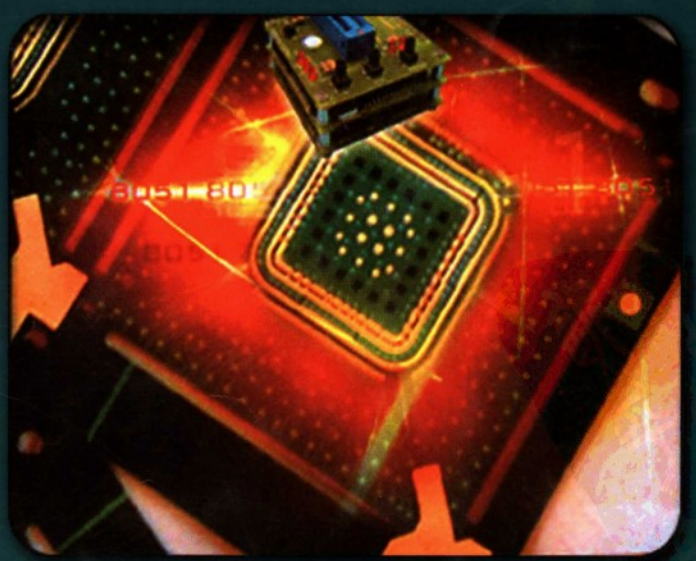
8051单片机技术应用系列

附范例源程序光盘

8051 单片机

彻底研究 基础篇

林伸茂 编著



中国电力出版社
www.infopower.com.cn

8051 单片机彻底研究 入门篇
8051 单片机彻底研究 基础篇
8051 单片机彻底研究 实习篇
8051 单片机彻底研究 经验篇

本书主要以旗威科技公司生产的 FLAG51 单片机控制板为描述主体，全面介绍了 8051 单片机的基础知识。全书共分为七大部分，分别为 8051 的由来与应用范围、8051 的指令与汇编语言的用法、AT2051 控制板的设计原理与用法、8051 单片机定时 / 计数与中断的应用、8051 的串行通信彻底研究、一系列 8051 的练习程序以及 8051 程序与相关仪器的使用。

本书选材的实用性和可操作性极强，范例丰富，文字叙述清楚，是 8051 单片机初学者的入门指南，对 8051 有一定基础的读者也具有较高的参考价值，非常适合作为高等院校学生做实验、专题制作、研究和设计单片机产品的专业参考书，同时也适合于广大单片机从业人员的学习使用。

- 8051 程序设计详解
- 8051 指令寻址模式
- 8051 基本程序练习
- 8051 控制线路说明
- LED 的接法
- 蜂鸣器的使用
- 指示灯的制作
- 七段显示器的设计
- 按键的特性与用法

责任编辑：李富颖
封面设计：杜长清

ISBN 978-7-5083-4686-1
9 787508 346861 >
定价：39.00 元（含 1CD）

8051 单片机

彻底研究 基础篇

林伸茂 编著

中国电力出版社
www.infopower.com.cn

内 容 提 要

本书主要以旗威科技公司生产的FLAG51单片机控制板为描述主体，全面介绍了8051单片机的基础知识。全书共分为七大部分，分别为8051的由来与应用范围、8051的指令与汇编语言的用法、AT2051控制板的设计原理与用法、8051单片机定时/计数与中断的应用、8051的串行通信彻底研究、一系列8051的练习程序以及8051程序与相关仪器的使用。

本书选材的实用性和可操作性极强，范例丰富，文字叙述清楚，是8051单片机初学者的入门指南，对8051有一定基础的读者也具有较高的参考价值，非常适合作为高等院校学生做实验、专题制作、研究和设计单片机产品的专业参考书，同时也适合于广大单片机从业人员的学习使用。

图书在版编目（CIP）数据

8051单片机彻底研究——基础篇 / 林伸茂编著. —北京：中国电力出版社，2007.1
（8051单片机技术应用系列）
ISBN 978-7-5083-4686-1

Ⅰ.8... Ⅱ.林... Ⅲ.单片微型计算机，8051—基本知识 Ⅳ.TP368.1
中国版本图书馆CIP数据核字（2006）第148752号

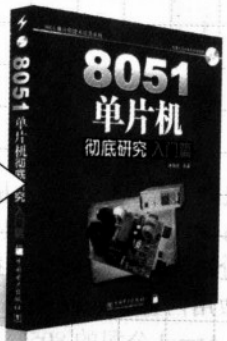
北京市版权局著作权合同登记号 图字：01-2006-5847号

责任编辑：李富颖
责任校对：崔燕菊
责任印制：李文志

丛 书 名：8051单片机技术应用系列
书　　名：8051单片机彻底研究——基础篇
编　　著：林伸茂
出版发行：中国电力出版社
　　地址：北京市三里河路6号　邮政编码：100044
　　电话：（010）68362602　传真：（010）68316497
印　　刷：北京市同江印刷厂印刷
开本尺寸：185 × 260　　印　张：22.75　　字　数：552千字
书　　号：ISBN 978-7-5083-4686-1
版　　次：2007年4月北京第1版
印　　次：2007年4月第1次印刷
印　　数：0001—4000
定　　价：39.00元（含1CD）

8051 单片机学习地图

8051 单片机的应用到处都是，可是我一点基础都没有，要怎样开始学习呢?



8051单片机
彻底研究——入门篇

我对 8051 已经有了基本的认识，我想更进一步彻底学好 8051 汇编语言!

8051单片机
彻底研究——基础篇

要想学好 8051，练习是少不了的，我希望找一些有趣的题目自己动手做做看，不然怎样写都是一些简单的范例。

8051单片机
彻底研究——实习篇

基本功都练扎实了，想进一步提高水平，吸取前辈的经验是最好的办法。

8051单片机
彻底研究——经验篇

序　言

让技术生根

10年前，我个人写了第一本有关8051单片机的书籍——《单片机8051彻底研究》，当时的硬件架构主要是由8051配合外部32KB或64KB的EPROM组成的，而内置4KB的EPROM的8751（或8KB的EPROM的8752）价格还比较昂贵。随着半导体制造技术的不断进步，现在我们看到以8051为主体的设计，都改用内置Flash程序空间的AT89C51（4KB）或AT89C52（8KB），这两种IC（集成电路）都是Atmel公司的8051兼容产品，与原厂Intel公司的8751和8752在使用上完全相同，而且它们可以用+12V的电压就可以清除内部的程序内容，而不像EPROM需要用紫外线超过30分钟的照射，才能清除内部的程序。Atmel公司的8051单片机兼容产品可以让系统设计者获得设计上极大的弹性，解决了在工具准备上的诸多难题。

在设计工作中，10余年来我也把8051单片机应用在许多工业设计上，保守地估计至少有百件以上。不论是在线路设计或是在程序编写上，个人深刻感受到了8051单片机的种种优势，尤其是数年前Atmel公司推出的AT89C2051，内置Flash存储器，只有20根引脚，程序空间为2KB，但是指令与8051完全兼容，除了串行通信的功能外，还有两个I/O端口（P1与P3，但是少了P3.6）。该芯片还有锁码，只要锁定后就无法再读回程序内容，这使得8051的硬件线路空间可以变得更小，应用范围更广，丝毫不比单片机的另一个竞争对手PIC差。

随着8051的专利权到期，半导体制造商可通过购买知识产权（IP）的方式，买到8051的整个微代码（Microcode，也有人称为Core），全盘理解后再加上其他特殊的控制电路，这就变成了CD-ROM的专用控制器。有人在8051的Microcode外，又加入高速的模拟—数字转换器（ADC），该IC就成为扫描仪专用的控制器，再对部分的硬件稍做修改就成为数码相机的核心控制单元，这也是近几年来光驱、扫描仪与数码相机价格持续下跌的原因之一。上述这些事情正在说明一个事实：除非你彻底弄懂，否则你只能做表面的修改而已。也唯有真正的技术生根，企业才得以存活下去，个人也要技术生根，否则就会被别人取代。当多数人在担心产业转移时，做技术的人真正担心的是我们有哪些技术真正生了根？

单片机系列书籍介绍

在写本书时，我们的视野与目标是相当开阔的。单靠一本书是无法把8051的指令与所有软硬件应用都交待清楚的，所以，我们规划了一系列的8051书籍供你参考：

（1）《8051单片机彻底研究——入门篇》：本书包括8051初学者需要的知识与基本程序范例。

（2）《8051单片机彻底研究——基础篇》：本书适合初学者阅读，重点在于对8051指令与架构的认识，以及8051系统的原理与基础设计。

（3）《8051单片机彻底研究——实习篇》：主要强调8051的系统扩充与整合应用，许多硬件的用法与设计技巧都在该书中加以介绍。

（4）《8051 单片机彻底研究——经验篇》：我们在 8051 应用经验的分享，其中包括各种烧录器的设计与程序共享等等，还有 8051 单片机的优缺点分析，这些都是相当宝贵的专业知识。

如何阅读本书

本书适合于 8051 单片机初学者阅读，我们期望初学者一定要动手去焊接，动脑去写 8051 的汇编语言程序，花点时间去研究别人写的程序，再花一点时间去推敲别人设计的电路，因为我们也是遵循这种法则，掌握相关专业知识的。

人要进步就要一直学习，在 8051 单片机的知识领域里也是如此，而充实知识最佳的方法为“做中学（Learning by Doing）”，唯有实际地做了才能学得更多。在本书中我们只提到两个实验用的装置：AT2051 控制板（面积相当小，只有 80mm×52mm）与 USB 烧录器，本书所有的范例都是以 AT2051 控制板为主要工作平台，当你弄懂 AT2051 控制板上所有的程序后，就代表你已经熟悉了单片机与数字控制的背景知识，而 USB 烧录器则只是一台烧录 AT89C2051 的工具而已，但是别忘了，它也是 8051 单片机的应用实例之一。

在随书光盘中，包括了本书中所提到的汇编语言范例程序，我们建议你每读一个章节就进行相关程序的验证，可以的话也把程序稍做修改，然后观察结果有何不同，这点是非常重要的！学习新知识的第一步是模仿，第二步是修改，这段时间可能会持续很长，第三步才是创新。若没经过模仿与修改的过程，是谈不上创新的，没有创新就达不到“技术生根”。我们由衷地希望读者学习 8051 单片机的心态是如此，自己装配时如此，写程序的时候如此，程序除错时也是如此。

本书的内容安排

本书共分成七大部分，每部分都有相关的探讨主题，阅读时可依读者本身的需要，选择合适的主题进行学习与操作。我们希望 8051 的初学者能循序渐进地学会本书各章内容，我们更希望这本书是学习 8051 单片机时，读者最常参考与翻阅的工具书。

第一部分：介绍了 8051 的由来与应用范围。

第二部分：讲述了 8051 的指令与汇编语言的用法，并以最简单的操作来建立单片机系统。

第三部分：介绍了 AT2051 控制板的设计原理与用法，知道相关的用法后你就可以开始着手写些简单的 8051 程序了。

第四部分：介绍了 8051 单片机最重要的定时/计数与中断的应用，这一部分一定要实际操作演练才能加深印象。

第五部分：8051 的串行通信彻底研究，我们在 AT2051 控制板上也设计有串行接口，我们强烈建议，如果你是 8051 的初学者的话，一定要详细读这部分内容。

第六部分：一系列的 8051 练习程序，从基础的 LED 应用，到高级的串行通信控制都有完整的示范程序案例，这部分有许多珍贵的宝藏正等待你的细心挖掘。

第七部分：8051 程序与相关仪器的使用，总有人会认为写程序怎么会用到仪器去检测，写程序不就是一连串的修改，不对再改就是了。可是如果有相关仪器的辅助，将可以大幅减少除错的时间，写程序可能不需要其他设备帮忙，但是学习 8051 单片机真地要“软硬都来”才行。

致谢

编写 8051 单片机系列书籍绝对不是一个人所能完成的，它绝对是一个团队工作的整合，3 年前我就开始筹备新书的出版事宜，所有的文章与内容经过整理过滤与调整补充，在这段整合的期间，我要特别感谢以下帮助我的人们：

姜莹贞小姐：初步整理已发表过的文章，光是校稿就校了很多次，并拍摄许多照片，让本系列的书籍得以完成初步的架构。

李浩蓁先生与曾琼惠小姐：进行本书版面调整与最后的校稿，整本书是在他们的手中完成的。

太克科技台湾分公司罗仕林先生及浩网科技公司的庄昱宏与黄芳川先生：提供最高级的示波器与逻辑分析仪，以及技术上的协助，让本书的图表资料与数据更有看头。

旗标出版股份有限公司的施威铭总经理与陈宗贤经理：对本书的章节安排与内容调整提供最好的建议，让本书得以对读者有最佳的学习效果与启示。

最后，我还是要谢谢家人所给予的鼓励，尤其是刚在牙牙学语的小女儿，没有他们几近狂热的激励与支持，就没有本系列丛书的问世。

旗威科技有限公司
林伸茂
chipware@chipware.com.tw

目　录

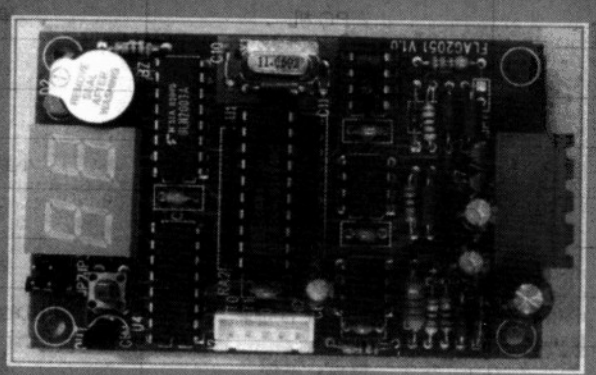
彻底研究篇

进阶练习篇

工具善用篇

入门篇

1



AT2051 控制板，程序空间可达 2KB 或 4KB，可作为学习用，也可做实际工业应用

第1章　单片机的来龙去脉

很多人认为学习单片机就是编写汇编语言的苦差事，其实只答对了一半。运用单片机所开发出来的产品，早已应用到现代人许多日常生活用品上。你用手机吗？手机本身就是单片机最好的表现形式，而充电器与电池上也有单片机存在。因此，不用心地去认识单片机是不行的！

1-1　单片机从头说起

你碰过单片机吗？先不要急着回答没有。其实它就藏在我们日常生活的常用设备中，而只是我们没有感觉到它的存在，例如：电子手表、掌上游戏机、数码相机、录音笔以及电视的遥控器等等。在一般 PC 机内部也有单片机，譬如网卡、键盘和鼠标的控制芯片等等。

