPI 总线协议，与单片机接口通常仅占用 2 个～4 个 I/O 口，可以最大限度地节省单片机的资源，并且数据可以反复擦写，这些芯片自身的功耗也较低(μA 级)，并且价格便宜，因此近年来在数据处理中已得到广泛的应用。下面以比较典型的 I^2C 总线结构的 AT24Cxx 为例，介绍具有 I^2C 串行总线结构的数据存储芯片的基本应用。

1. I^2C 总线串行存储器的结构

AT24Cxx 是美国 ATMEL 公司的串行 EEPROM 芯片，容量分别为 128×8 位和 256×8位，该芯片具有页写功能，数据保存周期达 100 年。存储器容量与型号有关，如 AT24C02 为 256 字节(又称 1 页)；AT24C04 为 512 字节(2 页)；AT24C08 为 1024 字节(4 页)，AT24C16 为 2048 字节(8 页)。AT24Cxx 的管脚排列如图 10-26 所示。其中 A0、A1、A2 为器件地址选择线，SDA 为串行数据线，SCL 为串行时钟线，WP(EN)为写保

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

护端(当该端为高电平时,不可对存储器写操作),V_{cc}为1.8V～5.5V正电压,V_{ss}为地。

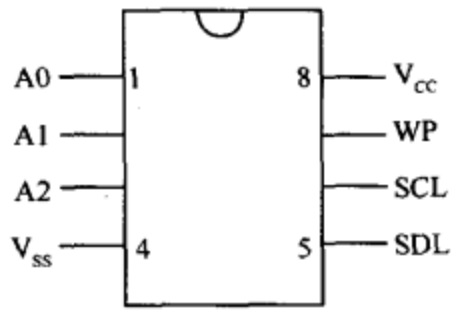


图10-26　AT24Cxx的管脚排列

2. I²C总线串行存储器的从地址

AT24Cxx从地址设置如图10-27所示。

从图中可以看出,AT24Cxx的从地址由7位地址和1位方向位组成。其中,高4位1010为器件地址,由I²C委员会分配,最低1位R/W为方向位,当R/W=0时,对存储器进行写操作;当R/W=1时,对存储器进行读操作。其他3位为硬地址位,可选择接地、接V_{CC}或悬空。

对于容量只有128/256字节的AT24C01/02而言,A2、A1、A0为硬地址,可选择接地或V_{cc},当选择接地时,该存储器的写从地址为10100000(十六进制为0A0H),读从地址为10100001(十六进制为0A1H)。

AT24C01　　AT24C02

1	0	1	0	A2	A1	A0	R/W

AT24C04

1	0	1	0	A2	A1	P0	R/W

AT24C08

1	0	1	0	A2	P1	P0	R/W

AT24C16

1	0	1	0	P2	P1	P0	R/W

图10-27　AT24Cxx从地址设置

对于容量具有512字节的AT24C04而言,硬地址是A2、A1,其中A0悬空,划归页地址P0使用,读/写第0页的256个字节子地址时,其从地址应赋予P0=0,读/写第1页的256个字节子地址时,其从地址应赋予P0=1,因为8位子地址只能寻址256个字节。可见,当A0悬空时,可对512个字节进行寻址。若A0接地,其子地址只能在第0页(256个字节)中寻址。这说明,尽管AT24C04的字节容量有512个,但第1页的存储容量被放弃。

对于AT24C08,A1、A0应选择悬空,对于AT24C16,A0、A1、A2应选择悬空,只有这样,才能充分利用其内部地址单元。

另一方面,若A2、A1、A0未悬空,可以任选接地或接V_{CC},这样,A2、A1、A0就有8种不同的选择,说明一对总线系统最多可以同时连接8块AT24C02而不发生地址冲突。不言而喻,一对总线最多也可以同时连接4块AT24C04或者2块AT24C08,因为可以选择的硬地址位分别有A2、A1或A2。不过这种使用多块存储器的方法在单片机设计中很少采用。

重点提示　通常EEPROM写入时,总需要一定的写入时间(5ms～15ms),因此,在写入程序中无法连续写入多个数据字节。为了能连续写入多个数据字节,常设置页写功能,即在EEPROM器件中设有一定容量(页写)的数据寄存器,用户一次写入EEPROM

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

的数据字节不大于页写字节数时，可按通常 RAM 的写入速度，装载至 EEPROM 中的数据寄存器中，随后启动自动写入定时控制逻辑，经过 5ms～10ms 时间，自动将数据寄存器中的数据同步写入 EEPROM 的指定单元中。这样一来，只要一次写入字节数不多于页写容量，总线对 EEPROM 的操作可视为对静态 RAM 的操作，只要求下次数据操作在 5ms～10ms 时间之后进行。AT24C01/02/04/08/16 的页写字节数分别为 4/8/16/16/16。

3. I²C 总线软件设计

PHILIPS 公司提供了标准的 I²C 总线状态处理软件包，并要求系统主从器件都具有 I²C 总线接口，这对于 PHILIPS 公司的单片机如 87LPC76x 系列而言，通过这个软件包去处理 I²C 器件是比较容易的。但是目前其他公司的绝大多数单片机并不具有 I²C 总线接口，如 89C51、78E51 等，这时可以采用普通 I/O 口模拟 I²C 总线的工作方式来实现 I²C 总线上主控制器对从器件的读、写操作，软件编写只要符合 I²C 总线数据传输的时序要求即可，启动、停止、发送应答和发送非应答信号时序要求如图 10-28 所示。

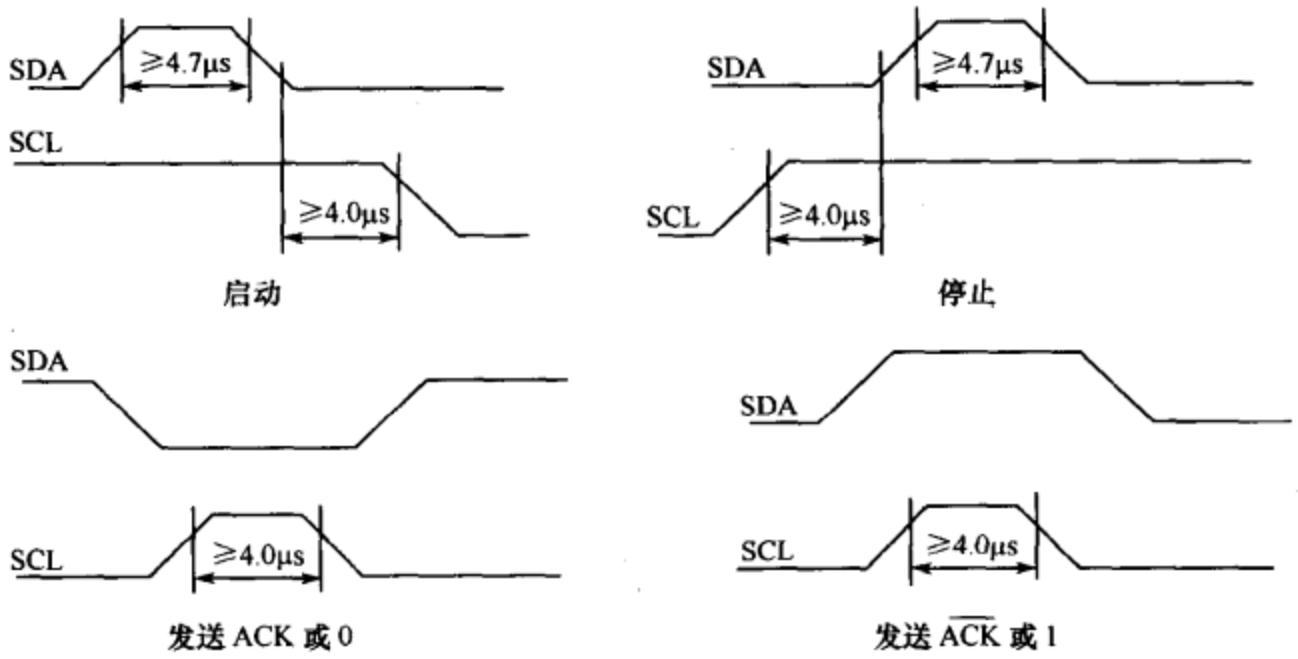


图 10-28　串行 EEPROM 时序规范

下面根据模拟方式下的 I²C 总线时序，选用 P3.6、P3.7 作为时钟线 SCL 和数据线 SDA，给出模拟 I²C 程序清单：

```
#include <reg51.h>                    //头文件的包含
#include <intrins.h>
#define uchar unsigned char           //宏定义
#define uint unsigned int
sbit SDA=P3^7;                        //模拟 I²C 数据传送位
sbit SCL=P3^6;                        //模拟 I²C 时钟控制位
bit ack;                              //应答标志位
```

(1)起动总线函数

函数原型：void Start_I2c()

功能：启动 I²C 总线，即发送 I²C 起始条件

```
void Start_I2c()
  {
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    SDA=1;                          //发送起始条件的数据信号
    _nop_();
    SCL=1;
    _nop_();                        //起始条件建立时间大于 4.7μs,延时
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    SDA=0;                          //发送起始信号
    _nop_();                        //起始条件锁定时间大于 4μs
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    SCL=0;                          //钳住 I²C 总线,准备发送或接收数据
    _nop_();
    _nop_();
  }
```

(2)结束总线函数

函数原型:void Stop_I2c()