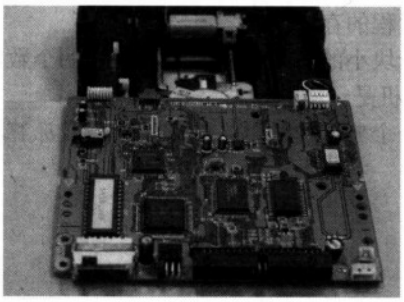
各种单片机的实例如图 1-1 所示。

1-2　单片机与个人电脑的比较

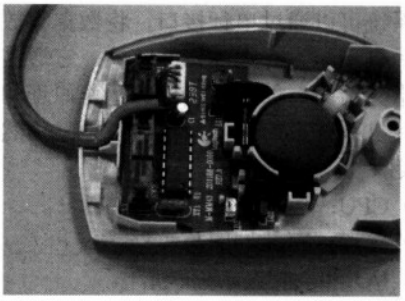
个人电脑（以下称 PC 机）可以说是一套完整的计算机系统，它的主板上有 CPU 和至少上百兆字节的内存，PC 机可外接显示器作为输出设备，而键盘与鼠标则是输入设备，磁盘驱动器与硬盘也可看成计算机另外的重要设备，它们负责保存数据。可是，在单片机的应用领域里，上述设备不是被简化，就是从成本或体积的考虑而被省略掉。

表 1-1　PC 机与单片机的比较

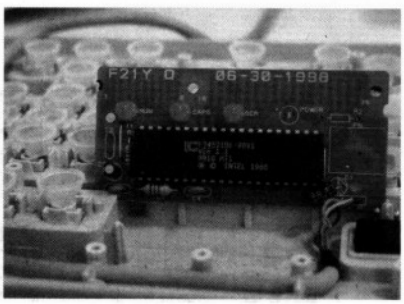
	PC 机	单 片 机
单　　价	高	低
功　　能	多	单一
体　　积	大	小到不能再小
内　　存	以百 MB 计	2～8KB
操作环境	Windows、Linux 等	自行开发
标准输出	CRT 或 LCD 屏幕	LED 或七段显示器
标准输入	键盘、鼠标	数个按键
常用的程序编程语言	VB 或 VC	汇编语言或 C 语言



（a）光驱上的单片机

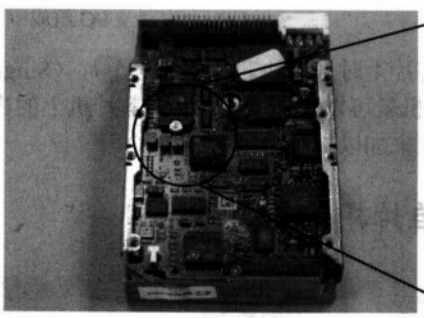


（b）鼠标上的单片机

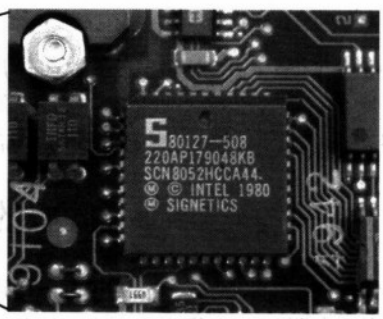


（c）键盘上的单片机

（d）遥控器上的单片机



（e）硬盘上的单片机



硬盘电路板的部分放大图

图 1-1　单片机的各种应用实例

单片机的系统没有类似 PC 机的显示器，当有显示的需要时，它会以简单的 LED（发光二极管）、七段显示器或体积不大的 LCD（液晶显示器）模块替代，单片机的显示信息只需要简单的界面就可以满足，而 PC 机的屏幕体积过大反而不方便。

单片机的系统里很少有 PC 机上的大键盘，因为单片机系统本身的体积就比键盘还要小，如果有输入的需要，通常会以几个简单的按键替代，只要稍做几个键的组合处理就可以达到单片机系统的要求。

单片机系统没有磁盘驱动器、没有硬盘，更不可能有 CD-ROM 光驱，当需要保存数据时，它会把数据存放在带电池的 SRAM（静态存储器）中，或是将数据存放在 E^2PROM（电可擦可编程只读存储器）中，这样就是关闭电源也不怕数据丢失。当有更多的数据要存放时，

单片机系统可以通过串行接口，将数据存放在远程的存储设备上。

单片机系统没有大体积的主板，它通常是一块小的单片机板，上面元件的个数非常少，构造也十分精简，但是它整体的工作原理和 PC 机是一样的。

单片机系统虽然如此简单，但是它仍然是一个典型的微机系统，具有组成计算机系统的 3 个要素：

（1）CPU：运算或逻辑上的判断。

（2）内存：存放程序与数据。

（3）I/O：与外界沟通的桥梁。

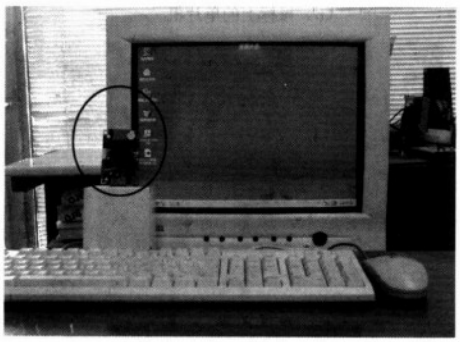


图 1-2　PC 机与单片机大小的比较

注：中间加圆圈者为单片机。

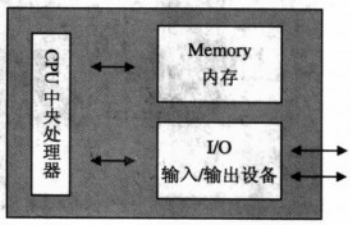


图 1-3　所有的计算机系统都由 CPU、内存及 I/O 组成

当我们同时把上述三要素都集中在一片芯片上时，这块芯片就叫做单片（Single Chip）机。只要加上电源及几个必要的元件后，单片机就可以独立工作了。当单片机上的内存或 I/O 不够用时，我们还可以外加其他的硬件或芯片来加以扩展。

1-3　典型的单片机应用系统

单片机组成的系统是无所不在的，家用电器（如微波炉、电磁炉、电视及智能型冰箱），汽车上的诸多控制单元（如 ABS 防抱死装置、电喷发动机控制器以及空调恒温控制器）和工厂里的许多控制设备都广泛地使用了单片机。在实际的应用中，单片机必须和外部的机电系统连接，以便得到必要的信息来实施控制。为了达到外界物理数据的获取及有效控制的目的，往往需要加上传感器（Sensor）及驱动（Driver）电路，因此单片机系统的设计，不可避免地必须包括硬件线路设计的部分。另一方面单片机为了能实现智能感应、控制与驱动的目标，自然也包括软件程序设计的部分。

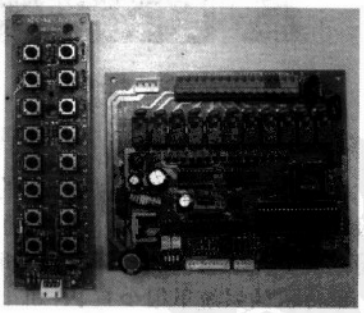


图 1-4　典型的单片机控制系统

注：左边是遥控设备输入，按钮较多但输出只有 3 个 LED，右边则是油电动机与电磁阀控制单元。

所以，当你接到一个单片机系统的设计项目时，基本上它是一个“系统”的设计，其中包含了软件、硬件以及整体的系统设计工作,这是所有单片机初学者必须了解的重要概念，如果无法以系统的眼光来看，那么就会使系统变得非常不稳定，最后的结果不是经常死机就是经常出现意外状况。

因此，一项单片机应用系统的设计开发过程，必须先思考并分析系统的需求，规划出哪些部分由硬件来做，哪些部分由软件来处理。先把硬件结构确定下来，然后把软件程序烧录到单片机中，最后把该芯片插入线路板，进行实际操作的检测，若有问题则需要再重头进行修正。

当程序所有的操作都确认无误后，单片机的系统设计者会认真地把硬件线路检查一次，进行成本费用与线路优化的修改。当然产品商品化的考虑也是重点之一，所以不可避免的软件程序也要再做一次确认，这样才可以把该项单片机的产品推到市场上。如果上市后碰到竞争者时，只有持续地优化硬件与软件功能，才能使该产品保持足够的优越性与竞争力。

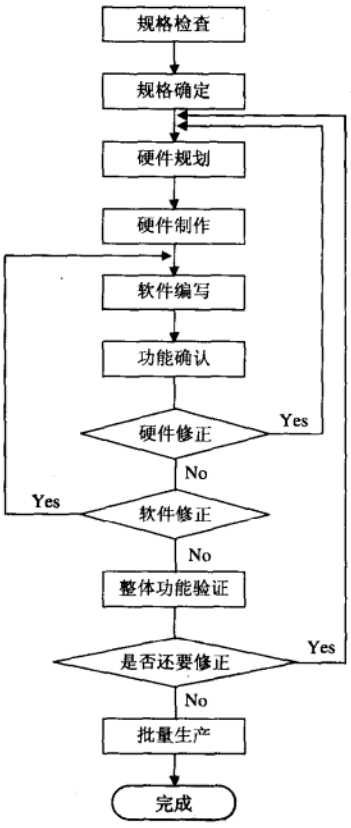


图 1-5　系统设计的流程图

1-4　最简化的单片机系统

单片机系统可以简化到什么程度？以本书所使用的 AT89C2051 为例，它是一块内置 2KB Flash 内存的 8051 兼容单片机。AT89C2051 共有 20 个引脚，只要在其 RST 脚上接一个电容，并且在 XTAL1 与 XTAL2 振荡脚加上石英晶体，最后把+5V 的电源接入，这样就是一个最简化的 8051 单片机系统了。

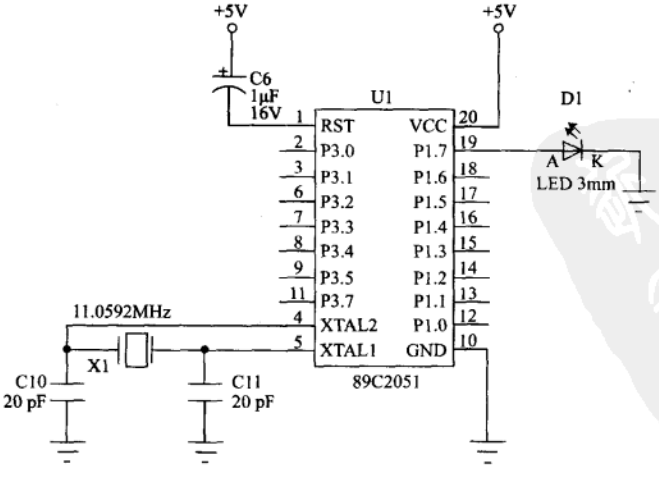


图 1-6　最简化的系统线路图

在图 1-6 上我们另外加上一个 LED 发光二极管，当程序运行时，就把其所有的操作结果用 LED 的亮、灭或闪烁来表现。如果你有实验用的“面包”板，只要花 5 分钟的时间就可以把这个电路接好，然后把随书光盘中的程序文件 Prog0-2.tsk 烧录到 AT89C2051 单片机上，并把该单片机插入“面包”板上，确定电源无误接上后，就可以看到 LED 正依照我们设计的速度在点亮与熄灭。

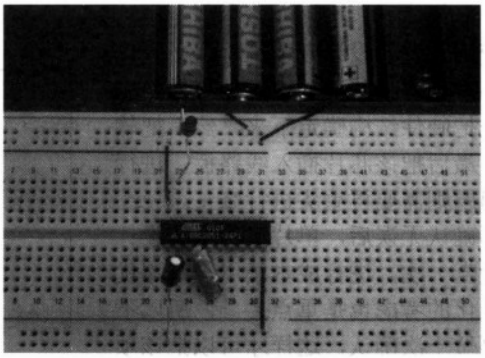
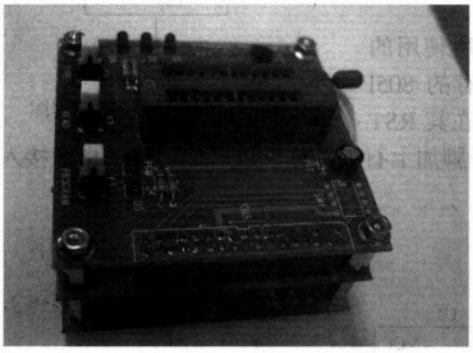
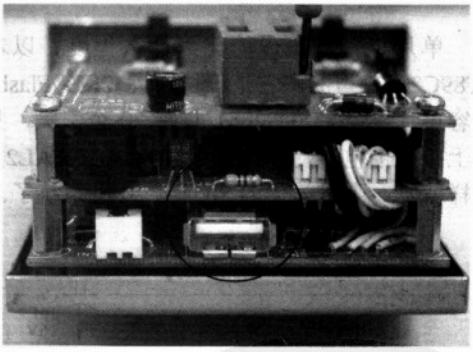


图 1-7　用“面包”板接出最简化的 8051 单片机系统

我们可以选用合适的烧录器来烧录 AT89C2051，本书中我们选用旗威科技公司的 USB 烧录器，该烧录器的体积小，随身携带方便，采用 USB1.1 接口与 PC 机连接，即插即用不需要外接电源，不占电源插座和不占桌面，十分好用。



（a）全貌图



（b）侧视图（可以看到 USB 接口）

图 1-8　旗威科技的 AT89C2051 烧录器

单片机系统的开发过程

一般单片机系统的开发流程是先完成硬件部分，接着开始编写程序的，如果要求执行速度很快而内存很小的话，那么汇编语言应该是首选。程序经过编译程序翻译成机器码的二进制文件后，就可以把该二进制文件由 PC 机发送给烧录器对单片机进行烧录，烧录完成后，把二进制代码烧入单片机后，再把该芯片放到已布好线等待测试的硬件线路板上，接上+5V 电源后看看程序执行是否如我们所预期的，如果不是，那么可能是程序某个部分有错误，必须重新修改程序才行。这时就要重新进入程序主体，判断哪一部分程

序出问题，接下来又是重新编译与重新烧录，再做一次测试。单片机的开发过程是由多次的“修修改改”构成的，当然其中也包括硬件的修正在内。这里所谓的多次对有经验的设计工程人员而言，可能是 3～5 次，而对正在学习 8051 的初学者可能就是数十次的修改了。

Prog0-2.asm 程序

```
        ORG     0000H
        MOV     R0,#00H
        DJNZ    R0,$
START   MOV     SP,#50H
LOOP    MOV     P3,#00H
        CALL    DELAY
        MOV     P3,#FFH
        CALL    DELAY
        SJMP    LOOP
;
DELAY   MOV     R0,#00H
$1      MOV     R1,#00H
$2      DJNZ    R1,$2
        DJNZ    R0,$1
        RET
```