功能:结束 I²C 总线,即发送 I²C 结束条件

```
void Stop_I2c()
  {
    SDA=0;                          //发送结束条件的数据信号
    _nop_();                        //发送结束条件的时钟信号
    SCL=1;                          //结束条件建立时间大于 4μs
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    SDA=1;                          //发送 I²C 总线结束信号
    _nop_();
    _nop_();
    _nop_();
    _nop_();
  }
```

(3)字节数据发送函数

函数原型: void SendByte(uchar c)

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

功能：将数据 c 发送出去，可以是地址，也可以是数据，发完后等待应答，并对此状态位进行操作（不应答或非应答都使 ack＝0 假），发送数据正常，ack＝1；ack＝0 表示被控器无应答或损坏。

```
void SendByte(uchar c)
  {
   uchar BitCnt;
   for(BitCnt=0;BitCnt<8;BitCnt++)   //要传送的数据长度为 8 位
    {
       if((c<<BitCnt)&0x80)SDA=1;  //判断发送位
       else SDA=0;
       _nop_();
       SCL=1;                      //置时钟线为高，通知被控器开始接收
                                   数据位
       _nop_();
       _nop_();                    //保证时钟高电平周期大于 4μs
       _nop_();
       _nop_();
       _nop_();
       SCL=0;
    }
       _nop_();
       _nop_();
       SDA=1;                      //8 位发送完后释放数据线，准备接收
                                   应答位
       _nop_();
       _nop_();
      SCL=1;
      _nop_();
      _nop_();
      _nop_();
      if(SDA==1)ack=0;
      else ack=1;                  //判断是否接收到应答信号
      SCL=0;
      _nop_();
      _nop_();
  }
```

(4)字节数据接收函数

函数原型：uchar RcvByte()

功能：用来接收从器件传来的数据，并判断总线错误（不发应答信号），发完后用应答

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

函数

```
uchar RcvByte()
{
  uchar retc;
  uchar BitCnt;

  retc=0;
  SDA=1;                          //置数据线为输入方式
  for(BitCnt=0;BitCnt<8;BitCnt++)
      {
        _nop_();
        SCL=0;                    //置时钟线为低,准备接收数据位
        _nop_();
        _nop_();                  //时钟低电平周期大于 4.7μs
        _nop_();
        _nop_();
        _nop_();
        SCL=1;                    //置时钟线为高,使数据线上数据有效
        _nop_();
        _nop_();
        retc=retc<<1;
        if(SDA==1)retc=retc+1;  //读数据位,接收的数据位放入 retc 中
        _nop_();
        _nop_();
      }
  SCL=0;
  _nop_();
  _nop_();
  return(retc);
}
```

(5)应答子函数

函数原型：void Ack_I2c(bit a)

功能:主控器进行应答信号(可以是应答或非应答信号)

```
void Ack_I2c(bit a)
  {
    if(a==0)SDA=0;              //在此发出应答或非应答信号
    else SDA=1;
    _nop_();
    _nop_();
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    _nop_();
    SCL=1;
    _nop_();
    _nop_();                          //时钟低电平周期大于 4μs
    _nop_();
    _nop_();
    _nop_();
    SCL=0;                            //清时钟线,钳住 I²C 总线以便继续接收
    _nop_();
    _nop_();
  }
```

(6)向无子地址器件发送字节数据函数

函数原型: bit ISendByte(uchar sla,ucahr c)

功能:从启动总线到发送地址,数据,结束总线的全过程,从器件地址 sla,如果返回1,表示操作成功,否则操作有误。注意:使用前必须已结束总线。

```
bit ISendByte(uchar sla,uchar c)
  {
    Start_I2c();                      //启动总线
    SendByte(sla);                    //发送器件地址
    if(ack==0)return(0);
    SendByte(c);                      //发送数据
    if(ack==0)return(0);
    Stop_I2c();                       //结束总线
    return(1);
  }
```

(7)向有子地址器件发送多字节数据函数

函数原型: bit ISendStr(uchar sla,uchar suba,ucahr * s,uchar no)

功能:从启动总线到发送地址,子地址,数据,结束总线的全过程,从器件地址 sla,子地址 suba,发送内容是 s 指向的内容,发送 no 个字节。如果返回 1 表示操作成功,否则操作有误。注意:使用前必须已结束总线。

```
bit ISendStr(uchar sla,uchar suba,uchar * s,uchar no)
  {
    uchar i;
    Start_I2c();                      //启动总线
    SendByte(sla);                    //发送器件地址
    if(ack==0)return(0);
    SendByte(suba);                   //发送器件子地址
    if(ack==0)return(0);
    for(i=0;i<no;i++)
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    {
      SendByte(*s);                    //发送数据
      if(ack==0)return(0);
      s++;
     }
    Stop_I2c();                        //结束总线
    return(1);
  }
```

(8)向无子地址器件读字节数据函数

函数原型：bit IRcvByte(uchar sla,ucahr *c)

功能：从启动总线到发送地址，读数据，结束总线的全过程，从器件地址 sla，返回值在 c。如果返回 1，表示操作成功，否则操作有误。注意：使用前必须已结束总线。

```
bit IRcvByte(uchar sla,uchar *c)
  {
    Start_I2c();                       //启动总线
    SendByte(sla+1);                   //发送器件地址
    if(ack==0)return(0);
    *c=RcvByte();                      //读取数据
    Ack_I2c(1);                        //发送非应答位
    Stop_I2c();                        //结束总线
    return(1);
  }
```

(9)向有子地址器件读取多字节数据函数

函数原型：bit ISendStr(uchar sla,uchar suba,ucahr *s,uchar no)

功能：从启动总线到发送地址，子地址，读数据，结束总线的全过程，从器件地址 sla，子地址 suba，读出的内容放入 s 指向的存储区，读 no 个字节。如果返回 1，表示操作成功，否则操作有误。注意：使用前必须已结束总线。

```
bit IRcvStr(uchar sla,uchar suba,uchar *s,uchar no)
  {
    uchar i;
    Start_I2c();                       //启动总线
    SendByte(sla);                     //发送器件地址
    if(ack==0)return(0);
    SendByte(suba);                    //发送器件子地址
    if(ack==0)return(0);
    Start_I2c();
    SendByte(sla+1);
    if(ack==0)return(0);
    for(i=0;i<no-1;i++)
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    {
      * s=RcvByte();                    //发送数据
       Ack_I2c(0);                      //发送应答位
      s++;
    }
   * s=RcvByte();
    Ack_I2c(1);                         //发送非应答位
    Stop_I2c();                         //结束总线
    return(1);
}
```

三、实验演练

实验:在下载型实验板上做以下实验:将数据1、2、3、4、5写入到AT24C01从00开始的5个单元子地址中,然后,再读出5个数据,读完后,P1口外接的LED显示器点亮1s。

下载型实验板采用的串行存储器是AT24C01,AT24C01的5脚(SDA)与单片机的P3.7相连,AT24C01的6脚(SCL)与单片机的P3.6相连,AT24C01的1、2、3脚(A0、A1、A2)接地,因此,AT24C01的写从地址为0xa0,读从地址为0xa1。

实验源程序如下:

```
#include <reg51.h>                          //头文件的包含
#include <intrins.h>
#define uchar unsigned char                 //宏定义
#define uint unsigned int
sbit SDA=P3^7;                              //模拟I²C数据传送位
sbit SCL=P3^6;                              //模拟I²C时钟控制位
bit ack;                                    //应答标志位
//加入前面(1)~(9)模拟I²C总线函数
//以下是延时函数
void Delay(uint i)
{
    uint j;
    for(;i>0;i--)
    for(j=0;j<125;j++)
    {;}
  }
//以下是主函数
  main()
  {
    unsigned  char  T[5]={1,2,3,4,5};   //定义要写的数据
    unsigned  char  R[5];               //定义读缓冲区
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
bit   bp;
P1=0xff;                          //P1 口 LED 熄灭
bp=ISendStr(0xa0,00,T,5);         //从 00 单元开始写 5 个数据
bp=IRcvStr(0xa1,00,R,5);          //从 00 单元开始读取 5 个数据
P1=0x00;                          //P1 口 LED 点亮
Delay(1000);
P1=0xff;                          //P1 口 LED 熄灭
while(1);
}
```

方法技巧 也可使用 Keil 软件将前面介绍的(1)～(9)模拟 I^2C 总线函数进行编译，保存为 VIIC_C51.C 文件，然后，将产生的库文件 VIIC_C51.LIB 复制到 C51\LIB，把 VIIC_C51.H 复制到 C51\INC，然后在用户程序开头加入 #include <VIIC_C51.h>，即可使用模拟 I^2C 函数。

第五节 A/D 和D/A 转换接口

单片机的外部设备不一定都是数字式的，也经常会和模拟式的设备连接。例如单片机控制温度、压力时，温度和压力都是连续变化的，都是模拟量，在单片机与外部环境通信的时候，就需要有一种转换器来把模拟信号变为数字信号，以便能够输送给单片机进行处理。这种将模拟量转换为数字量的接口电路称为模数(A/D)转换电路。

单片机送出的控制信号，也必须经过变换器变成模拟信号，才能为控制电路所接受。这种将数字量变换为模拟量的接口电路称为数模(D/A)转换电路。

A/D 转换器和 D/A 转换器已成为计算机系统中不可缺少的接口电路。

一、D/A(数模)转换接口

目前，D/A 转换器从接口上可分为两大类：并行接口 D/A 转换器和串行接口 D/A 转换器。并行接口 D/A 转换器的引脚多，体积大，占用单片机的口线多；而串行 D/A 转换器的体积小，占用单片机的口线少。下面以常用的并行转换接口 DAC0832 为例进行介绍。

图 10-29 所示是 DAC0832 的内部逻辑结构图。

各引脚的功能如下：

DI0～DI7：8 位数据输入端。

ILE：输入寄存器的数据允许锁存信号。

$\overline{CS}$：输入寄存器选择信号。

$\overline{WR_1}$：输入寄存器的数据写信号。

$\overline{WR_2}$：DAC 寄存器写信号，并启动转换。

XFER：数据向 DAC 寄存器传送信号，传送后即启动转换。

V_{REF}：参考电压输入端。

I_{OUT1}、I_{OUT2}：电流输出端。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

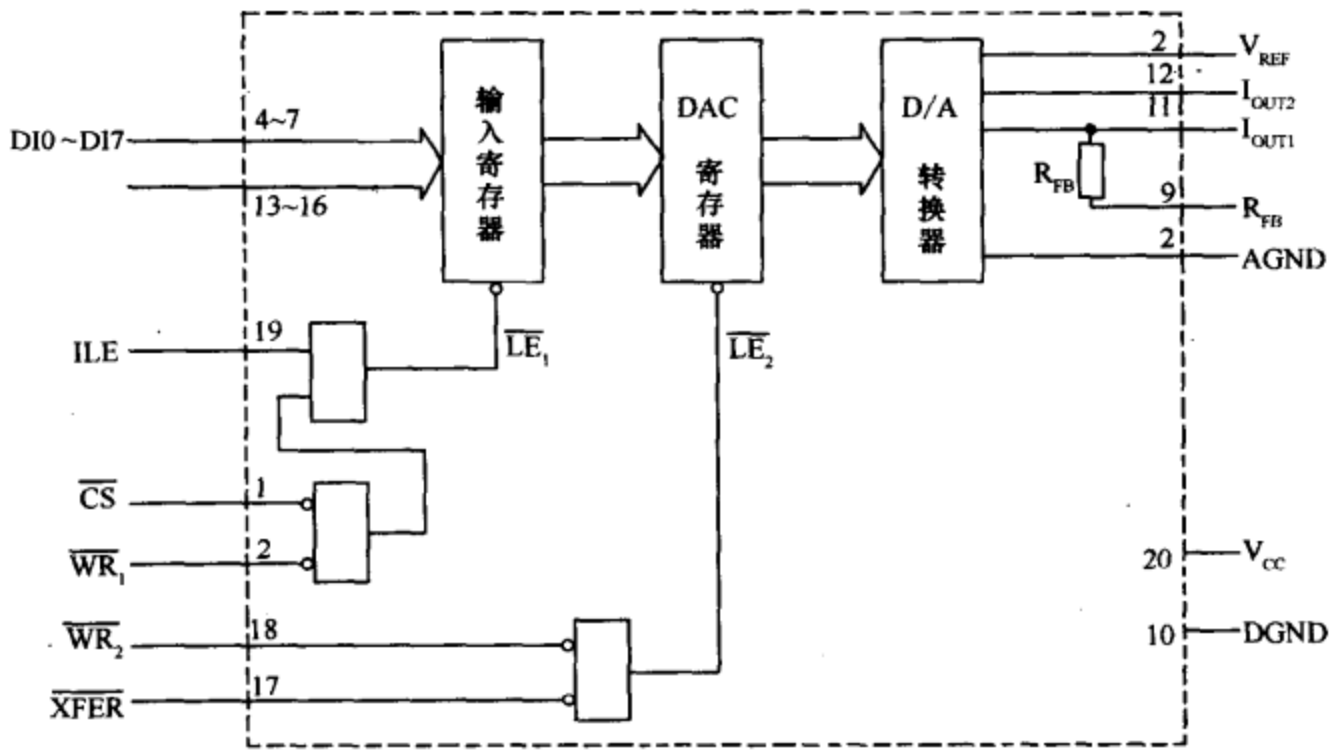


图 10-29　DAC0832 的内部逻辑结构图

R_{FB}：反馈信号输入端。

V_{CC}：电源电压端。

AGND：模拟地。

DGND：数字地。

DAC0832 的输出是电流型的，如果需要输出电压信号，可通过运算放大器进行变换实现。图 10-30 是 DAC0832 与 89C51 单片机的一种接口电路，通过运算放大器 5G24 可得到 0V～5V 的输出电压。该电路采用线选方式规定 DAC0832 的端口地址为 0x7fff。

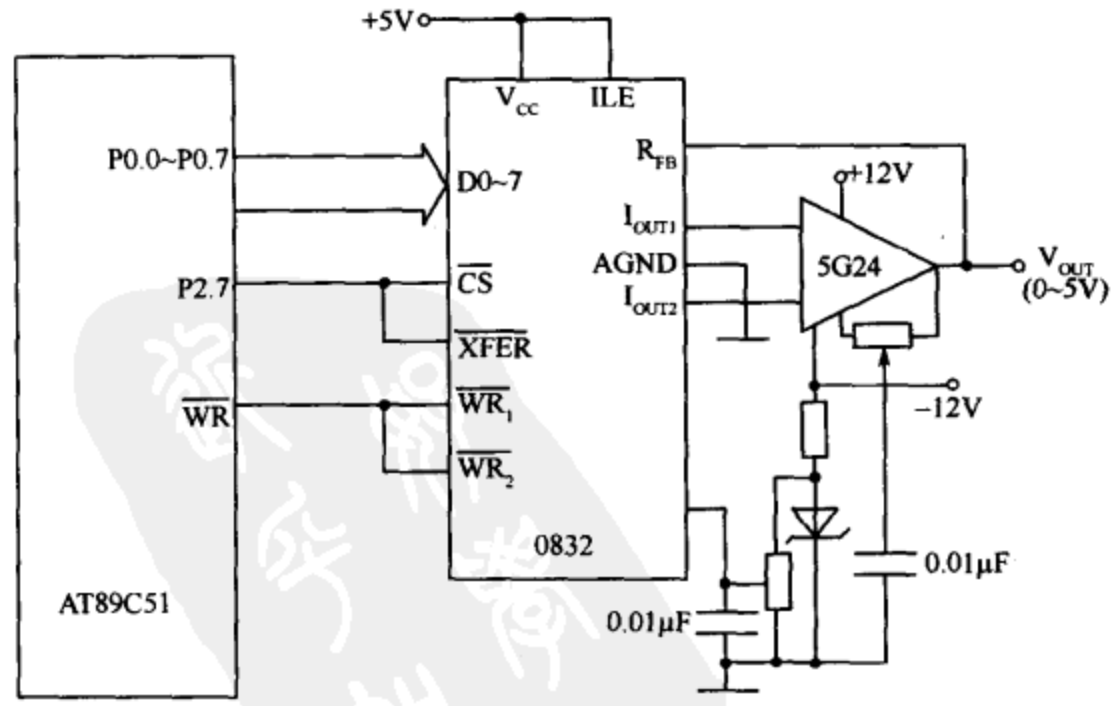


图 10-30　DAC0832 与单片机的连接

下面给出对应于该接口电路的 Cx51 驱动程序：

```
#pragma db oe sb                    //预处理命令
#include<reg51.h>
#include<absacc.h>                  //包含绝对地址操作预定义头文件
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
#define DAC0832 XBYTE[0x7fff]   //定义 DAC0832 端口地址
#define uchar unsigned char
//以下是延时函数
void delay(uchar t)
  {
   while(t--);
  }
//以下是锯齿波发生函数
void saw(void)
  {
    uchar i;
    for (i=0;i<255;i++)
    {DAC0832=i;}
  }
//以下是方波发生函数
void square(void)
  {
    DAC0832=0x00;
    delay(0x10);
    DAC0832=0xff;
    delay(0x10);
  }
//以下是主函数
void main(void)
  {
    uchar i,j;
    i=j=0xff;
    while(i--)
    {saw();}                    //产生一段锯齿波
    while(j--)
    {square();}                 //产生一段方波
```

重点提示 #pragma 是 Cx51 的一个预处理命令，通常用在源程序中向编译器传送各种编译控制命令。其使用格式为：

#pragma 编译命令名序列

例如 #pragma db oe sb 中，各命令表示的内容如下：

db：将符号调试信息加入目标文件；

oe：目标文件中包含附加变量类型信息；

sb：产生程序模块使用过的符号表。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

二、A/D(模数)转换接口

A/D转换器的种类很多,按其工作原理不同分为直接A/D转换器和间接A/D转换器两类。直接A/D转换器可将模拟信号直接转换为数字信号,这类A/D转换器具有较快的转换速度,其典型电路有逐次比较型A/D转换器。间接A/D转换器则是先将模拟信号转换成某一中间电量(时间或频率),然后再将中间电量转换为数字量输出。此类A/D转换器的速度较慢,典型电路是双积分型A/D转换器、电压频率转换型V/F转换器。下面主要以常用的A/D转换器ADC0809为例进行介绍。

ADC0809是一种8路模拟量输入、8位数字量输出的逐次比较式ADC芯片,图10-31所示是ADC0809的原理结构框图,图10-32为ADC0809引脚排列图。芯片的主要部分是一个8位的逐次比较式A/D转换器。为了能实现8路模拟信号的分时采集,在芯片内部设置了多路模拟开关及通道地址锁存和译码电路,因此能对多路模拟信号进行分时采集和转换。转换后的数据送入三态输出数据锁存器。