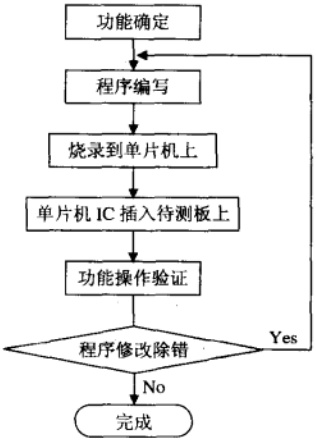


图 1-9　单片机系统软件开发流程图

1-5　单片机开发的实际问题

在单片机的开发过程中，我们印象最深的是程序的设计很少一次就成功，总是会有一些小修改在持续进行着，每次修改程序代码、编译程序、插拔芯片和烧录芯片总会耗掉许多宝贵的时间，所以仅是软件开发流程就要反复进行好几遍，如果再包括硬件的开发，那么工程就更大了。因此为了要让单片机系统的开发更方便、更有效率，人们研制出一系列的开发工具（Development Tool），主要有在线仿真器（In Circuit Emulator，ICE）、ROM 仿真器、可下载（Download）程序并具有仿真功能的单片机学习板以及具有实模式（非仿真）的单片机控制板等等。以下分别加以介绍：

（1）在线仿真器（ICE）：采用在线仿真时，必须把原来的单片机换成一块复杂的电路板（即 ICE），以替代原来单片机的所有功能，另外它还能配合硬件中断，强迫单片机暂停并检查所有寄存器的内容，当然它也可以检查或修改任一个存储器的内容，ICE 功能非常强大，因此价格也是最高的。

（2）ROM 仿真器：这类仿真器适合外接程序存储器的控制板，仿真器是插在系统的 ROM 接口上。我们随时可以把程序下载到该仿真器上，然后打开电源即可查看程序的执行情况。当发现程序不对时，只要重新下载程序即可，不需要每次都烧录 EPROM，进而有效减少开发的时间。不过，较新的单片机都属带有 4KB 或 8KB 的程序存储空间，真正用外部扩展存储器的机会相对就比较少了。

（3）单片机学习板：这是市面上最常见的单片机学习板，一般会有一个监控程序（Monitor Program），平常都是执行系统的监控程序，让用户可以很容易从 RS232 通信端口上下载我们

所设计的应用程序，当收到特别的指令时，再由监控程序切换到用户的程序，最后又返回系统的监控程序。该类型的学习板通常比 ICE 便宜，但是体积较大，仅适合用在学习阶段，较少用来做实际的应用。

（4）实模式控制板：这是我们最近一直提倡的学习方式，它本身就是一块真实的控制板，有足够的 I/O 端口与程序空间（最多 4KB），也具有与 PC 机通信的能力。当我们设计好程序，并将程序烧录到芯片中，就可以把该芯片插到控制板上，如果没办法运行，则表示软件一定有问题（因为控制板硬件已经完整验证无误了），反之，如果程序运行一切正确，则表示软件不需要做任何修正就可以复制进而销售。它可能没有 ICE 那么多的检查功能和在线除错功能，但是我们会在本书中告诉你许多设计上的技巧，例如如何只用一个 LED 或蜂鸣器来除错，如何通过数字式示波器来除错等等。只要学会这些除错技巧，以后不论碰到任何软、硬件问题，都可以迎刃而解。

综上所述，价格越高的单片机，系统的功能就会非常多，但是其操作上的复杂度也相对地提高。最近几年，单片机系统所使用的系统频率快了许多，所有的硬件时序特性也和之前的单片机明显不同，如果你使用早期的开发工具来编写程序，我们强烈建议一定要实际烧录一个芯片后，再进行一连串的实体测试。毕竟，模拟考试跟真正的大考还是有很多区别的。

表 1-2　单片机开发学习工具的特点

单片机开发学习工具	特　点
ICE	用 ICE 取代单片机，以掌握系统执行的每一细节及内容
ROM 仿真器	用 ROM 仿真器取代外接程序存储器，可以快速重新加载程序
单片机学习板	用监控程序来管理监督应用程序的执行
实模式控制板	让应用程序在真正的硬件环境下执行，因为使用 Flash 存储器，所以程序可以快速重新加载，学习、修改都十分方便

1-6　本书的单片机学习环境

学习任何事务都是要花费时间的，选对一个好的学习环境与工具绝对是必要的。在本书的所有范例中，以下的工具是必备的：

（1）PC 机：基本配备即可。

（2）单片机 8051 的汇编语言编译器：可购买合法的编译程序或从网络上取得类似的共享软件。

（3）AT2051 控制板或是类似能够验证程序的控制板，当然也可以自己焊接一块。

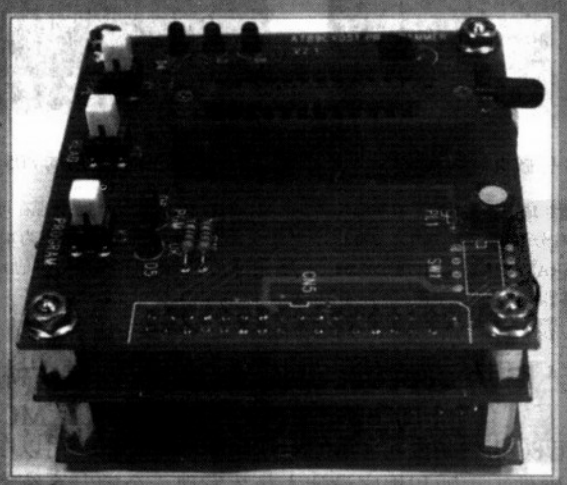
（4）可以烧录 Atmel AT89C2051 芯片的烧录器，旗威科技公司的烧录器（USB 接口）或市面上常见的万用烧录器。

学习就是不断地尝试和除错，当然学习本身是快乐的，而且写汇编语言绝对可称得上过五关斩六将，整个软硬件除错的过程非常辛苦，但是，只要看到你设计的单片机系统应用在各行各业中，再多辛苦与煎熬都是值得的！你准备好了吗，让我们开始学习单片机，开始学习 8051 吧！

习　题

1. 请画出开发单片机软硬件系统的完整流程图。
2. 什么是 ICE?
3. 什么是 ROM 仿真器?
4. 单片机学习板的监控程序有些什么用途?
5. 实模式控制板的好处在哪里?
6. 你认为设计单片机系统，除了熟悉单片机 CPU 指令之外，还必须具备哪些能力?

2



USB 烧录器，通过 USB 接线与 PC 连接，不需要外加任何电源，体积小，携带方便

第 2 章　8051 单片机简介

现在单片机的种类很多，而且功能越来越强，单片机的存储空间也增加了很多，可是单片机的设计原则和运行原理始终是不变的，在本书中是以 Intel 公司的 8051 单片机为主要介绍对象。熟悉了单片机的设计和制作技巧后，再去了解其他单片机是轻而易举的。

2-1　微型控制器与微型处理器

编写程序和设计硬件对学习者是很奇妙的，你看到的是“完全在自己掌握中”的设备。而掌握的核心就是 CPU，其中的程序就表现在机器的运行与反应上。早期的微机上的硬件分工是很细的，一套完整的系统分成 CPU、ROM、RAM 与 I/O 等单元，每个单元都有其独自运行的功能。随着半导体制造技术的成熟，I/O 与 ROM/RAM 等单元被整合到 CPU 中，这就是我们今天所称的单片机。由于这类单片机能独立执行内部程序，所以我们称这类单片机为微型控制器（Microcontroller），而不是微型处理器（Microprocessor）。

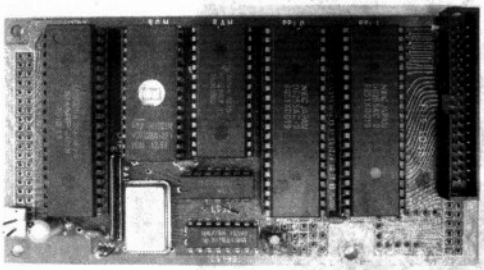


图 2-1　典型的 8 位微机控制系统

注：传统的 Z80 CPU 所组成的控制系统，本系统没有串行通信功能，由左到右分别是 Z80 CPU、27128 EPROM（16KB×8）、6264 SRAM（8KB×8）、两个 8255 I/O 接口以及下方的石英振荡器和地址译码电路。

图 2-2　整合后的 8051 单片机系统

注：图 2-1 所示的系统现在可以被整合到一块 8051 单片机芯片，下面的芯片，则为 4KB Flash 的 AT89C51，上面的芯片是比 8051 快 3 倍的 80C320。

2-2　时势造英雄：MCS-51 系列单片机

Intel 公司在 20 世纪 80 年代初发布了 MCS-51 系列的单片机，用以取代先前功能简单的 8048 与 8049 微控器，硬件工程师和程序设计人员才发现一个控制系统竟然可以简化到如此地步，其代表的芯片包括 8051、8031、8052、8032、8751 和 8752，我们统称为 8051 系列单片机，或称 51 系列单片机。这些芯片结合了传统 8 位 CPU 的很多优点，并将必要的 I/O 嵌

入到 CPU 中，除此之外，并增加了足够的 ROM 及 RAM 存储空间，使得单片机的线路变得非常简洁，Intel 的工程师希望用户只要加上石英振荡晶体及电源后，单片机系统就可以正式运行，事实上要让 8051 单片机运行也只需如此而已，用户只要将汇编语言程序写妥烧录到程序 ROM 内即可。不过，上面的例子仅适合非常简单的控制系统，真正的应用控制系统必须有多方面软件和硬件的配合才行。

表 2-1 是 Intel 公司的 MCS-51 系列单片机的比较表，这系列的单片机最早是以 HMOS 制造的，经过数年后，又以更先进的 CMOS 制造完成并增加了数项省电的功能。再经过数年后，单片机的架构已逐渐被设计工程师接受，不过此时 Intel 的发展方向却改向 80x86 高功能的 16/32 位 CPU，所以开始以授权的方式将 MCS-51 系列单片机交由其他 IC 制造厂生产，这就是所谓的 Second Source（芯片第二供应商），今天我们购买的 51 系列单片机厂商主要有: Philips、Siemens、Atmel 和 Intel，其主要特性都是相同的。

表 2-1　MCS-51 系列单片机的比较

型　号	制 造 工 艺	芯片程序空间	数 据 空 间
8051AH	HMOS II	4KB-ROM	128
8030AH	HMOS II	NONE	128
8751H	HMOS I	4KB-EPROM	128
80C51	CHMOS	4KB-ROM	128
80C31	CHMOS	NONE	128
8052	HMOS II	8KB-ROM	256
8032	HMOS II	NONE	256

Intel 公司在 MCS-51 系列的数据手册中，对由其开发出来单片机的特性说明如下：

（1）由 HMOS 或 CMOS 制程开发完成。

（2）内部有两个定时/计数器。

（3）两级中断优先等级。

（4）32 个 I/O 引脚，分成 4 个 8 位控制端口。

（5）64KB 的程序储存空间，Intel 称此段空间为程序空间（Program Memory）。

（6）64KB 的数据储存空间，这个空间是可以写入并读取的，Intel 称此段空间为数据空间（Data Memory）。

（7）8751 和 8752 单片机另外提供数据保护功能（Security Feature），可以防止程序内容遭到恶意的拷贝。

（8）所有系列单片机都可以进行单一位的布尔运算（Boolean Processor），这是传统 CPU 最弱的一个环节。

（9）提供位寻址数据区（Bit Addressable RAM）。

（10）可编程全双工串行传输接口（Full Duplex Serial Channel），可以同时进行发送与接收的串行通信。

（11）共有 111 个指令，其中有 64 个指令可以在一个机器周期内执行完毕，若系统使用

12MHz的石英振荡晶体时，单独一个机器周期只1μs的时间，较MCS-51系列的前身8048/8049系列单片机快了2～10倍。

图 2-3　51 系列单片机

注：所有的 IC 编号下方都有©Intel1980'标记，说明该 IC 是在 1980 年被开发出来的。

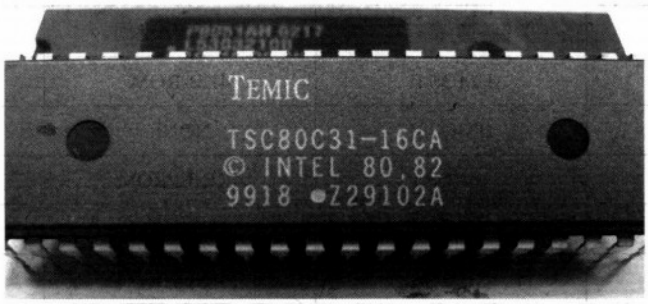


图 2-4　Second Source 的 8051 系列单片机

注：图中的单片机是 TEMIC 的 TSC80C31，其 IC 编号下方另外加注©Intel 80,82 标记，说明该 IC 采用的是 Intel 的微码(Microcode)，也有人称为 core。

2-3　8051 单片机功能方框图

8051 的芯片中有多少种功能呢？图 2-5 是简化后 8051 的主要单元的方框图。

1. 振荡及时序单元（OSC）

8051 内部有晶体振荡电路，只要在外部加上石英振荡晶体，即可产生频率非常稳定的振荡信号，这个振荡信号正是 8051 的心脏，所有 8051 的时钟序列都以此振荡信号为基准，由图中我们也可以看到该单元将提供 8051 CPU 必要的各种信号。

2. 内部数据存储器（Data Memory）

当程序运行时，有些数据是经常变动的，例如 LED 的显示状态或显示值等，它会因 I/O 测量的结果而变动，这些值就暂时放置在此区域中，供其他程序进一步读取，这也代表此区域是可以随时读写的。8051 系列的 CPU 提供 128 个字节可读写的数据存储单元，52 系列的 CPU 则提供 256 个字节的数据存储区，这个数据区域中有一段区域 16 个字节共 128 位是可以进行单一位寻址的，MCS-51 系列的 CPU 有相当强的位处理指令，可以妥善并有效地运用该数据存储器，使用方法将在 8051 的指令说明中再详述。