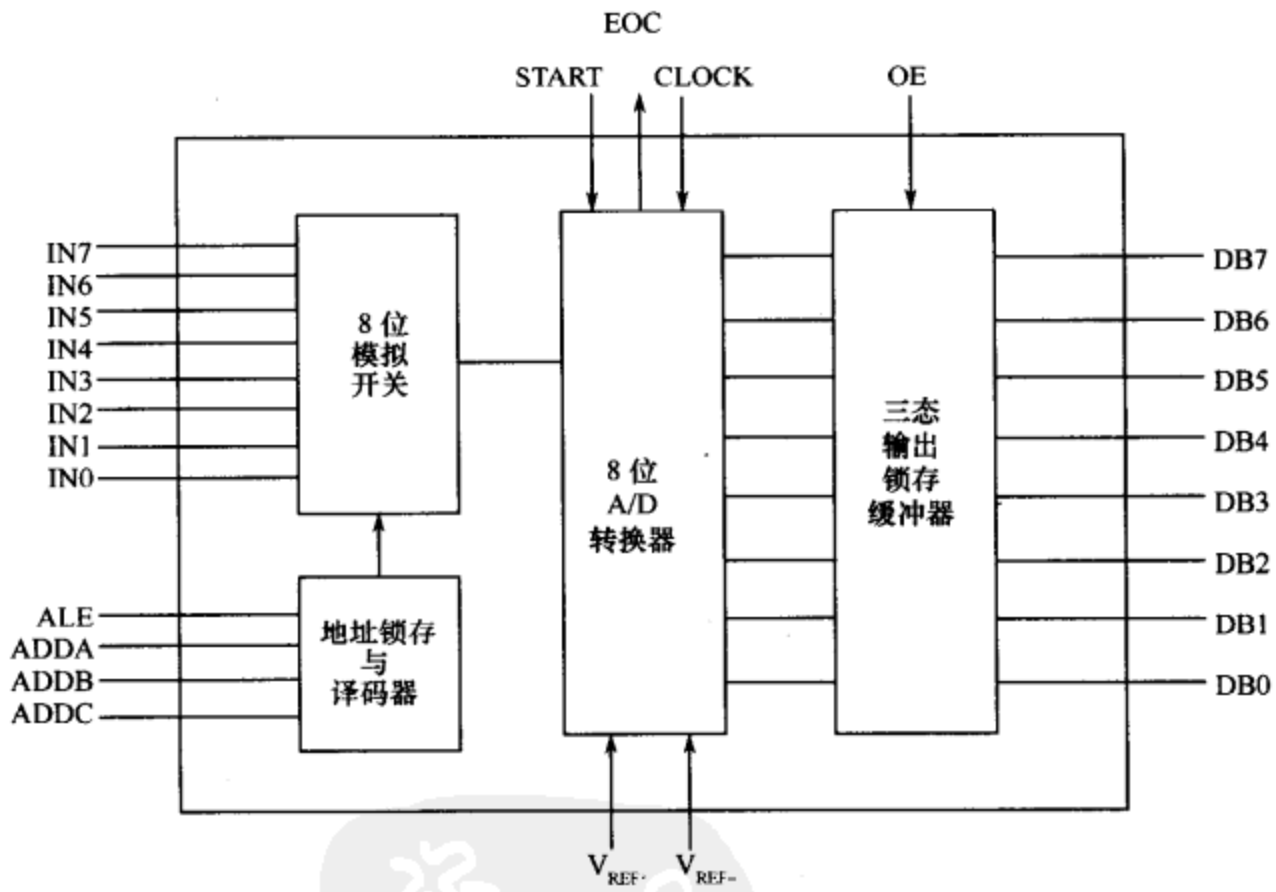


图10-31　ADC0809原理结构框图

ADC0809各引脚的功能如下:

IN0～IN7:8路模拟量输入端。

D0～D7:数字量输出端。

START:启动脉冲输入端,脉冲上升沿复位0809,下降沿启动A/D转换。

ALE:地址锁存信号,高电平有效时把3个地址信号送入地址锁存器,并经地址译码得到地址输出,用以选择相应的模拟输入通道。

EOC:转换结束信号,转换开始时变低,转换结束时变高,变高时将转换结果打入三态输出锁存器。如果将EOC和START相连,加上一个启动脉冲,则连续进行转换。

OE:输出允许信号输入端。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

CLOCK：时钟输入信号，最高允许值为 640kHz。

$V_{REF(+)}$：正基准电压输入端。

$V_{REF(-)}$：负基准电压输入端。通常将 $V_{REF(+)}$ 接+5V，$V_{REF(-)}$ 接地。

V_{CC}：电源电压，可从+5V～+15V。

图 10-33 为 ADC0809 与单片机的一种接口电路。

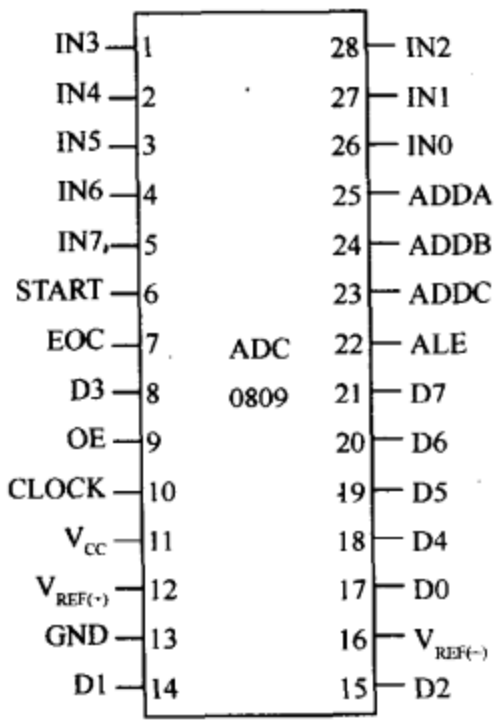


图 10-32　ADC0809 引脚排列图

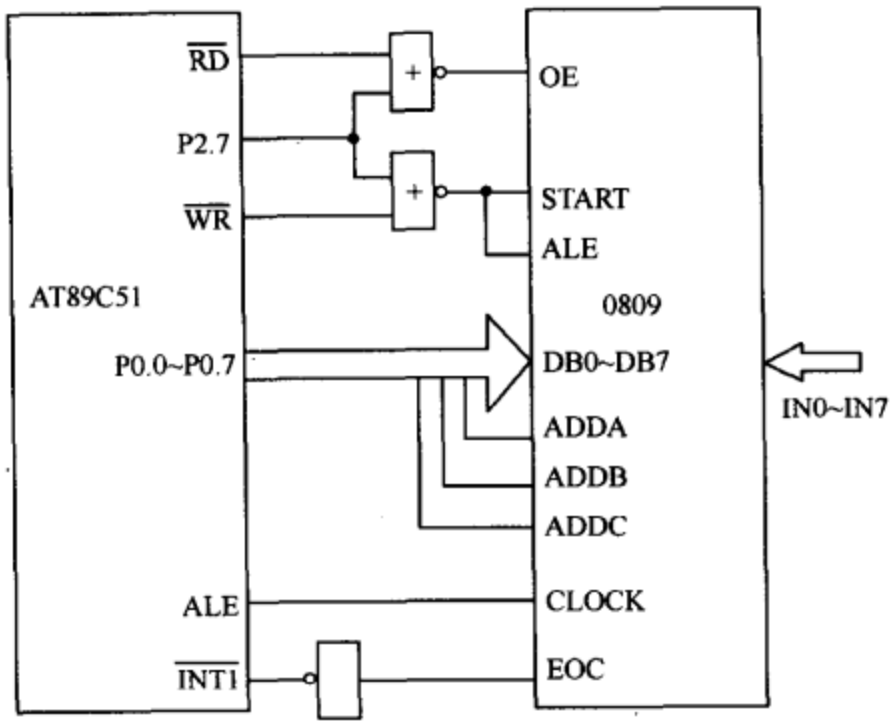


图 10-33　ADC0809 与单片机的连接

采用线选法规定其端口地址，用单片机的 P2.7 引脚作为片选信号，片选信号和$\overline{WR}$信号一起经或非门产生 0809 的启动信号 START 和地址锁存信号 ALE；片选信号和$\overline{RD}$信号一起经或非门产生 0809 输出允许信号 OE，因此端口地址为 0x7fff。0809 的 EOC 信号经反相后接到单片机的$\overline{INT1}$引脚用于产生转换完成的中断请求信号。0809 芯片的 3 位模拟量输入通道地址码输入端 A、B、C 分别接到单片机的 P0.0、P0.1 和 P0.2，故只要向端口地址 0x7fff 分别写入数据 0x00～0x07，即可启动模拟量输入通道 0～7 进行 A/D 转换。

下面给出采用中断工作方式操作以上接口电路的 Cx51 程序，该程序对 8 路模拟输入信号依次进行 A/D 转换，并把转换结果存入外部 RAM 的 0x1000～0x1007 单元。本例采用指针变量实现对存储器地址的直接操作，程序中定义了两个指针变量 * ADC 和 * ADCdata，分别指向 0809 端口地址 0x7fff 和外部 RAM 单元地址 0x1000～0x1007。在 main()函数中通过赋值语句“ * ADC＝i;”启动 0809 进行 A/D 转换，转换结束时产生$\overline{INT1}$中断。在中断服务函数 intl()中通过赋值语句“tmp＝ * ADC;”和“ * ADCdata＝tmp;”读取 A/D 转换的结果值并存储到外部 RAM 单元中去。