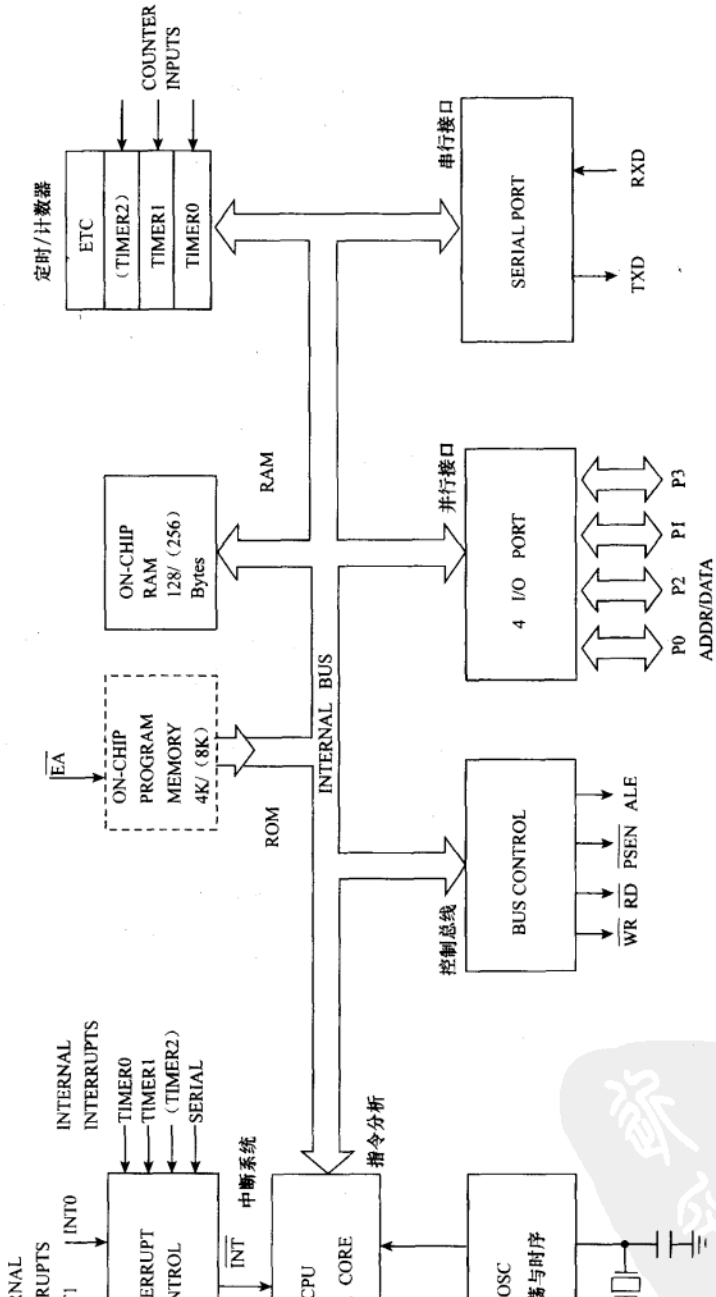


图 2-5　8051 内部的方框图

3. 内部程序存储器（Program Memory）

这段区域用于存放我们的应用程序，而且这部分一经确认后是永远不做修改的。8051 系列的 CPU 提供内部 4096 个字节（4KB）的程序存储器，8052 系列的 CPU 则提供两倍即 8192 个字节（8KB）的程序存储器，而 8031 和 8032 则不含此单元。如果有此单元时,CPU 可以选择由内部的程序区启动或由外部的程序区启动，前者的做法具有保护功能，后者成本较低。在 8051 系统中特地将程序区和数据区分隔开，前者在 Intel 的术语中称为 PROGRAM MEMORY（ROM），后者称为 DATA MEMORY（RAM），两者最大的差异是后者可以读取和写入。

4. 定时/计数器（Timer/Counter）

51 系列的 CPU 有两个定时/计数器，而 52 系列 CPU 则有 3 个定时/计数器，每个定时/计数器又有多种模式可供选择，详细功能将在稍后的章节中进行介绍。

5. 串行接口（Serial Port）

8051 可通过此接口与外部的计算机等设备连接交换信息，也可以通过此接口进行 I/O 的扩展，详细的设置与操作将在稍后的章节进行说明。

6. 并行输入/输出端口（Programmable I/O）

不论是 8051 或 8052 单片机都有 4 个输入/输出端口，总共有 32 个输入/输出线，而且每个点都可以单独定义成输入或输出。

7. 控制总线（Bus Control）

当程序的空间超出 MSC-51 系列内部程序空间的限制时，会通过本单元的控制线路向外部送出地址线信号和控制信号，同时当程序执行 MOVX @DPTR,A 指令时，代表对外部数据存储器做写入的操作，此时也要靠本单元送出必要的控制信号，才能实现外部 READ 与 WRITE 的操作请求，关于这方面进一步的时序分析，请参考本系列书籍实习篇的“时序彻底研究”。

8. 8051 运算处理单元（Core）

这是整个单片机的控制处理核心，它读取程序码，经过计算及处理后，将结果送到各个寄存器或输入/输出端口上，并且接受内部和外部的中断信号，然后执行特定的中断服务程序。只要加入电源并且振荡线路开始工作后，本单元就一直不停地工作着，通常我们所谓的死机是指本单元跳入一个未知没有出口的循环中执行，而不是指 CPU 停止一切的运行，不再执行任何程序，这一点请特别留意。

图 2-6 是 8051 的内部单元详细图示，它比图 2-5 更能表现 51 系列单片机的运行流程。主要表现各个寄存器和其他控制单元之间的关系，如果对这个单元能彻底理解，那么对 8051 单片机的架构就会有更进一步的认识。在这里我们将对图中的单元功能做简略的说明。更深入的介绍，请看本书后面各章的彻底研究说明。

9. 程序状态字（Program Status Word, PSW）

PSW 寄存器内存最近一次运算后的程序状态值，这些值包括：

（1）CY：进位标志位

（2）AC：辅助进位标志位，供 BCD 运算用

（3）F0：供用户自由运用的标志位

（4）RS1 和 RS0：寄存器组的选择位，总共有 4 个 Bank 可供选择

（5）OV：溢位标志位

（6）P：奇偶标志位

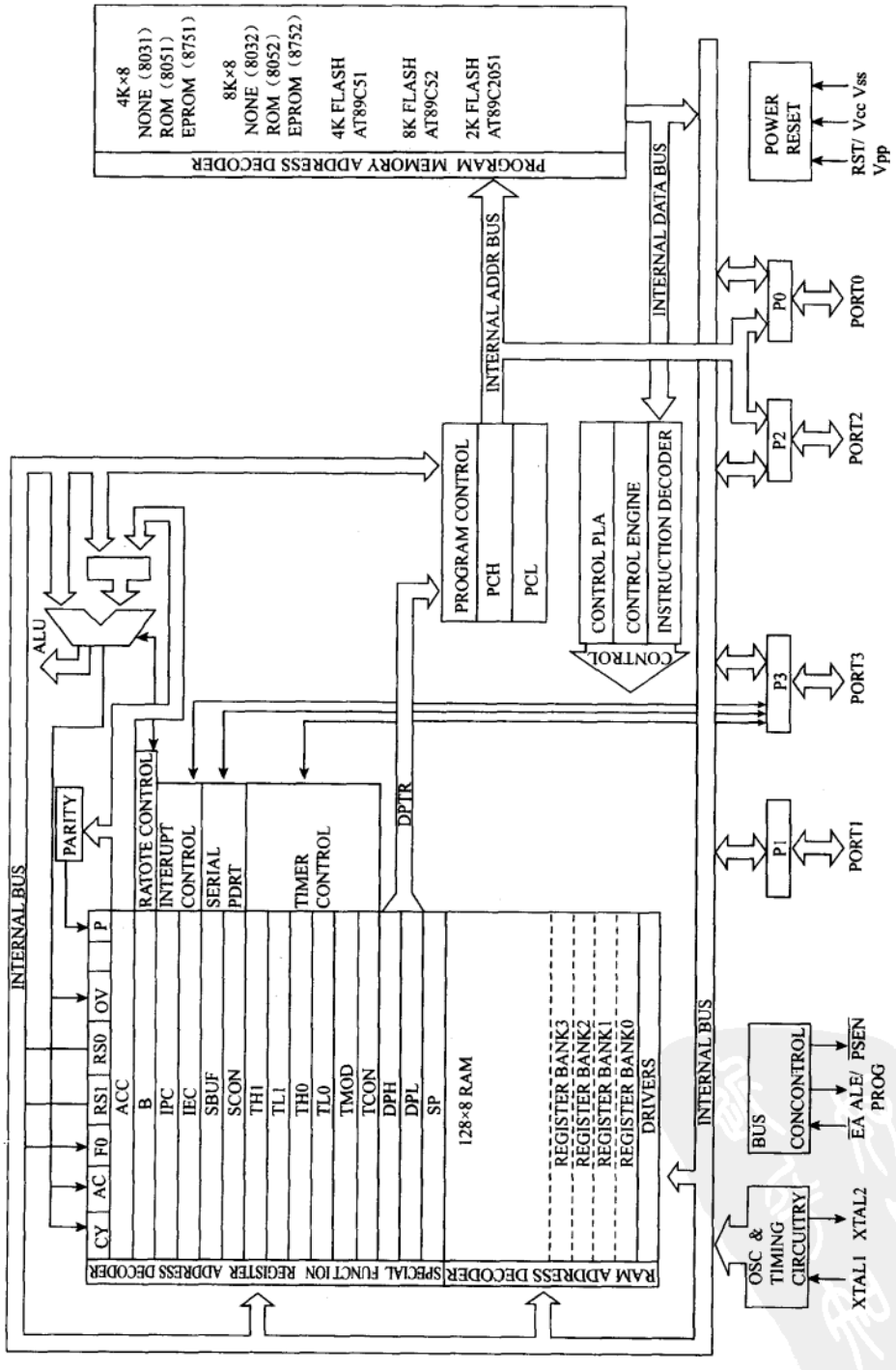


图 2-6 8051 单片机内部详细方框图

综上所述，每次算术逻辑单元处理后，会将 CY、AC 和 OV 标志位更新，而 P 标志位则在其右边的同位计算电路中将累加器 ACC 值处理后才得到更新值，另外 F0 标志位是随程序而变，RS1 和 RS0 一经设置后会引导 CPU 选用 4 个寄存器组中的其中 1 个。

10. 累加器（Accumulator）

累加器是众多寄存器中最重要的一个寄存器，通常以简写 ACC 代表累加器，8051 的指令中有许多指令和 ACC 寄存器有关，也有多个指令非通过 ACC 不可，基本上，8051 的指令中赋予 ACC 累加器的权限最大，Intel 公司建议程序的执行尽量以 ACC 为主。

11. B 寄存器（B Register）

B 寄存器是一个一般用途的工作寄存器，当 8051 使用乘除指令时，则一定要通过 B 寄存器来做运算。

12. IPC 中断优先顺序控制寄存器（Interrupt Priority Control）

这个寄存器中存放中断时的优先顺序表，若对应的位设成 1 时，代表该中断有较高的中断优先权。

13. IEC 中断使能控制寄存器（Interrupt Enable Control）

此寄存器内置系统允许中断的中断源设置值，8051 共有 5 个中断可供选择，8052 则有 6 个中断源，IEC 寄存器内中的位 7 称为 $\overline{EA}$ 位，若 $\overline{EA}$ =0 时就禁止系统所有的中断要求。

14. SBUF 串行输入/输出缓冲寄存器（Serial Buffer）

所有待送出或刚进入的串行数据值都存放在此寄存器中，8051 的串行通信是非常简单的，只要一设置完通信协议后，再执行一个 MOV SBUF, A 指令，就可立即将并行数据值转化成串行数据送到外部。

15. SCON 串行通信控制寄存器（Serial Control 或称 UART）

此寄存器主要在设置串行通信的模式，当串行数据已经送完或数据已收妥时，会有对应的位被设置成 1，这些位也可以当成中断要求信号，请求 CPU 执行特定的串行中断服务程序。

16. TH1/TL1 和 TH0/TL0 定时/计数寄存器

TH1 与 TL1 组合成一个 Timer1，TH0 与 TL0 组合成 Timer0，当程序允许定时/计数器运行时，上述的寄存器中存放着最近之一的定时/计数值。8051 及 8052 单片机的计数器都是增值型的（Up Counter），当数到最大值时会产生一个定时/计数溢位的标志位（Timer Overflow），此标志位也可以当成一个中断要求信号，请求 CPU 做中断服务。

17. TMOD 定时/计数模式设置寄存器（Timer/Counter Mode）

这个寄存器上存放定时/计数器的模式设置值，定时/计数器运行前必须先定好 TMOD 寄存器的内容，否则定时/计数器是不会正常运行的。

18. TCON 定时/计数控制寄存器（Timer/Counter Control）

这个寄存器可以控制定时/计数器的打开或关闭，若一经打开且计数到溢位时，TCON 上就有对应的位被设成 1，CPU 必须针对此位的状态，决定是否重新设置定时/计数值。

19. DPH 和 DPL 数据指针寄存器（Data Pointer）

DPH 和 DPL 共组成一个 16 位的数据指针 DPTR，8051 单片机中只有一个 16 位的地址指针，所以程序中 DPTR 的使用率很高。例如，要得到一个程序码（MOVC 指令），或者对外部数据空间执行存取操作（MOVX 指令）时，都需借助 DPTR 指针。

20. SP 堆栈指针寄存器（Stack Pointer）

8051 利用 SP 指引最近一次存入堆栈内的地址，每当我们在程序中调用其他子程序时，

原程序的返回地址就会自动存入内部 DATA MEMORY 组成的堆栈（Stack）中，而当子程序执行到 RET 指令时，CPU 会自动由堆栈中取回原先存入的返回地址，继续执行原程序。每当 CPU 将 8 位值存入堆栈时，我们称之为 PUSH（推入），这时 SP 值会增加 1，反之由堆栈中取回 8 位值时，则称之为 POP（提回），此时 SP 值会减少 1。

我们写 8051 单片机的控制程序时，在程序起始状态阶段一定要设置 SP 值，以便程序有足够的堆栈空间，也可以利用软件程序随时灵活调整堆栈指针 SP 的值。

21. Port3、Port2、Port1 和 Port0

这 4 个端口共提供 32 条 I/O 线与外界交换信息，所有的端口都可以进行字符输入/输出（Byte I/O）或者是单一位的输入/输出（位 I/O），当 8051 要进行外部空间扩展时，必须靠 Port2 和 Port0 送出地址和数据值，另外由 Port3 产生必要的控制信号，如读出（Read）和写入（Write）信号等。

22. 特殊功能寄存器区（SFR，Special Function Register）

8051 单片机内部将多个寄存器统称成 SFR，代表其特定的功能，甚至 Port0、Port1、Port3 也都是属 SFR 的成员之一，在这些 SFR 中有部分的寄存器可以进行位寻址，以下是这些 SFR 寄存器的整理，其中加（*）记号的寄存器可进行位寻址。

算术运算寄存器：ACC（*）、B（*）、PSW（*）

指针类寄存器：SP、DPL、DPH

并行输入/输出端口：P0（*）、P1（*）、P2（*）、P3（*）

中断控制寄存器：IP（*）、IE（*）

定时/计数寄存器：TMOD、TCON（*）、TL0、TH0、TL1、TH1

串行通信寄存器：SCON（*）、SBUF

23. 内部程序存储器（Internal Program Memory）

MCS-51 系列的 CPU 中，8051 和 8751 内部芯片中规划 4KB 的程序存储空间，而 8052 和 8752 则有 8KB 的内部程序空间，8031 和 8032 则缺少这一空间，此时一定要读取外部的程序空间。内置 ROM 的单片机也可以将其 $\overline{EA}$ 接低电位，强迫 CPU 启动后就直接读取外部 ROM。

24. 内部数据存储器（Internal Data Memory）

MCS-51 系列 CPU 中，共规划有 128 个字节的空间给程序使用，这个区域中有一小段特别区（20H～2FH）共 128 个位是可以做位寻址的，而此区前端（00H～1FH）则划分成 4 个寄存器组（R0～R7），可以由 PSW 寄存器上的 RS1 和 RS0 位，决定系统取用哪一个 BANK，用户真正能使用的数据区为 30H～7FH，不过系统堆栈区还会占用部分的空间，假设系统启动后，设置 SP 堆栈区由 60H 开始时，则剩下的可使用空间（30H～5FH）就显得非常有限了。

单片机系统的启动

MCS-51 系列单片机中，系统本身拥有 CPU、内部 ROM、内部 RAM 和输入/输出控制端口，的确是加上电源后就能独自运行，当系统收到复位 RESET 信号后，会将程序计数值设成 0000H（即 PCH=00H、PCL=00H），然后送出 PC 程序计数值给内部的 ROM 地址输入点，并且读入一个程序码，接着通过指令解码器得知指令的种类和长度，然后依指令码的指示调整各个 SFR 的内容或是数据区的值，周而复始不停地执行预先存入 ROM 中的程序。

2-4 8051 系统复位分析

系统复位（RESET）对任何一个微处理电路都是相当重要的环节，当 RESET 信号发生后，所有的数字电路都要恢复到一个已知或默认的状态，而 CPU 也是如此，它必须跳到系统控制程序的起始点上，等待 RESET 信号消失后，就开始执行程序。为了防止系统被非正常信号干扰而复位，通常 CPU 的 RESET 输入端都是设计成史密特触发电路（Schmitt Trigger），只有正确的复位信号才可以复位系统，8051 单片机的 RST 输入引脚也是如此。

当系统振荡电路已在运行，而且 RST 引脚维持在高电位超过两个机器周期共 24 个振荡周期后，8051 随即进入内部系统复位的状态，并设置原本是输出引脚的 ALE 及 $\overline{\text{PSEN}}$ 为输入引脚状态，8051 一直在此状态中等待，直到 RST 引脚降到低电位后，才开始执行内部或外部的程序。这里所谓的内部系统复位，是 8051 内部将 SFR 等重要寄存器设置成某个值，至于内部的存储器则不做任何设置，所以大部分的程序在起始后，就先进行数据存储器的清除操作，以免程序运行错误，以下就是 Intel 手册谈到在复位后，内部寄存器被设置的值：

（1）PC 程序计数器被设成 0000H，这意味着系统复位后程序是从 0000H 地址开始执行。

（2）ACC 累加器被清除成 00H。

（3）B 寄存器被清除成 00H。

（4）PSW 程序状态值被清除成 00H。

（5）SP 堆栈指针被设成 07H，这个值不是很妥当，它使系统的堆栈区安排在 08H～7FH，所以通常应将起始程序的 SP 值重新设成 50H 或 60H，以免整个内部数据空间都被堆栈区占去。

（6）DPTR 数据指针被设成 0000H。

（7）P0 端口被默认成 FFH，即输入的状态。

（8）P1 端口被默认成 FFH，即输入的状态。

（9）P2 端口被默认成 FFH，即输入的状态。

（10）P3 端口被默认成 FFH，即输入的状态。

（11）IP 中断优先顺序寄存器被设成 XXX00000B 的位状态。

（12）IE 中断控制寄存器被设成 0XX00000B 的位状态。

（13）TMOD 定时计数模式寄存器被设成 00H。

（14）TCON 定时计数控制寄存器被设成 00H。

（15）TH0 定时器 0 的高位被设成 00H。

（16）TL0 定时器 0 的低位被设成 00H。

（17）TH1 定时器 1 的高位被设成 00H。

（18）TL1 定时器 1 的低位被设成 00H。

（19）SCON 串行通信控制寄存器被设成 00H，但是 SBUF 串行通信缓冲器的值则未定。

（20）PCON 功率控制寄存器的设置分成两种情况：①采用 HMOS 的单片机规格时，PCON 被设成 0XXXXXXXB 的位状态；②采用 CHMOS 的规格时，PCON 被设成 0XXX0000B 的位状态，这是因为 HMOS 的 8051 芯片没有省电（Power Down）的模式设置位。

若是 8052 系列的单片机，其内部多了一组定时/计数器，所以起始设置值有若干的更改：

（1）IP 中断优先顺序寄存器被设成 XX000000B 的位状态。

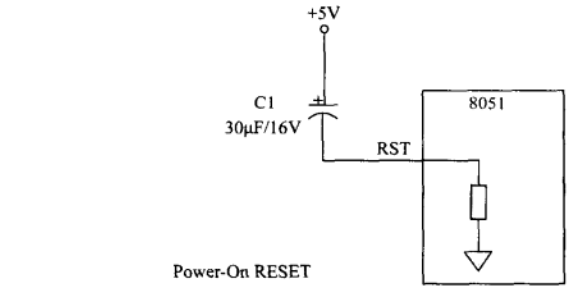
（2）IE 中断控制寄存器被设成 0X000000B 的位状态。

（3）T2CON 定时器 2 被设成 00H。

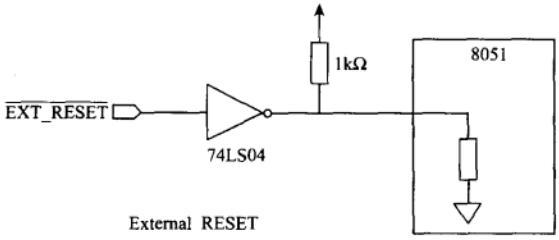
（4）RCAP2H 定时器 2 的特殊寄存器高位被设成 00H。

（5）RCAP2L 定时器 2 的特殊寄存器低位被设成 00H。

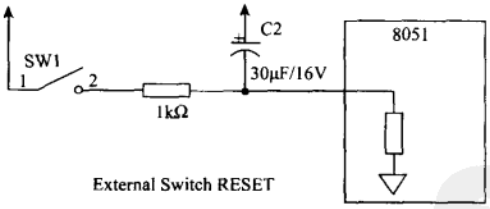
单片机 8051 的复位电路应该如何安排呢？Intel 手册中提供一个简易的电路，只要一个电阻和一块电容即可。在电源和 RST 引脚间并上一个 10μF 的电解电容，并在 RST 到 GND 间加入 8.2kΩ 的电阻即可运行，当电源刚加入时，10μF 电容两端没有储存电荷视同短路，所以 RST 引脚维持在高电位，但是电容开始充电，促使 RST 引脚逐渐降到低电位，此时系统就开始工作了。



（a）直接接一个电容到+5V，电容值不能太大，否则回影响RESET时间



（b）由外部送来RESET信号



（c）Power-On RESET外也可以用按钮强制RESET

图 2-7　几种常用的 RESET 线路

Intel 的手册也提到电源加到 8051 Vcc 端的电源上升时间（Rise Time）不可超过 1 ms，而且振荡电路的起始时间也不得超过 10 ms，否则 RST 引脚会运行错误。在实际应用电路中，图 2-7 接法的信赖度不高，造成系统 RESET 运行的不正常，所以也有人改用电压检测 IC 作为系统 RESET 的依据，只要系统的电源一低于 4.5V 时，电压检测 IC 就自动送出 RST 信号给 CPU，这种做法不仅信赖度高，而且系统供应电压一有异常时，RST 信号适时出现，可以让程序运行得更准确些。

表 2-4　8051 SFR 表与 RESET 后的初始值

F8H									FFH
F0H	B 00000000								F7H
E8H									EFH
E0H	ACC 00000000								E7H
D8H									DFH
D0H	PSW 00000000								D7H
C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 0000000	TL2 00000000	TH2 00000000			CFH
C0H									C7H
B8H	IP XX000000								BFH
B0H	P3 11111111								B7H
A8H	IE 0X000000								AFH
A0H	P2 11111111								A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	**AUXR'** **XXX00XX0**		8FH
80H	P0 11111111	SP 000000111	DP0L 00000000	DP0H 00000000	**DP1L** **00000000**	**DP1H** **00000000**		PCON 0XXX0000	87H

注：1. 只有最左侧的寄存器才能进行单一位寻址。

2. 有阴影的寄存器为 8052 新增的。

3. 灰色部分是 AT89S8252 所专有的。

习　题

1．单片机的定义是什么？

2．MCS-51 系列单片机有哪些？请列举 5 种。

3．MCS-51 系列单片机有哪些特性？请列举 7 项。

4．试述 8051 单片机有哪些主要功能。（提示：见功能方框图）

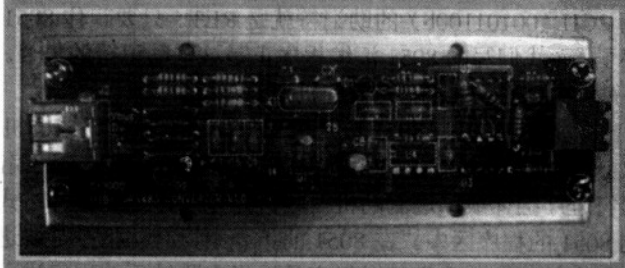
5．51 系列与 52 系列的 CPU 分别提供几个定时/计数器？

6．什么是 SCON？其别称是什么？

7．8051 提供多少个端口及多少条输入/输出线？如果扩展外部存储器时，必须使用哪两个端口送出地址及数据值？

8．试说明 8051 单片机如何系统复位。

3



分散式控制器具有 RS485 接口，如果要与 PC 连接就要有 USB 转 485 的转换电路板

第3章　单片机的汇编语言

熟悉了单片机的基础知识后，接下来我们学习 8051 的指令，而控制单片机操作的是一连串 8051 的指令，这些指令就是汇编语言。任何一种 CPU 的汇编语言都需要花时间去学习，我们深信学会汇编语言后，再学其他语言会变得很轻松。

3-1　8051 单片机的程序设计

决定一切单片机操作的应该是其内部的软件程序，不同的程序可以赋予 8051 单片机不同的角色。硬件线路赋予单片机可以运行的躯壳，而软件程序则提供了所有操作的依据。

十几年前的设计工程师受限于存储器空间的不足，程序设计时首先注重程序能够运行，接着是程序执行空间要小和执行速度要快等等。目前的程序设计仍然是先要求程序能够运行，接着则是执行速度要快和程序设计要思路清晰，这是因为经过这么多年的研究，许多线路和软件程序架构已经随手可得，我们只要稍做修改后可能就很适合我们的控制系统，而且各种存储器的价格也非常便宜，程序执行的空间绝对足够，唯一缩减的是除错时间。以我们的实际经验为例，一个硬件架构若花了两个月的设计时间，其软件的设计时间至少是硬件时间的 3 倍以上。如果在开发过程中详细记下各种文件数据，就能有效地节省除错及开发时间。

3-2　写汇编程序的预备知识

8051 单片机内部有一块程序区，其容量为 4KB，当电源打开后 8051 会自动从程序区 0000H 的地址开始执行程序，每次它送出地址线后读取 1 字节的机器码（Machine code），这个机器码进入 CPU 后会被当成有效的指令（Microcode），进行完运算后并进行寄存器或 I/O 的调整。每组机器码都是 0 与 1 的组合，如果要我们记住所有的指令的机器码几乎是不可能的。所以，我们会把机器码对应到的指令赋予一个特定的助记符。例如 00H（00000000b）其助记符即为 NOP，而 2CH（00101100b）的助记符就是 RET。只要记住 8051 的所有助记符就可以开始着手写程序了，而我们写的 8051 汇编程序正是这些助记符的集合。不管程序如何复杂，所有的 8051 程序本质上还是由这些有效的机器码组成。

人只看得懂 8051 的助记符，但是 8051 单片机却只懂由 0 与 1 组成的机器码，怎么办呢？因此，就有人开发出 Assembler 的汇编语言工具程序。Assembler 是一个汇编语言的翻译程序，它会把我们写的助记符程序翻译成二进制的机器码，并存入特定的文件格式中，以方便我们将这些机器码固化到 8051 的程序区中。在 8051 的指令中，有些指令是要进行往前或往后跳转的操作，通过汇编程序的帮忙，我们可以很容易地指定跳转后的程序进入点，若要用人工的方式来计算跳转点会比较费事的。

汇编程序能把我们所写的助记符程序翻译成一连串的机器码，它还会帮助我们找出程序

的问题处并且除错吗？答案是否定的。汇编程序仅能真实地翻译所有的助记符，至于对程序的写法及流程走向皆不做任何调整或警示，所有的结果都是我们的程序所定义的。如果程序有错，那么是我们使它出错的，与汇编程序无关。

在本书的稍后章节里就会提到一个重要的概念：Design for Test，也就是说所有我们做的设计，不论软件或硬件都要考虑到测试的问题。如果我们在写程序的初期就考虑到测试，那么系统就很容易除错了，不仅硬件线路如此，软件程序更是如此。

3-3 汇编语言的基本架构

汇编编译程序是学习 8051 单片机的必备工具，在市面上最常见的汇编程序应该是美国 2500AD 公司所开发的 8051 宏汇编程序，几乎所有 8051 书籍的程序范例都是采用 2500AD 所提供的格式，本书也不例外。另外，你也可以从 Internet 下载免费的汇编程序，其操作方式也和 2500AD 的汇编程序类似。

典型的 8051 汇编程序包括下列各项：

```
LABEL    MNEMONIC    OPERAND      ;COMMENT
TEST:    MOV         A,#A5H       ;把 Acc=A5H,可用中文做注释
标记     助记符      操作数       ;注释
```

LABEL（标记部分）可有可无，这是提供程序进入、跳转或分支的记号，程序经过汇编程序翻译后，所有的标记都变成真正的地址值。在某些场合中，加上标记可以让程序更容易阅读和分析。

MNEMONIC（助记符）则是一个操作指令，每种 CPU 的助记符都不同，以 8051 为例，MOV 代表数据传送，CPL 代表数据反相，我们必须完全了解各个 CPU 的助记符后，写程序才会得心应手。

OPERAND（操作数）则提供必要的数据给 CPU，以便数据处理时有所依据。例如，我们要累加器增加 10H，即 10 进位的 16，这时的写法就成为 ADD A，#10H，其中的#10H 就是操作数。