采用中断方式的 Cx51 程序如下：

```
#include <reg51.h>
unsigned char xdata * ADC;                    //定义 ADC0809 端口指针
unsigned char xdata * ADCdata;                //定义 ADC0809 数据缓冲器指针
unsigned char i;
void main()
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
  {
    ADC=0x7fff; ADCdata=0x1000;        //定义端口地址和数据缓冲器地址
    i=8;                               //ADC0809 有 8 个模拟输入通道
   EA=1; EX1=1;IT1=1;                  //开中断
    * ADC=i;                           //启动 ADC0809
    while(i);                          //等待 8 个通道 A/D 转换完毕
  }
//以下是外中断 1 服务程序
void int1() interrupt 2
  {
    unsigned char tmp;
    tmp= * ADC;                        //读取 A/D 转换结果
    * ADCdata=tmp;                     //结果值存储到数据缓冲器
    ADCdata++;                         //数据缓冲器指针地址加 1
    i--;
    * ADC=i;                           //启动下一个模拟输入通道 A/D 转换
  }
```

在实际应用中，除了可以采用指针变量来实现对内存地址的直接操作之外，Cx51 编译器还提供了一组关于访问绝对地址的预定义宏“absacc. h”，利用它可以十分方便地实现对任意内存空间直接操作。

采用预定义宏“absacc. h”可将上面的程序改写如下：

```
#include <reg51. h>
#include <absacc. h>                   //包含绝对地址操作预定义头文件
#define ADC 0x7fff                     //定义 ADC0809 端口地址
#define ADCdata 0x1000                 /定义数据缓冲器地址
unsigned char i;
void main()
{
  i=8;                                 //ADC0809 有 8 个模拟输入通道
  EA=1; EX1=1;IT1=1;                   //开中断
  XBYTE[ADC]=i;                        //启动 ADC0809
  while(i);                            //等待 8 个通道 A/D 转换完毕
}
//以下是外中断 1 服务程序
void int1() interrupt 2
{
  unsigned char tmp;
  tmp=XBYTE[ADC];                      //读取 A/D 转换结果
  i--;
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    XBYTE[ADCdata+i]=tmp;      //结果值存储到数据缓冲器
    XBYTE[ADC]=i;              //启动下一个模拟输入通道 A/D 转换
}
```

第六节　单片机应用技术综合实例

一、999 计数器

用下载型实验板制作一个 999 计数器，要求 LED 显示器从 000 开始显示，直至 999 后再回到 000 重新循环显示。

源程序如下：

```
#include<reg51.h>
sbit P22=P2^2;
sbit P23=P2^3;
sbit P24=P2^4;
//以下是延时函数
void Delay(unsigned int i)      //延时程序,i 是形式参数
{
    unsigned int j;
    for(;i>0;i--)               //变量 i 由实际参数传入一个值,因此 i 不能赋初值
    for(j=0;j<125;j++)
    {;}
}
//以下是主函数
void main()
  {
  unsigned char code tab[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,
  0x90};//段码表
  unsigned char a;
  unsigned int m,upbit,lowbit,hibit,cnt=1000;
  while(1)
  {
    for(m=0;m<cnt;m++)
    {
      hibit=m/100;                  //提取百位
      upbit=(m-hibit*100)/10;       //提取十位
      lowbit=m%10;                  //提取个位
      for(a=0;a<30;a++)             //调整 a 的大小可改变计数的显示速度
        {
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
            P0=tab[lowbit];
            P22=0;                          //个位的位选
            Delay(10);
            P22=1;
            P0=tab[upbit];
            P23=0;                          //十位的位选
            Delay(10);
            P23=1;
            P0=tab[hibit];
            P24=0;                          //百位的位选
            Delay(10);
            P24=1;
        }
    }
  }
}
```

该实验程序文件名为 counter. c,在本书附赠光盘的 ch_10/counter 文件夹中,可用下载型实验板进行仿真实验。

二、计数显示器

用下载型实验板制作一个计数显示器,将 T0 作为计数器,对外部信号进行计数,即按开关 K3 时(定时/计数器 T0 的计数输入端 P3. 4),能将所按次数显示在 LED 显示屏上。