COMMENT（注释部分）则是对此行指令的用意做必要的说明，注释前一定会加上“;”记号，这是告知汇编程序不必理会“;”后的任何信息，继续编译下一行汇编语言，我们所写的程序中若有注释说明，经过汇编程序编译后产生的机器码中是不含任何注释的。8051 程序的执行是不需要任何注释的。但是加上注释可以让我们对程序有更进一步的理解，当我们在除错时，注释又可发挥其帮忙的加乘效果。如果我们写一段程序（约一两百行左右）并且不做任何注释，在 3 个月后再来看这个程序时，可能当初的构思与想法都忘了，这时你必定能同意程序加上一些必要的注释绝对是必需的。

汇编程序除了能将助记符编译成机器码外，还有其他额外的处理能力，以方便我们完成一个完整的汇编程序。请看以下的程序范例：

范例 1：标记的替换

```
SRAM_2K  EQU 8000H            ;这一行不是 8051 的指令,而是指定 SRAM_2K=8000H
         MOV DPTR,#SRAM_2K    ;这行被汇编程序解读成 DPTR=8000H
         MOV DPTR,#8000H      ;这行与上一行的结果完全相同
```

既然上两行的结果完全一样，可是方便性还是有所差异的，当硬件 SRAM 区由 8000H 改

为 9000H 时，我们的 SRAM 地址定义这一行只需要改成 SRAM_2K EQU 9000H 即可，汇编程序会自动把程序中有关 SRAM_2K 的地方全部置换成 9000H。第二种写法就没有如此方便了。

EQU 对汇编程序而言是一个标志位（Directive），也有人称之为伪指令（Pseudo-opcode），它可以让我们更方便地应用标记符号去替换真正的地址值。

范例 2：窗体属性的设置

```
;A=0 则取到 B1H
;A=1 时取到 A2H
;A=2 时取到 C3H
        MOV     DPTR,#TABLE         ;定义 TABLE 的进入点
        MOVC    A,@A+DPTR           ;取得 TABLE 内的值
        LJMP    NEXT
;
TABLE DB        B1H,A2H,C3H         ;这一行不是 8051 的指令,但汇编程序
                                    ;会产生 3 字节充做程序执行时查表的依据
```

DB（Define Byte）是我们写汇编程序时常使用的伪指令，尤其是通过查表的方式取得某些特定值时，必定会用 DB 来做值的设置。另一个常用的伪指令为 DW（Define Word），一次可以定义 16 位的值。

范例 3：可位寻址的使用

8051 单片机上有 128 位的可位寻址区，分别是 20H.0～2FH.7，如果我们运算完的进位（Carry）要存放到 20H.7 位上，我们的汇编语言可以写成这样：

```
SAVE        MOV     20H.7,C         ;20H.7=Carry bit
            JB      20H.7,NEXT      ;如果 20H.7=1,程序就跳到 NEXT
                                    ;处也可以写成以下的样子:

SAVE_CY     REG     20H.7           ;这一行声明 要放在程序最前面

SAVE        MOV     SAVE_CY,C       ;20H.7=Carry
            JB      SAVE_CY,NEXT    ;如果 20H.7=1,程序就跳到 NEXT 处的地址
```

从上面的写法看来，当然是第二种写法较具有弹性，而且可读性也较高。

写汇编程序能够训练一个人具有更周密的思路和处理能力，由于汇编语言属最底层的程序语言，若处理不当对控制系统的杀伤力最大，而且除错时间最长，并不适合开发大型的应用程序。本书中的例子均是以汇编语言写成的，在后续《8051 单片机彻底研究——实习篇》和《8051 单片机彻底研究——经验篇》中，我们将介绍如何用高级语言来编写 8051 的控制程序。

3-4 写汇编语言前：熟悉寄存器与指令

知道了 8051 汇编程序的正确格式后，还有一些信息是要做补充的，分别是熟悉寄存器的内容，以及大致理解 8051 指令的分类情形。你可以先浏览一下所有的寄存器及指令助记符的写法，然后才开始写程序。如果在写 8051 程序的过程当中，发现对某个指令不熟悉时，再随时补救一下，你就能放心去写不算大的 8051 汇编程序了。

8051 的寄存器整理如下：

（1）算数运算有关的寄存器：ACC(*)、B(*)、PSW 程序状态字(*)。

（2）指针类寄存器：SP 堆栈、DPH、DPL。

（3）并行输入/输出端口寄存器：P0(*)、P1(*)、P2(*)、P3(*)。

（4）中断控制寄存器：IP(*)、IE(*)。

（5）定时计数寄存器：TMOD、TCON(*)、TL0、TH0、TL1、TH1。

（6）串行通信寄存器：SBUF、SCON(*)。

有*号的寄存器代表该字节内每个位都可以使用位寻址的指令，例如我们可以用 CLR ACC.7，直接清除 ACC 的位 7 内容，但是 8051 没有类似 CLR SBUF.5 的指令，因为 SBUF 寄存器并未具备可单一位寻址的能力。

8051 的指令可区分为以下几类：

（1）跟算数运算有关的指令。

ADD,ADC	加法
SUBB	减法
INC,DEC	加 1 及减 1
MUL,DIV	乘法与除法
DAA	十进制调整

（2）跟逻辑运算有关的指令。

AND,ORI,XRL	逻辑上的 AND,OR 与异或
CLR,CPL	清除与反相
RL,RLC	左移
RR,RRC	右移
SWAP	累加器的位 7～位 4 与位 3～位 0 互换

（3）跟数据传送有关的指令。

MOV,MOVC	数据传送
MOVX	外部数据传送
PUSH,POP	堆栈数据传送
XCH,XCHD	寄存器数据互换

（4）跟布尔变量有关的指令。

CLR,SETB	清除或设置
CPL,ANL,ORL	逻辑上的反相,AND 与 OR
MOV	直接设置该位
JC,JNC	判断是否有 Carry 后做跳转
JB,JNB,JBC	判断该位状态后做跳转

（5）跟程序分支有关的指令。

ACALL,LCALL,RET,RETI	调用与返回
AJMP,LJMP,SJMP,JMP	跳转
JZ,JNZ,CJNZ,DJNZ	判断是否为 0 的跳转

还有一个跟程序没有关系但却很重要的指令：NOP。这个指令被用在进行时间延迟或补偿上。看完以上的说明，是否觉得 8051 的汇编语言不会太难，可以开始进行程序的编写了。

3-5 试写一个 8051 汇编程序

写 8051 的汇编程序一定要有硬件的配合，这样在操作验证上才有所依据。在这里我们以 AT2051 控制板为例，开始写一个非常简单的 8051 程序，文件名为 P2-4.ASM，程序让控制板上的 LED（P3.7）进行闪烁的操作，说起来很简单，先看看这段汇编语言是怎样完成的。

```
;PROGRAMNAMEP2-4.ASM
;WRITTERNBYCHIPWARESYSTEMSINC
;先把声明放在汇编程序的最前面
LED     REG     P3.7          ;P3.7=0 时 LED 亮,反之则熄灭
;
        ORG     0000H         ;RESET 之后程序由此点进入
        MOV     P3,#FFH       ;把 P3 全设成 1，连带着使 LED OFF
        MOV     P1,#FFH       ;把 P1 全设成 1
        MOV     R0,#00H       ;R0=00H
$       DJNZ    R0,$          ;让系统延迟一下才开始工作
        MOV     SP,#40H       ;SP 堆栈设成 40H
START   CLR     LED           ;P3.7=0,LED 亮
        CALL    DELAY         ;延迟一小段时间
        SETB    LED           ;P3.7=1,LED 熄灭
        CALL    DELAY         ;延迟一小段时间
        LJMP    START         ;重新再做一次
;
DELAY   MOV     R0,#00H       ;R0=00H,外循环的延迟次数
DLY     MOV     R1,#00H       ;R1=00H,内循环的延迟次数
$       DJNZ    R1,$          ;R1=R1-1,共执行了 256 次
        DJNZ    R0,DLY        ;R0=R0-1,外循环也执行了 256 次
        RET                   ;这个子程序约延迟半秒钟(正确的时间等待)
;
        END                   ;程序结束,这个伪指令可有可无
```

我们试写的第 1 个范例程序，实际上包含了以下几个设计要点：

（1）声明要放在最前面。

（2）程序最前面一定要加一小段时间延迟，让系统所有电源都稳定后才开始工作。

（3）堆栈的值一定要设置。

（4）LED 亮时一定要持续一段时间，人的肉眼才能察觉。

（5）开机时要设置某些 I/O 的起始值。

接下来是我们操作 8051 的汇编程序，进行编译（X8051）与链接（LINK）时所看到的画面，经过链接最后产生 P2-4.TSK 二进制文件，只要将此文件烧录到 AT89C2051 内，再将该 IC 放进控制板内，即可验证我们写的第 1 个程序是否真的有 LED 亮灭的功能。经过 X8051 编译时，我们可以指定输出 LIST 文件，看看与原来的 P2-4.ASM 有何不同。

```
2500 A.D.8051 Macro Assembler   -   Version 4.05b
-----------------------------------------------

                Input   Filename:P2-4.asm
                Output  Filename:P2-4.obj

1   ;PROGRAM NAME P2-4.ASM
2   ;WRITERN BY CHIPWARE SYSTEMS INC
3   ;先把声明放在汇编语言程序的最前面
```

```
 4          00B7 LED   REG   P3.7              ;P3.7=0 时 LED 亮,反之则熄灭
 5  ;
 6   0000                     ORG     0000H     ;RESET 之后程序由此点进入
 7   0000   75 B0 FF          MOV     P3,#FFH   ;把 P3 全设成 1,连带着使 LED 熄灭
 8   0003   75 90 FF          MOV     P1,#FFH   ;把 P1 全设成 1
 9   0006   78 00             MOV     R0,#00H   ;R0=00H
10   0008   D8 FE    $        DJNZ    R0,$      ;让系统延迟一下才开始运行
11   000A   75 81 40          MOV     SP,#40H   ;SP 堆栈设成 40H
12   000D   C2 B7    START    CLR     LED       ;P3.7=0,LED ON
13   000F   12 00 1A          CALL    DELAY     ;延迟一小段时间
14   0012   D2 B7             SETB    LED       ;P3.7=1,LED OFF
15   0014   12 00 1A          CALL    DELAY     ;延迟一小段时间
16   0017   02 00 0D          LJMP    START     ;重新再做一次
17  ;
18   001A   78 00    DELAY    MOV     R0,#00H   ;R0=00H,外循环的 DELAY COUNT
19   001C   79 00    DLY      MOV     R1,#00H   ;R1=00H,内循环的 DELAY COUNT
20   001E   D9 FE    $        DJNZ    R1,$      ;R1=R1-1,共执行了 256 次
21   0020   D8 FA             DJNZ    R0,DLY    ;R0=R0-1,外循环也执行了 256 次
22   0022   22               RET     ;这个子程序约延迟半秒钟(正确的时间待查)
23  ;
24   0023                     END               ;程序结束,这个伪指令可有可无
     Lines  Assembled: 24              Assembly Errors: 0
```

3-6 配合示波器做汇编语言的除错

写 8051 汇编语言绝对不是只坐在电脑桌前键入程序而已，写程序会出错，写汇编程序更会出错，有时候还会错得很离谱。当 8051 的程序出现非我们预期的结果时，你会如何处置呢？程序看来看去都是对的，可是运行就是不对，怎么办？先不要怀疑硬件，依照我们的统计与经验，当系统不工作时，有 80%以上的几率是程序有漏洞，10%是整合时的程序有问题，最后的 10%才是硬件的问题，不过硬件问题中的一半可以用软件来克服，所以系统有问题时，总体来讲有 95%是软件的问题。如果你写的是汇编语言的话，千万要记住这种说法。

我们所写的第 1 个汇编程序中，你可否发现一个疑问：子程序延迟执行一次实际上花了多少时间？有两个方法可用来计算延迟的时间，方法一是对照本书附录的指令表，查出每个指令所需的工作周期乘以次数（256 次），然后再乘上系统每个工作周期所花的时间。方法二是直接用示波器观看输出的波形，就可以从时间轴上看出。用示波器看波形的方法其时间的误差应该在 1%以内，所以除非是非常精准的延迟控制，否则用方法二就够了。

```
DELAY    MOV     R0,#00H     ;12 clock
DLY      MOV     R1,#00H     ;12 clock
$        DJNZ    R1,$        ;24 clock
         DJNZ    R0,DLY      ;24 clock
         RET                 ;12 clock
```

假设系统使用的石英振荡频率为 11.0592MHz，每个 clock 所要的时间为 0.09μs
内循环要执行 256 次，外加第一次设置 R1，共用了 256×24+12=6156 clocks
外循环也是执行 256 次，外加第一次设置 R0，共用了 256×6156+12=1575948 clocks
调用延迟程序一次所用的时间=1775948+12=1575960 clock=0.142s

图 3-1 查 8051 指令表来计算所需时间

配合示波器的波形查看，我们把上一个程序稍微修改，变成第 2 个 8051 汇编程序 P2-5.ASM，P3.2 位代表延迟的时间。

```
;PROGRAMNAMEP2-5.ASM
;WRITTERNBYCHIPWARESYSTEMSINC
;先把声明放在汇编程序的最前面
SCOPE   REG     P3.3        ;P3.3=1 时表示 DELAY 开始
LED     REG     P3.7        ;P3.7=0 时 LED 亮，反之则熄灭
;
        ORG     0000H       ;RESET 之后程序由此点进入
        MOV     P3,#FFH     ;把 P3 全设成 1，连带着使 LED 熄灭
        MOV     P1,#FFH     ;把 P1 全设成 1
        MOV     R0,#00H     ;R0=00H
$       DJNZ    R0,$        ;让系统延迟一下才开始运行
        MOV     SP,#40H     ;SP 堆栈设成 40H
START   CLR     LED         ;P3.7=0,LED ON
        CALL    DELAY       ;延迟一小段时间
        SETB    LED         ;P3.7=1,LED OFF
        CALL    DELAY       ;延迟一小段时间
        LJMP    START       ;重新再做一次
;
DELAY   SETB    SCOPE       ;P3.3=1
        MOV     R0,#00H     ;R0=00H,外循环的延迟次数
DLY     MOV     R1,#00H     ;R1=00H,内循环的延迟次数
$       DJNZ    R1,$        ;R1=R1-1,共执行了 256 次
        DJNZ    R0,DLY      ;R0=R0-1,外循环也执行了 256 次
        CLR     SCOPE       ;P3.3=0
        RET                 ;
```

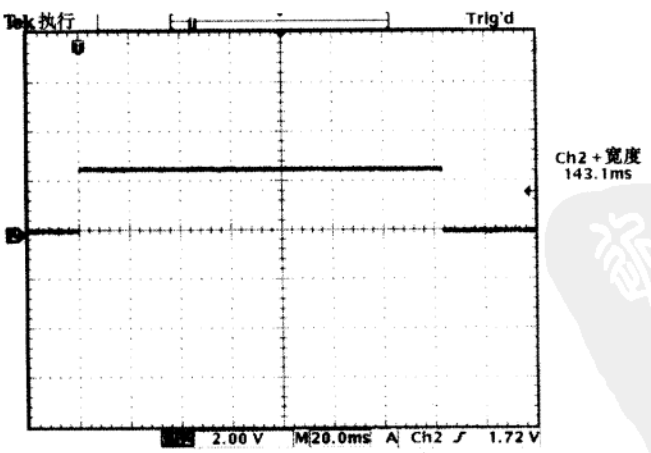


图 3-2　示波器 P3.7 波形图

注：测得的时间约是 144 ms，所以 LED 的闪烁速率约是 3.5Hz

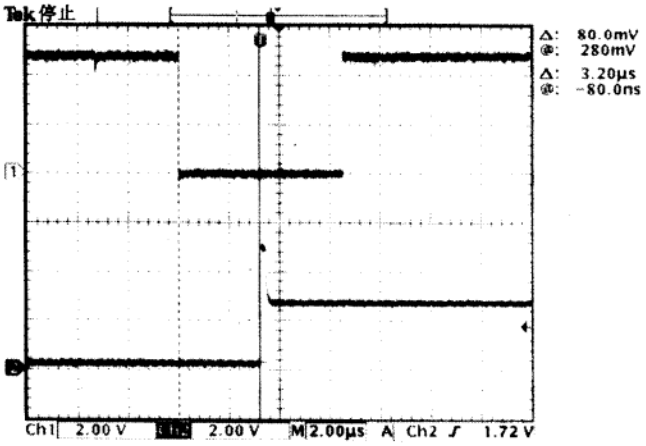


图 3-3　P3.3 与 P3.7 间的时间差约是 3.2µs，其中 CH1=P3.3 SCOPE，CH2=P3.7 LED

上图显示 SCOPE 信号降至 0 后，又过了 3.20µs 时间后，LED 的状态值才变成 0，这段时间要如何解释呢？

```
CLR     SCOPE    ;t=0
RET              ;2 Machine cycles
CLR     LED      ;1 Machine cycles
```

总共经过了 3 个机器周期，才使 LED 状态升至 1。AT2051 使用 11.0592MHz 的石英晶体，1 个机器周期=12clocks≈1.085µs，3 个机器周期就等于 3.255µs，与示波器测量到的 3.20µs 相去不远。

当程序运行用示波器验证无误后，就可以把有关于 SCOPE 的几行指令用“;”标记起来，让程序在执行时能够更节省时间。我们可以在这里说，写汇编语言不只是写写程序而已，你还要学会使用示波器来帮忙除错，另外要留意系统使用的频率及随时把 8051 指令表放在身边，不这样做的话，写超过 100 行以上的程序时，保证让你汗流浃背。

3-7　更进一步的 8051 汇编程序

如果我们又重新浏览一次 8051 所有的指令，其实上一个程序 P2-5.ASM 可以更精简一些。8051 本身就有布尔代数的处理能力，每个位地址的位置都可以做 AND、OR 或反相的操作，所以我们可以把 LED 点亮或熄灭的指令，改为 CPL 直接反相，这样会使程序看起来更为简洁。

```
;PROGRAMNAMEP3-7.ASM
;WRITTERNBYCHIPWARESYSTEMSINC
;先把声明放在汇编程序的最前面
LED     REG     P3.7          ;P3.7=0 时 LED 亮,反之则熄灭
;
        ORG     0000H         ;RESET 之后程序由此点进入
        MOV     P3,#FFH       ;把 P3 全设成 1,连带着使 LED OFF
```

```
        MOV     P1,#FFH     ;把 P1 全设成 1
        MOV     R0,#00H     ;R0=00H
$       DJNZ    R0,$        ;让系统延迟一下才开始运行
        MOV     SP,#40H     ;SP 堆栈设成 40H
START   CPL     LED         ;P3.7 反相一次
        CALL    DELAY       ;延迟一小段时间
        LJMP    START       ;重新再做一次
;
DELAY   MOV     R0,#00H     ;R0=00H,外循环的延迟次数
DLY     MOV     R1,#00H     ;R1=00H,内循环的延迟次数
$       DJNZ    R1,$        ;R1=R1-1,共执行了 256 次
        DJNZ    R0,DLY      ;R0=R0-1,外循环也执行了 256 次
        RET                 ;
```

这种改善的做法好像可有可无似的，不过碰到较复杂的 8051 控制程序，程序的空间又不太够，有时候就只差几个字节时，这种写法就真能够帮上忙，不过，要做这种修改的先决条件是你必须要对程序有全盘的理解，否则会越改越糟的。

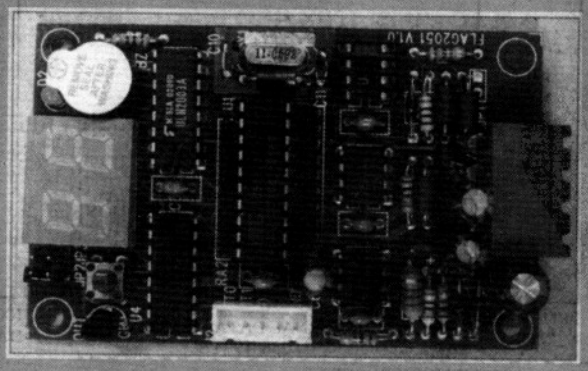
3-8 8051 的反汇编程序

8051 单片机的程序设计工具除了有汇编程序（Assembler）外，另外还有反汇编程序（Disassembler），前者是将我们所写的汇编语言转换成 8051 单片机的机器码，这些机器码只要烧录到 EPROM 或 Flash 就能让单片机执行，后者则是把机器码还原回为原来的汇编语言格式，我们可通过此工具学习别人程序的写法，经过吸收和整理使之变成我们最宝贵的参考文件及资料，因为这些设计上的经验绝对无法在教科书中找到。本书的附书光盘中有一个相当实用的 8051 反汇编程序 DIS51，它除了将机器码还原外，还能对相关重要的调用地址和表（Table）进行整理，详细的操作情形请参考《8051 单片机彻底研究——经验篇》或《8051 单片机彻底研究——实习篇》书后附录的说明。

习　题

1. 学习 8051 最基本的程序工具有哪些？
2. 什么是助记符？它和机器码有何差别？
3. 如何理解 Design for Test？
4. 典型 8051 汇编语言的架构中，包含哪些要点？
5. 运算器的功能是什么？
6. 与数据传送相关的指令有哪些？
7. 什么是 Disassembler？
8. 当 8051 程序出现非预期的结果时，较有效的解决方法有哪些？

4



AT2051 控制板上有温度感测器与数字显示器，所以能把测到的温度值显示出来

第 4 章　8051 的存储器

8051 的存储器空间不论是程序存储器还是数据存储器都可以到达 64KB，这是与传统 CPU 衔接的考量。随着半导体技术的发展，把 64KB 的程序空间直接固化到 CPU 芯片中也已经不是问题了，困难的是程序如果写这么大,如果没有完整的除错模式,你猜结果如何呢?

4-1　8051 内部存储器的分配

8051 的存储器组织分成 3 种地址空间和一个程序计数器 PC，如图 4-1 所示，其中包括：

（1）一个 16 位的程序计数器：赋予 8051 最大有 64KB 的寻址能力。

（2）64KB 的程序存储器空间：这代表我们所写的程序码可达 65536 个字节，传统 8 位 CPU 如 Z80 或 6502 的程序空间也是 64KB。

（3）64KB 的外部数据存储器空间：这部分空间是同时可以读和写的，8051 单片机特地将数据与程序分开处理，这使得数据存储器有绝对足够的空间存放系统的变量和数据，如果想另外做其他 I/O 设备 IC 的扩展时，还可以分出部分的数据存储器空间给这些设备使用，完全不干扰到程序存储器的空间，凭借这个特点就使得许多设计工程师没有采用 Z80 或 6502，而改采用 8051 系列的单片机来完成他们的控制系统。

（4）256 字节的内部存储器空间，这里面包括 SFR 特殊功能寄存器、堆栈区、数据区和通用寄存器组 R0～R7，如果系统不是很复杂时，256 字节的内部存储器空间大约是足够的。8052 系列的单片机则有 384 个字节的内部存储器空间。

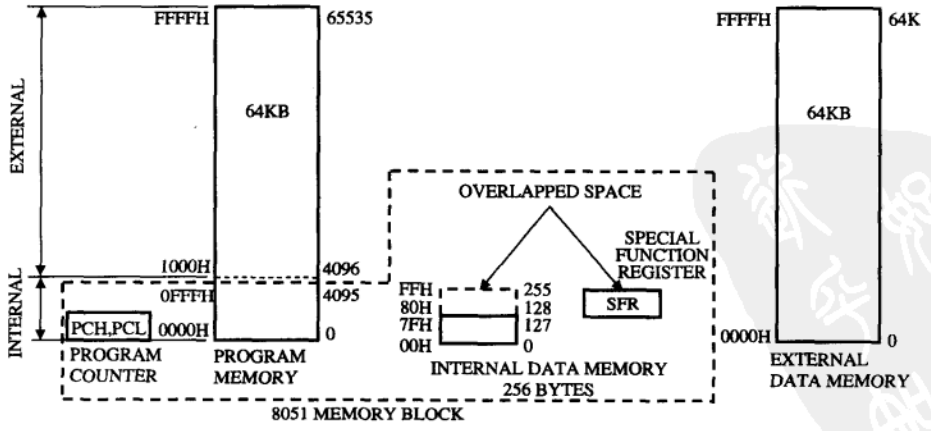


图 4-1　8051 存储器示意图

注：虚线外则是可扩展的部分，虚线内是 8051 芯片内的存储器，不论是程序空间或数据空间都可以加到 64KB。

4-2 程序存储器空间

在 8051 和 8751 的单片机中，64KB 的程序存储器空间最前的 4KB 是存在内部的 ROM 和 EPROM 上，如果把引脚 EA 拉成高电位时，可以迫使微处理器在系统重置后直接读取这 4KB 的程序内容，而当系统程序超过 4KB 时，8051 会自动将程序计数值 PC 送到 P0 和 P2 端口上，以便读取外部扩展的程序存储器。如果 EA 脚开机时就是低电位时，代表系统只使用外部程序存储器空间，Intel 特别强调说：不论使用内部或外部程序存储器，其执行速度是完全相同的。

假如我们的程序长度不到 4096 个字节，理论上这些程序可以存放在 8051 的内部 ROM 或 8751 的内部 EPROM 中，同时还可以妥善运用 P0 和 P2 的双向 I/O 功能，不过要特别留意的是，应保留程序空间最后的几个字节不用，以免程序无故跳到这几个地址，导致 8051 在指令码的预先捕获操作时呈送出状态。

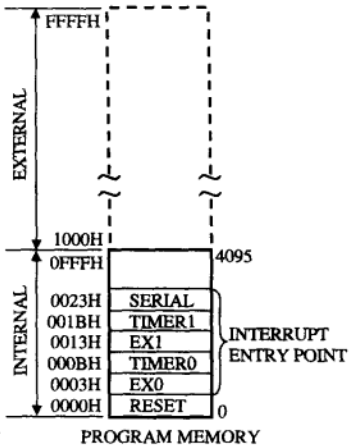


图 4-2 8051 程序存储器的配置图

注：我们的程序放置位置应该从 002EH 开始才会比较安全，这样就能避免 8051 中断的进入点。

在程序存储器的最开头，有许多地址是保留给特殊程序使用的，这些地址分别是：

（1）地址 0000H：程序起始进入点，通常我们在这个地址上安插一个 LJMP ENTRY 指令（占 3 个字节），直接跳到程序真正的进入点。

（2）地址 0003H：外部中断 0（INT0）的进入点。

（3）地址 000BH：定时/计数器 0（Timer0）溢位中断的进入点。

（4）地址 0013H：外部中断 1（INT1）的进入点。

（5）地址 001BH：定时/计数器 1（Timer1）溢位中断的进入点。

（6）地址 0023H：串行通信中断的进入点。

（7）地址 002BH：定时/计数器 2（Timer2）溢位中断的进入点，此功能只有 8052 系列单片机才有。

中断进入点上依然是以 LJMP ISR 的指令，将程序跳到各个中断服务程序的真正进入点上。由上述的说明中我们得知，实际的应用控制程序应是由地址 002EH 以上开始的。

4-3 外部数据存储器空间

8051 的外部数据存储器的存取只能通过 MOVX 指令来执行，即使系统扩充的外部存储器不大时，可以使用 MOVX A,@Ri 和 MOVX @Ri,A 指令，此时 Ri 寄存器上是存放 8 位的外部存储器的地址值，Intel 当初研发此指令时，静态 SRAM 的价格并不便宜，而且容量也不大（顶多只达 1KB 而已），今天我们买得到的 SRAM 6116 其容量是 2KB，所以通过 Ri 间接地址的指令可能不太合时宜了，这时只剩下 MOVX @DPTR,A 和 MOVX A,

@DPTR 指令可用，上述两个指令都通过 16 位的 DPTR 指针对外部的数据存储器进行存取的操作，DPTR 上必须事先设置外部数据存储器的地址值，所以对外部数据存储器读写的格式就成为：

```
MOV   DPTR,#SRAM
                ;设置外部数据存储器地址
MOVX  A,@DPTR
                ;读取一个外部数据存储器值
                ;此时 DPTR 值会出现在地址线上
                ;而且伴随送出 RD 信号

MOV   DPTR,#SRAM
                ;设置外部数据存储器地址
MOVX@DPTR,A  ;将 ACC 值写入外部数据存储器中
                ;此时 DPTR 值和 ACC 值会分别出现
                ;在地址线和数据线上，并配合
                ;WR 信号，以便将数据写入 SRAM 中
```

如果系统有其他设备 IC 做扩充时，也都是通过上述两个指令来做存取，通常系统硬件线路中要加入地址译码电路，区分出哪些地址是给 SRAM 使用，哪些地址是给周边 IC 使用的，当然此时系统的外部数据存储器就不能到达 64KB 了。

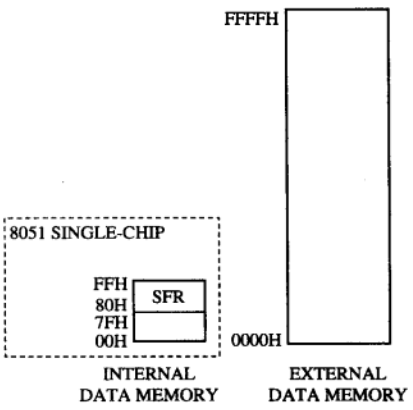
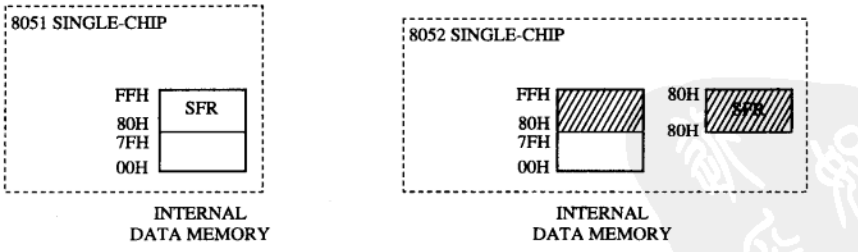