源程序如下:

```
#include <reg51.h>
#define uchar unsigned char
#define uint  unsigned int
uchar code tab_bit[]={0x7f,0xbf,0xdf,0xef,0xf7,0xfb};    //位码表
uchar code tab[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,
                  0x88,0x83,0xc6,0xa1,0x86,0x8e,0xff};//"0～F"和"灭"的段
                  码表
uchar count;                            //计数器,显示程序通过它得知现正显示哪个
                                        数码管
uchar disp_buf[6];                      //6 字节的显示缓冲区
//以下是定时/计数器 1 中断函数
void timer1() interrupt 3
{
    uchar tmp;
    TH1=(65536-3000)/256;
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    TL1=(65536-3000)%256;      //定时时间为 3000 个周期
    tmp=tab_bit[count];        //取位值
    P2=P2|0xfc;                //P2 与 11111100B 相或
    P2=P2&tmp;                 //P2 与取出的位值相与
    tmp=disp_buf[count];       //取出待显示的数
    tmp=tab[tmp];              //取段码
    P0=tmp;                    //段码送 P0 口
    count++;
    if(count==6)
    count=0;
}
//以下是主函数
void main()
  {
    uint tmp;
    P1=0xff;
    P0=0xff;
    TMOD=0x15;                 //定时器 0 工作于计数方式 1,定时器 1 工作
                               于定时方式 1
    TH1=(65536-3000)/256;
    TL1=(65536-3000)%256;      //定时时间为 3000 个周期
    TR0=1;                     //定时/计数器 0 开始运行
    TR1=1;                     //定时/计数器 1 开始运行
    EA=1;                      //开总中断
    ET1=1;                     //开定时/计数器 1 中断
    for(;;)
    {
        tmp=TH0*256+TL0;       //取出 T0 中的数值
        disp_buf[5]=tmp%10;    //第 6 个显示
        tmp/=10;
        disp_buf[4]=tmp%10;    //第 5 个显示
        tmp/=10;
        disp_buf[3]=tmp%10;    //第 4 个显示
        tmp/=10;
        disp_buf [2]=tmp%10;   //第 3 个显示
        disp_buf [1]=tmp/10;   //第 2 个显示
        disp_buf [0]=0;        //第 1 个显示
    }
}
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

重点提示 程序中将定时器 T1 用作数码管显示，通过 interrupt 3 关键字定义函数 timerl()为定时器 1 中断服务程序，在这个中断服务程序中，使用 TH1＝(65536－3000)/256；TL1＝(65536－3000)%256；来重置定时器初值，这其中 3000 即为定时周期，这样的写法可以直观地看到定时周期数。定时时间不可过长，否则，显示屏会出现闪烁。实验表明，当定时周期超过 5000 时，就会出现明显的闪烁现象。其余程序段分别完成取位码以选择数码管、从显示缓冲区获得待显示数值，根据该数值取段码以点亮相应笔段等任务。其中使用了一个计数器，该计数器的值为 0～5 对应第 1 到第 6 位的数码管。

主程序的第一部分是做一些初始化的操作，设置定时器工作模式、开启定时器 T1、开启计数器 T0、开启 T1 中断及总中断，随后进入主循环，主循环首先用 unsigned int 型变量 tmp 用语句 tmp＝TH0＊256＋TL0 取出 T0 中的数值，其后就是将 tmp 值不断地除 10 取整，这样将 int 型数据的各位分离并送入相应的显示缓冲区。

该实验程序文件名为 count_disp. c，在附赠光盘的 ch_10/count_disp 文件夹中，可用下载型实验板进行仿真实验。

三、单片机时钟

普通的家用时钟一般只能设置一个闹钟时间。很多人需为工作日、周末、早晨、午休等不同时段设置不同的闹钟时间，如果使用普通闹钟则很不方便，有时甚至会因忘记更改闹钟设置误事。针对这种情况，下面设计一款有 8 个闹时设置的时钟。硬件电路如图 10-34所示。

图中，74HC4511 是一种 BCD/7 段译码器，译码如表 10-11 所列。74HC4511 除了 BCD 码输入 D1、D2、D3、D4 外，另外还有 LE、、BI、LT 3 个输入端，它们的功能如下：

LE：锁定允许端。当 BI、LT 为“1”时 ，若该端为高电平，则加在 D1、D2、D3、D4 端的外部编码信息不能进 入译 码，所以译码器的输出状态保持不变；若 LE＝0，则 D1、D2、D3、D4 端的 BCD 码一经改变，译码器就立即输出新的译码值。

BI：消隐端。当 LT＝1 时，若该端为低电平，则译码器输出全为“0”，在一般应用中应接高电平。

LT：测试端。如果该端为低电平，则译码器输出全为高电平，该端拥有最高级别权限，只要它为“0”，即有上述现象，而与其余所有输入端状态无关。这一功能主要用于测试目的，因此正常使用中应接高电平。

表 10-11　7 段译码器 74HC4511 译码器真值表

输　入							输　出							显示
LE	BI	LT	D4	D3	D2	D1	a	b	c	d	e	f	g	
×	×	0	×	×	×	×	1	1	1	1	1	1	1	8
×	0	1	×	×	×	×	0	0	0	0	0	0	0	灭
1	1	1	×	×	×	×	锁存							维持
0	1	1	0	0	0	0	1	1	1	1	1	1	0	0
0	1	1	0	0	0	1	0	1	1	0	0	0	0	1
0	1	1	0	0	1	0	1	1	0	1	1	0	1	2
0	1	1	0	0	1	1	1	1	1	1	0	0	1	3

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

（续）

输入							输出							显示
LE	BI	LT	D4	D3	D2	D1	a	b	c	d	e	f	g	
0	1	1	0	1	0	0	0	1	1	0	0	1	1	4
0	1	1	0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	0	1	1	1	1	1	1	0	0	0	0	7
0	1	1	1	0	0	0	1	1	1	1	1	1	1	8
0	1	1	1	0	0	1	1	1	1	1	0	1	1	9
0	1	1	1	0	1	0	0	0	0	0	0	0	0	灭
0	1	1	1	0	1	1	0	0	0	0	0	0	0	灭
0	1	1	1	1	0	0	0	0	0	0	0	0	0	灭
0	1	1	1	1	0	1	0	0	0	0	0	0	0	灭
0	1	1	1	1	1	0	0	0	0	0	0	0	0	灭
0	1	1	1	1	1	1	0	0	0	0	0	0	0	灭

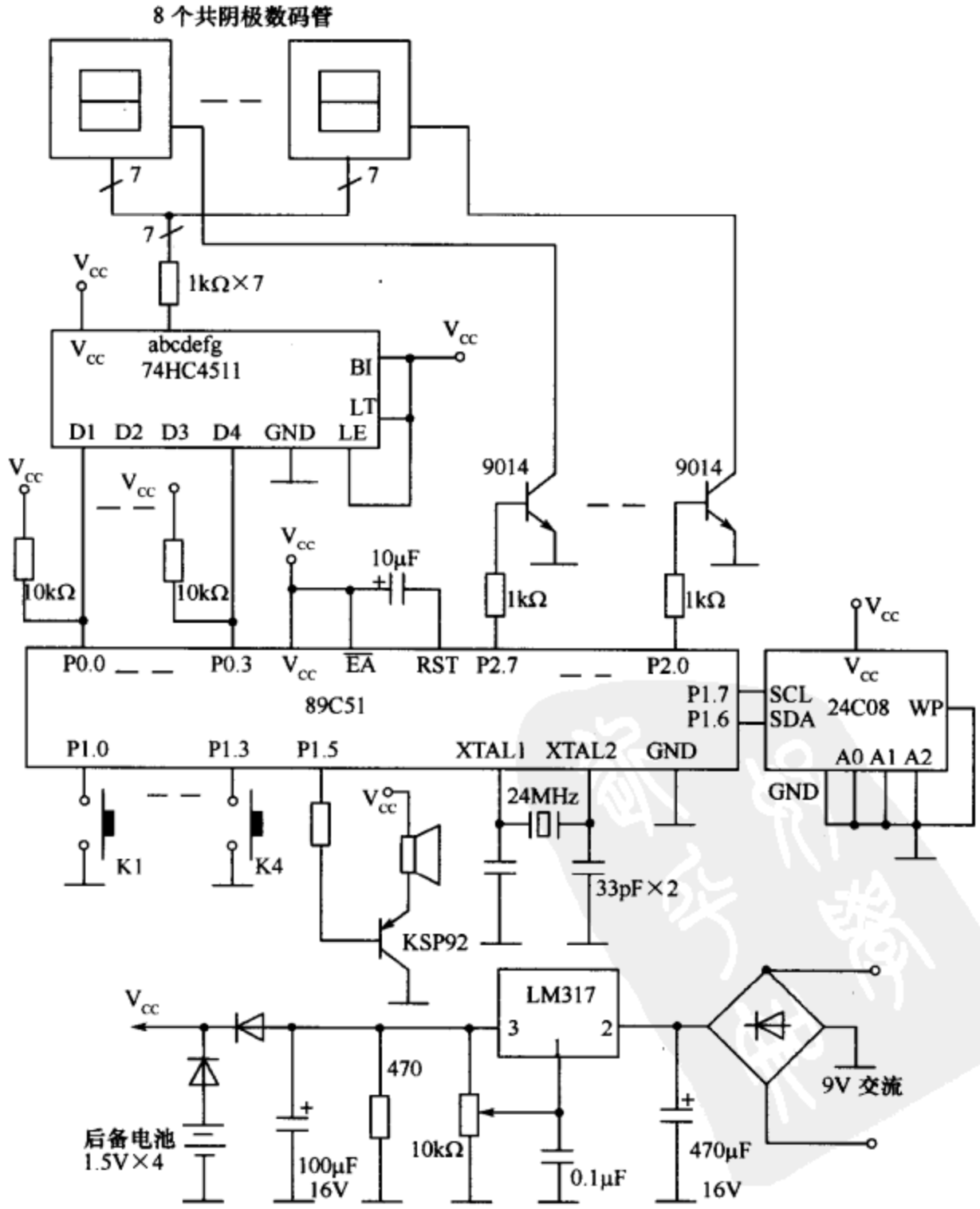


图 10-34　单片机时钟

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

该时钟以 24 小时制显示时间，还可显示 2000 年至 2049 年之间的任何日期及星期，按键可相互切换日期与时间显示；可设置 8 个闹时，每个闹时设置包括响铃的时间（小时与分钟）、对工作日有效还是对周末有效的标志，以及本项设置是否启用的标志等 3 部分。8 个闹时设置均保存在 EEPROM 中，即使掉电也不会丢失。使用者可通过按钮对任何一个设置做修改。可用按钮关闭显示，避免夜间刺眼，影响睡眠。调节 LM317 输出电压，可改变显示亮度，但电压不能低于后备电池的电压，否则系统将由后备电池供电。本时钟用 4 节 1.5V 电池串联作后备电源，保证市电停电时时钟继续走时。时钟的精度取决于晶振频率的精度。

该电子钟的程序用 C 语言编写，经 Keil Cx51 编译成二进制码后用编程器写入 89C51 内的 EPROM 内即可。源程序如下：

```
#include <AT89x51.h>
#include "intrins.h"
unsigned char hour,min,sec,year,month,day,weekday;   //当前时间、日期、星期
unsigned int count_down;          //1 秒钟计时用
bit led_on;                       //数码管是否点亮的标志
unsigned char display[8];         //8 位数码管要显示的数据
unsigned char attr;               //8 个数码管的闪烁控制字节，当为 0 时，对应数码管闪
bit flash;                        //LED 闪烁开关，与 attr 共同决定数码管是否闪烁
unsigned char show_status;        //LED 显示状态标志
                                  //0：设置闹钟数据
                                  //1：显示当前日期及星期
                                  //2：显示当前时间
                                  //3：设置当前日期
                                  //4：设置当前时间
bit km;                           //按键已去抖动标志
bit kp;                           //按键已处理标志
bit sound;                        //蜂鸣器响标志
bit alarm_stop;                   //蜂鸣器响后用户手工按停标志
struct{unsigned char h;           //小时
       unsigned char m;           //分钟
       }alarm[8];                 //定义结构数组，8 个闹钟项
unsigned char alarm_en;           //闹钟项启用标志
unsigned char alarm_wk;           //闹钟项周末启用标志
unsigned char cur_alarm_set;      //当前设置的闹钟项
unsigned char cur_alarm_active;   //当前到点的闹钟项
bit new_alarm_info;               //闹钟项内容已修改标志
sbit sound_output = P1^5;         //蜂鸣器驱动端口，输出 0 时蜂鸣器响
sbit SDA=P1^6;                    //EEPROM 数据线端口
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
sbit SCL=P1^7;                    //EEPROM 时钟线端口
//以下是 EEPROM 操作时需要的延时函数
void I2cDelay()
  {
  _nop_();_nop_();_nop_();
  _nop_();_nop_();_nop_();
  }
   //以下是延迟函数,参数为毫秒数
void DelayX1ms(unsigned char count)
  {
  unsigned char i,j;
  for(i=0;i<count;i++)
  for(j=0;j<240;j++);
  }
  //以下是 I2C 启动函数,24C08 使用 I2C 方式
void Start()
{
  SDA=1;
  I2cDelay();
  SCL=1;
  I2cDelay();
  SDA=0;
  I2cDelay();
  SCL=0;
}
//以下是 I2C 停止函数
void Stop()
{
I2cDelay();
SDA=0;
I2cDelay();
SCL=1;
I2cDelay();
SDA=1;
I2cDelay();
}
//以下是发送 1 字节数据给 EEPROM 函数
bit SendByte(unsigned char value)
  {
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
    unsigned char i;
    bit no_ack=0;
    for(i=0;i<8;i++)                    //发送 8 位数据
    {
    I2cDelay();
    if(value&0x80)   SDA=1;
    else      SDA=0;
    value=value<<1;
    I2cDelay();
    SCL=1;
    I2cDelay();
    I2cDelay();
    SCL=0;
}
    I2cDelay();
    SDA=1;                              //确认脉冲周期,等待 EEPROM 的确认
    I2cDelay();
    SCL=1;
    I2cDelay();
    if(SDA==1)   no_ack=1;
    I2cDelay();
    SCL=0;
    return no_ack;
    }
    //以下是向 EEPROM 写 1 字节函数
    void mywrite(unsigned char address,unsigned char value)
    {
    Start();
    SendByte(0xa0);                     //发送从地址
    SendByte(address);                  //发送子地址
    SendByte(value);                    //发送数值
    Stop();
    DelayX1ms(10);
    }
    //以下是从 EEPROM 接收 1 字节函数
    unsigned char ReadByte()
    {
    unsigned char i,bval;
    bval=0;
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
for(i=0;i<8;i++)                    //接收 8 位数据
{I2cDelay();
SDA=1;                              //从 P1 输入数据时,先往 P1 输入"1"
I2cDelay();
SCL=1;
I2cDelay();
bval=bval<<1;
if(SDA)   bval=bval|0x01;
I2cDelay();
SCL=0;}
I2cDelay();
SDA=1;                              //确认脉冲周期,不送出确认
I2cDelay();
SCL=1;
I2cDelay();
I2cDelay();
return(bval);
}
//以下是从 EEPROM 读入 1 字节数据函数
unsigned char myread(unsigned char address)
{
unsigned char tmp;
Start();
SendByte(0xa0);
SendByte(address);
Start();
SendByte(0xa1);
tmp=ReadByte();
Stop();
DelayX1ms(2);
return(tmp);
}
//以下是定时器 0 中断程序,用于走时,1/8000 秒一次
void Timer0ISR(void)interrupt 1 using 3
{
unsigned char tmp; unsigned int tmp_days;
count_down--;
if(count_down==1 || count_down==2001 || count_down==4001 || count_
down==6001)
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
{flash=~flash;                          //数码管闪烁的开关量
if(sound && flash)sound_output=0;       //驱动蜂鸣器
else sound_output=1;                    //关闭蜂鸣器
return;}
                                        //计算当前日期为星期几
if(count_down==3000)
{if(year==0)   weekday=5;               //2000 年 1 月 1 日为星期六
else {tmp=(year-1)/4+1;
tmp=(year-tmp)+tmp*2;
weekday=(tmp+5)%7;}                     //计算出当前年的 1 月 1 日为星期几
tmp_days=0;
for(tmp=1;tmp<month;tmp++)
if(tmp==1 || tmp==3 || tmp==5 || tmp==7 || tmp==8 || tmp==10)
tmp_days=tmp_days+31;
else if(tmp==4 || tmp==6 || tmp==9 || tmp==11)
tmp_days=tmp_days+30;
else if(tmp==2)
{if(year%4==0) tmp_days=tmp_days+29;
else   tmp_days=tmp_days+28;}
tmp_days=tmp_days+day-1;
weekday=(weekday+tmp_days%7)%7+1;
return;}
                                        //查询是否有闹钟时间项符合
                                        触发条件
if(count_down==5000)
{
if((alarm_stop || sound) && alarm[cur_alarm_active].m! =min) //触发后 1min
{alarm_stop=0; sound=0; }               //自动关蜂鸣器
if(sound==0 && alarm_stop==0)           //没有已触发的闹钟项
for(tmp=0;tmp<8;tmp++)                  //查询 8 个闹钟项内是否有符
                                        合条件的
{
if(((alarm_en>>tmp)&1)==0) continue;    //该闹钟项不启用
if(((alarm_wk>>tmp)&1)==1)              //该闹钟项周末有效
{if(weekday! =6 && weekday! =7) continue;}   //不是星期六或星期天
else
{ if(weekday==6 || weekday==7) continue;}
if(alarm[tmp].h==hour && alarm[tmp].m==min)//比较当前时间
{ sound=1;cur_alarm_active=tmp; break; }    //与该闹钟项的时间
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
}
return;
}
if(count_down==0)                                    //过了 1s
{
count_down=8000;
sec++;
if(sec==60)
{
sec=0;
min++;
if(min==60)
{
min=0;hour++;
if(hour==24)
{
hour=0;day++;
switch(day)
 {case 29: if(month==2 && year%4)  { day=1; month=3; } break;
case 30: if(month==2 && year%4==0) { day=1; month=3; } break;
  case 31:if(month==4 || month==6 || month==9 || month==11)
  {day=1; month++; } break;
  case 32: day=1;month++;
  if(month==13) { month=1; year++; }
  }
  }
  }
  }
  }
//以下是定时器 2 中断,用于按键扫描
void Timer1ISR(void) interrupt 3 using 2
{
unsigned char keytmp;
char tmp;
TH1=0x15; TL1=0xa0;      //每 30ms 中断一次
//当前显示的内容
if(show_status==0)          //当前正在设置闹钟项
{display[0]=cur_alarm_set;
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
display[1]=0xf;
display[2]=alarm[cur_alarm_set].h/10;
display[3]=alarm[cur_alarm_set].h%10;
display[4]=alarm[cur_alarm_set].m/10;
display[5]=alarm[cur_alarm_set].m%10;
display[6]=(alarm_wk>>cur_alarm_set)&1;
display[7]=(alarm_en>>cur_alarm_set)&1;}
if(show_status==1 || show_status==3)      //当前显示或设置日期
{display[0]=year/10;
display[1]=year%10;
display[2]=month/10;
display[3]=month%10;
display[4]=day/10;
display[5]=day%10;
display[6]=0xf;
display[7]=weekday;}
if(show_status==2 || show_status==4)      //当前显示或设置时间
{display[0]=hour/10;
display[1]=hour%10;
display[2]=min/10;
display[3]=min%10;
display[4]=sec/10;
display[5]=sec%10;
display[6]=0xf;
display[7]=0xf;}                          //最后两行无显示
                                          //按键扫描及处理
keytmp=~(P1) & 0x0f;
if(keytmp==0){ km=0;kp=0; }
else { if(km==0)   km=1;
else{ if(kp==0)
{kp=1;if(keytmp==1)                       //第一个按钮
{ if(sound) { alarm_stop=1; sound=0; }    //如果闹钟正响,按此键停
                                          止
else if((show_status==1 || show_status==2) && led_on) //正显示日期或时间
{ show_status=0;cur_alarm_set=0;attr=0x3f; }  //进入闹钟设置
else if(show_status==0)                   //如正在设置闹钟时间项
{ show_status=2;new_alarm_info=1; attr=0xff; }  //返回当前时间显示
return;
}
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
if(keytmp==2 && led_on)                                    //第二个按钮,仅当数码
                                                           管打开时有效
{switch(attr)
{ case 0xff: if(show_status==1) show_status=2;             //在显示时间与日期间切换
else if(show_status==2) show_status=1;break;
case 0x3f: if(show_status==0) cur_alarm_set=(cur_alarm_set+1)%8;
else if(show_status==3)
year=(year+1)%50;                                          //当前日期的"年"加 1
else if(show_status==4)
hour=(hour+1)%24;                                          //当前时间的"时"加 1
break;
case 0xcf: if(show_status==0)                              //闹钟设置的"时"加 1
alarm[cur_alarm_set].h=(alarm[cur_alarm_set].h+1)%24;
else if(show_status==3)
{month++;                                                  //当前日期的"月"加 1
if(month==13) month=1; }
else if(show_status==4)
min=(min+1)%60;                                            //当前时间的"分"加 1
break;
case 0xf3: if(show_status==0)
alarm[cur_alarm_set].m=(alarm[cur_alarm_set].m+1)%60;
else if(show_status==3)
{day++;                                                    //当前日期的"日"加 1
if(day==32) day=1; }
else if(show_status==4)
{count_down=8000;
sec=(sec+1)%60; }                                          //当前时间的"秒"加 1
break;
case 0xfd: if(show_status==0)
alarm_wk^=0x1<<cur_alarm_set;                              //周末标志位切换
break;
case 0xfe: if(show_status==0)
alarm_en^=0x1<<cur_alarm_set;                              //启用标志位切换
}
return;
}
if(keytmp==4)                                              //第三个按钮
{switch(attr)
{ case 0xff: if(show_status==1 || show_status==2)
```


资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
led_on=~led_on;                                          //打开或关闭数码管显示
break;
case 0x3f: if(show_status==0)                            //如果正在设置闹钟
{ if(cur_alarm_set==0)cur_alarm_set=7;
else cur_alarm_set--;}
else if(show_status==3)                                  //当前日期的“年”减 1
{ if(year==0)year=49;else year--; }
else if(show_status==4)                                  //当前时间的“时”减 1
{ tmp=hour-1;if(tmp<0) hour=23;else hour=tmp;}
break;
case 0xcf: if(show_status==0)                            //闹钟设置的“时”减 1
{ tmp=alarm[cur_alarm_set].h-1;
if(tmp<0)   alarm[cur_alarm_set].h=23;
else alarm[cur_alarm_set].h=tmp;
}
else if(show_status==3)
{ month--;                                               //当前日期的“月”减 1
if(month==0)month=12; }
else if(show_status==4)
{ tmp=min-1;                                             //当前时间的“分”减 1
if(tmp<0) min=59;else min=tmp;}
break;
case 0xf3: if(show_status==0)                            //闹钟设置的“分钟”减 1
{ tmp=alarm[cur_alarm_set].m-1;
if(tmp<0) alarm[cur_alarm_set].m=59;
else   alarm[cur_alarm_set].m=tmp;
}
else if(show_status==3)
{ day--;                                                 //当前日期的“日”减 1
if(day==0)day=31; }
else if(show_status==4)
{ tmp=sec-1;                                             //当前时间的“秒”减 1
 count_down=8000;
 if(tmp<0) sec=59; else sec=tmp; }
break;
case 0xfd: if(show_status==0)                            //切换周末标志
alarm_wk^=0x1<<cur_alarm_set;
break;
case 0xfe: if(show_status==0)                            //切换启用标志
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
alarm_en^ =0x1<<cur_alarm_set;
}
return;
}
if(keytmp==8 & led_on)                          //第四个按钮,仅当数码
                                                管打开时有效
{switch(attr)
{case 0xff: if(show_status==1)                  //如果当前显示日期
show_status=3;                                  //切换到调准日期状态
else if(show_status==2)                         //如果当前显示时间
show_status=4;                                  //切换到调准时间状态
attr=0x3f; break;                               //第一、二个数码管闪烁
case 0x3f: attr=0xcf; break;                    //第三、四个数码管闪烁
case 0xcf: attr=0xf3; break;                    //第五、六个数码管闪烁
case 0xf3: if(show_status==0)
case 0xf3: if(show_status==0) attr=0xfd;        //第七个数码管闪烁
       else if(show_status==3)
     { show_status=1; attr=0xff; }              //恢复正常显示日期
else if(show_status==4)
{ show_status=2; attr=0xff; }                   //恢复正常显示时间
break;
case 0xfd: if(show_status==0) attr=0xfe;        //第八个数码管闪烁
break;
case 0xfe: if(show_status==0) attr=0x3f;        //第一、二个数码管闪烁
}
}                                               //end of if(keytmp==4)
}                                               //end of if(kp==0)
}                                               //end of if(km==0)
}                                               //end of if(keytmp! =0)
}
//以下是主函数
main()
{
   unsigned char i;
   hour=23;min=58;sec=30;year=2;month=4;day=25;
   count_down=8000;
   flash=0;  attr=0xff;  led_on=1;
   km=0;  kp=0;  show_status=2;  //加电后显示当前时间
   new_alarm_info=0;  sound=0;  alarm_stop=0;
```


资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

```
for(i=0;i<8;i++)                                    //从 EEPROM 中读入 8
                                                    个闹钟设置
{ alarm[i].h=myread(i*2);alarm[i].m=myread(i*2+1); }
alarm_en=myread(i*2);
alarm_wk=myread(i*2+1);
IE=0;IP=0;                                          //禁止所有中断
TMOD=0x12;                                          //定时器 0 为模式 2,定
                                                    时器 1 为模式 1
TH0=6;TL0=6;                                        //定时器 0 为 0.125ms
TH1=0x15; TL1=0xa0;                                 //定时器 1 为 30ms
TR0=1;TR1=1;                                        //启动定时器 0 和定时器 1
ET0=1;ET1=1;EA=1;                                   //开定时器 0 中断、定时
                                                    器 1 中断和总中断
while(1)
{
  if(led_on)
  for(i=0;i<8;i++)
    { P2=0; if(flash || attr&(0x80>>i))
      { P0=display;P2=0x80>>i; DelayX1ms(1); }}
        else  P2=0;
        if(new_alarm_info)
        { P2=0;                                     //暂时关闭数码管
        new_alarm_info=0;                           //写入 EEPROM
        for(i=0;i<8;i++)
        { mywrite(i*2,alarm[i].h);mywrite(i*2+1,alarm[i].m); }
        mywrite(i*2,alarm_en); mywrite(i*2+1,alarm_wk);}
    }
}
```

该程序文件名为 clock.c,在本书附赠光盘的 ch_10/clock 文件夹中,读者可根据硬件电路制作实验板进行仿真实验。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

附录一　运算符的优先级和结合性

优先级	操作符	功　能	结合性
1 (最高)	( )	改变优先级	从左至右
	[ ]	数组下标	
	->	指向结构体成员	
	.	结构体成员	
2	++、--	增1减1运算符	从右至左
	&	取地址	
	*	取内容	
	!	逻辑求反	
	~	按位求反	
	+、-	取正数、负数	
	( )	强制类型转换	
	sizeof	取所占内存字节数	
3	*、/、%	乘法、除法、取余	从左至右
4	+、-	加法、减法	
5	<<、>>	左移位、右移位	
6	<、<=、>、>=	小于、小于等于、大于、大于等于	
7	==、!=	相等、不等	
8	&	按位与	
9	^	按位异或	
10	|	按位或	
11	&&	逻辑与	
12	||	逻辑或	
13	?:	条件运算符	从右至左
14	=、+=、-=、 *=、/=、%=、&=、 ^=、|=、<<=、 >>=	赋值运算符	从右至左
15(最低)	,	逗号运算符,顺序求值	从左至右

说明:同一优先级的运算符由结合方向确定,例如,* 和/具有相同的优先级,因此,3 * 5/4的运算次序是先乘后除。取负数运算符－和自加1运算符＋＋具有同一优先级,结合方向为自右向左,因此,表达式－i＋＋相当于－(i＋＋)。

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

附录二　Keil Cx51 编译器常见警告与错误信息的解决方法

1. Warning 280：'i'：unreferenced local variable
   说明：局部变量 i，在函数中未作任何的存取操作。
   解决方法：消除函数中 i 变量的宣告。
2. Warning 206：'Music3'：missing function-prototype
   说明：Music3()函数未作宣告，或未作外部宣告，所以无法给其他函数调用。
   解决方法：将叙述 void Music3(void)写在程序的最前端作宣告，如果是其他文件的函数，则要写成 extern void Music3(void)，即作外部宣告。
3. Compling：C：\8051\MANN. C
   Error：318：can't open file'beep. h'
   说明：在编译 C：\8051\MANN. C 程序过程中，由于 main. c 用了指令 # include "beep. h"，但却找不到所致。
   解决方法：编写了一个 beep. h 的包含档并存入到 c：\8051 的工作目录中。
4. Compling：C：\8051\LED. C
   Error 237：'LedOn'：function already has a body
   说明：LedOn()函数名称重复定义，即有两个以上一样的函数名称。
   解决方法：修正其中的一个函数名称，使得函数名称都是独立的。
5. * * * WARNING 16：NUCALLED SEGMENT，IGNORED FOR OVERLAY PROCESS SEGMENT：? PR? _DELAYX1MS? DELAY
   说明：DelayX1ms()函数未被其他函数调用，也会占用程序记忆体空间。
   解决方法：去掉 DelayX1ms()函数，或利用条件编译 # if..... # endif，可保留该函数并不编译。
6. * * * WARNING 6：XDATA SPACE MEMORY OVERLAP
   FROM：0025H
   TO：　0025H
   说明：外部资料 ROM 的 0025H 重复定义地址
   解决方法：外部资料 ROM 的定义如下：
   　　Pdata unsingned char XFR_ADC_at_0x25；其中 XFR_ADC 变量的名称为 0x25，请检查是否有其他的变量名称也是定义在 0x25 处并修正它。
7. WARNING 206：'DelayX1ms'：missing function-prototype
   C：\8051\INPUT. C
   Error 267：'DelayX1ms'：requires ANSI-style prototype C：\8051\INPUT. C

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

说明：程序中有调用 DelayX1ms( )函数，但该函数没定义，即未编写程序内容或函数已定义但未作宣告。

解决方法：编写 DelayX1ms()的内容，编写完后也要作宣告或作外部宣告(可在 delay. h 的包含文件声明成外部)，以便其他函数调用。

8. * * * WARNING 1：UNRESOVLED EXTERNAL SYMBOL
SYMBOL：MUSIC3
MODULE：C：\8051\MUSIC. OBJ(MUSIC)
* * * WARNING 2：REFERENCE MADE TO UNRESOLVED EXTERNAL SYMBOL：MUSIC3
MODULE：C：\8051\MUSIC. OBJ(MUSIC)
ADDRESS：0018H

说明：程序中有调用 MUSIC( )函数，但未将该函数的含扩文件(. C)加入到工程文件(. Prj)作编译和连接。

解决方法：(设 MUSIC3( )函数在 MUSIC. C 里)将 MUSIC. C 添加到工程文件中去

9. * * * ERROR 107：ADDESS SPACE OVERFLOW
SPACE：DATA
SEGMENT：_DATA_GOUP_
LENGTH：0018H
* * * ERROR 118：REFERENCE MADE TO ERRONEOUS EXTERNAL SYMBOL：VOLUME
MODULE：C：\8051\OSDM. OBJ(OSDM)
ADDRESS：4036H

说明：data 存储空间的地址范围为 0～0x7f，当公用变量数目和函数里的局部变量(如果存储模式设为 SMALL，则局部变量先使用工作寄存器 R2～R7 作暂存，当存储器不够用时，则会以 data 型别的空间作暂存)的个数超过 0x7f 时，就会出现地址不够的现象。

解决方法：将以 data 型别定义的公共变量修改为 idata 型别的定义。

资料来自网络，逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com

参 考 文 献

[1] 徐爱钧,彭秀华编著. Keil Cx51 V7.0 单片机高级语言编程与 uVision2 应用实践. 北京:电子工业出版社,2004.

[2] 马忠梅,刘滨,威军,马岩编著. 单片机 C 语言 Windows 环境编程宝典. 北京:北京航空航天大学出版社,2003.

[3] 胡伟,季晓衡编著. 单片机 C 程序设计及应用实例. 北京:人民邮电出版社,2003.

[4] 周坚编著. 单片机轻松入门. 北京:北京航空航天大学出版社,2004

[5] 田淑清主编. 计算机等级考试二级教程——C 语言程序设计. 北京:高等教育出版社,2002.

[6] 余文俊编著. 8051C 语言实习. 北京:水利水电出版社,2003.

[7] 周立功等编著. 增强型 80C51 单片机速成与实战. 北京:北京航空航天大学出版社,2003.

资料来自网络, 逆天PCB论坛仅整理收藏 http://bbs.ntpcb.com