图 4-3　8051 外部数据存储器

注：不论是 8051 或 8052，其外部的数据存储器空间都可到达 64KB，但是如果还有其他的外部 I/O 时，也会占用部分地址。在硬件方面一旦使用外部存储器时，8051 的 P0 与 P2 就不能作为一般的 I/O 了。

4-4　内部数据存储器空间

8051 的内部数据存储器空间有 128 字节（00H～7FH），其 SFR 寄存器则占用 80H～FFH 的部分地址，8052 单片机则有多一倍的内部数据空间，8051 和 8052 单片机内部数据存储器的差异见图 4-4 所示，8052 的内部数据存储器 80H～FFH 必须使用间接寻址的方式才能做存取的操作。虽然这段地址区和 SFR 地址重叠，但是其指令码并不相同，所以 CPU 在做存取时并不会有混淆。



（a）8051 共有 256 字节的数据存储器　（b）8052 共有 384 字节的数据存储器，但是其中有 128 字节重叠

图 4-4　内部数据存储器空间的差异

图 4-5 是 8051 内部数据存储器的配置图。图中我们也可以看到低地址 00H～1FH 被区分成 4 个 8 组常用寄存器组（Register Bank），在程序运行的期间内，一次只能有一组寄存器组被选择，当我们进行 R7～R0 常用寄存器的存取时，CPU 会自动取到对应的寄存器组。至于

是哪个寄存器组调用，就要由 PSW 寄存器上的 RS1 和 RS0 来决定了，单片机 RESET 后 RS1 和 RS0 都被清除为 0，所以系统首先使用 Bank0 上的 R7～R0 寄存器。

```
RS1=0,RS0=0  选用 Bank0
RS1=0,RS0=1  选用 Bank1
RS1=1,RS0=0  选用 Bank2
RS1=1,RS0=1  选用 Bank3
```

在 4 个 Bank 之上有 16 个字节（20H～2FH）可以进行位寻址，总共有 128 个位是位地址的，可以利用 8051 的布尔指令，对其中的单一位做设置、清除、判断及逻辑 AND 或 OR 的处理，使得系统可以很有效率地使用内部数据存储器的空间。除了这段区域外，其他的地址是没办法进行位寻址的，SFR 寄存器也有部分寄存器是可以进行位寻址。

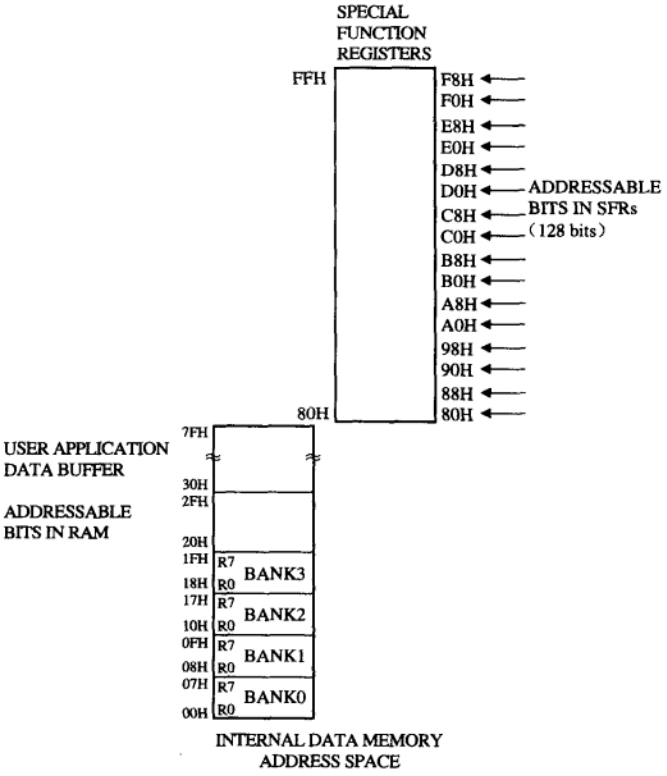


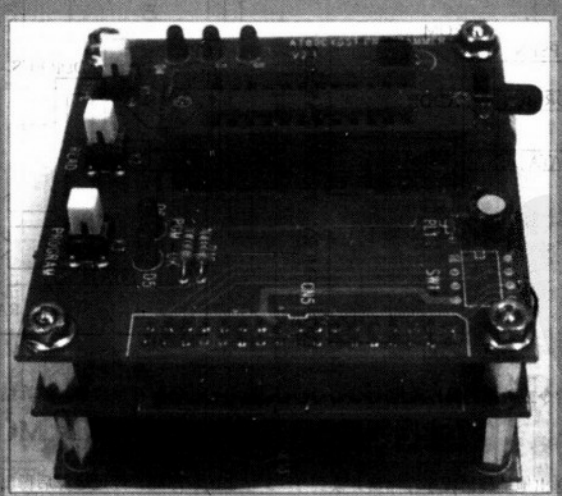
图 4-5　8051 内部数据存储器的配置图

注：右上方是 SFR 占用位置

习　题

1. 8051 的内部存储器如何分配？
2. 8051 的程序存储器空间最大容量为多少？
3. 8051 的程序存储器有多少字节是存在其内部的 ROM 或 EPROM 上？
4. 当系统程序超过内部存储器的空间时，8051 会通过哪些端口与外部存储器进行通信？
5. 程序存储器的开头有哪些地址是保留给特殊程序使用的？请列举 3 种并说明其功能。
6. PSW 寄存器上的 PS1 及 PS2 的作用是什么？
7. 在 8051 寄存器组之上，一共有多少字节可进行位寻址？其地址是什么？
8. 8052 的数据存储器中有多少字节是重叠的？

5



USB 烧录器烧录一颗 AT89C2051 的时间可在 10s 内完成，这个速度与 ICE 下载的速度相当

第 5 章　8051 指令的寻址模式

8051 单片机的几种寻址模式都是单片机初学者必须了解并熟练掌握的。有些指令有相同的结果，但是指令长度与执行时间却不同，原因就在于寻址模式的不同。当我们深刻理解其中的差异后，我们写出来的程序就会更简洁，执行速度也会更快。

5-1　8051 执行指令的过程

当我们利用程序来操作 CPU 执行程序时，其中的操作流程大概可细分成 5 个步骤：

（1）获取指令，这是指 CPU 通过地址线、数据线和控制线向程序 ROM 读取指令和待处理的数据值。

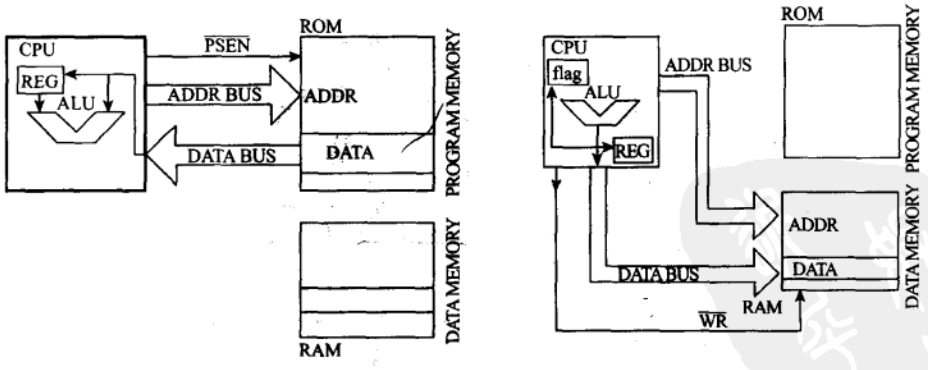
（2）CPU 内部数据处理，将步骤 1 取到的数据和某个寄存器做运算。

（3）运算后结果再做逻辑上或标志位（flag）上的处理。

（4）依照判断的结果进行跳转，条件为真时如何操作，条件为假时如何操作。

（5）将结果保存到其他寄存器或存储器中。

说得更简洁一点，上述的步骤其实就是采集数据、处理数据，然后存储数据。而 CPU 到哪个地址获取数据及将数据存往何处，也就是我们所说的寻址模式（Addressing Mode），前面所讲到的 MOV 数据传送指令，正是寻址模式的最好例子。



（a）CPU 获取指令或数据的操作流程　　（b）存放数据（运算结果）的操作流程

图 5-1　8051 执行指令的过程图

注：CPU 读取指令时，先送出地址线给程序存储器并读到 1～3 字节的程序码，经过运算后得出的结果，有可能暂时存在缓冲区或转存到数据存储器内

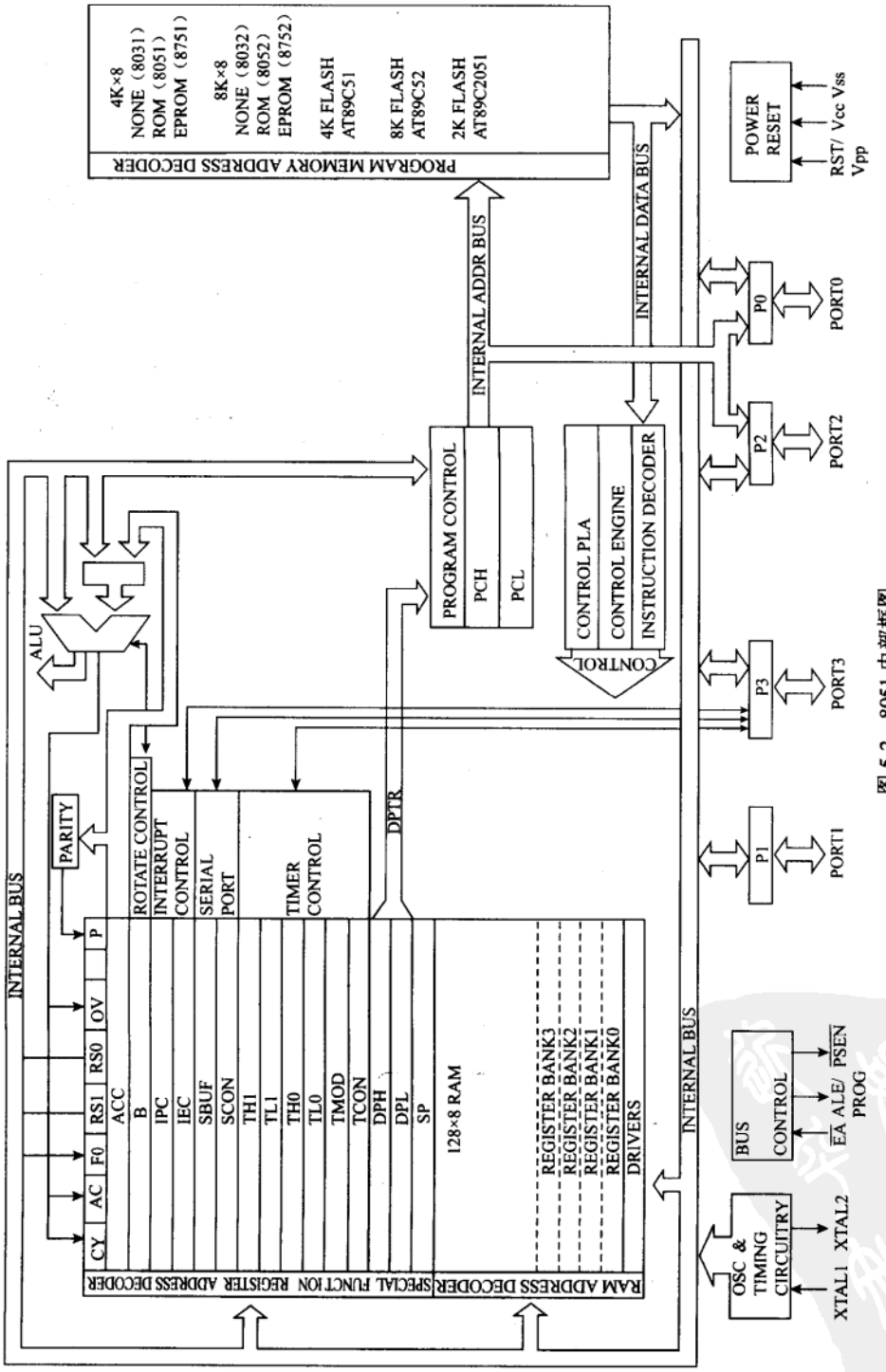


图 5-2　8051 内部框图

注：图中下方的指令译码器（Instruction Decoder）及控制器（Control Engine）会决定每个指令的寻址模式。

再来看看 8051 单片机内部的框图，其内部除了最重要的累加器 ACC 外，还有其他常用的寄存器（这些归入 SFR 特殊功能寄存器）、4 个输入/输出端口、4K 字节的程序区以及 128 字节的数据存放区。数据存放区内有一段（20H~2FH 共 128 位）是可单一位寻址的。而我们所写的程序就是控制 8051 内部数据的传送罢了！弄懂 8051 的寻址模式可以让我们对 CPU 的运行模式有更进一步的理解，请看我们的实际范例。

假设我们的程序里面有一行指令是 MOV A，30H，这是把地址 30H 的内容传递给 ACC 累加器的指令，你可以通过本书附录的 8051 指令表，找出其对应的十六进制机器码分别为 E5H 与 30H，E5H 这个程序码对 8051CPU 而言，正是一个把某个位置内的值转给 ACC 的指令，只要 CPU 读到 E5H 这个值时，就知道这是一个双字节的指令，它会自动往下获取另一字节的值（此例即 30H），在得到 30H 这个值之后，CPU 接着到数据存储器的 30H 地址上取出其内容，再把这个内容传进 ACC 上。

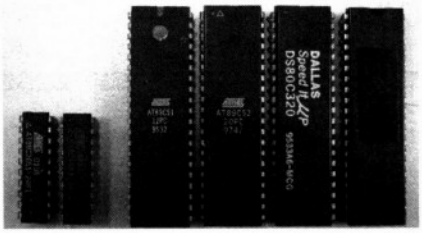


图 5-3　几款常见的 8051 单片机 IC

注：由左到右分别是 AT89C2051、AT89C4051、AT89C51、AT89C52、80C320 以及 8051，20 支引脚内置 Flash 的 AT89C2051 和 AT89C4051 本身无法扩展，所以无缘使用 MOVX 的间接寻址指令。内部无程序存储空间的 80C320 也不能用 MOVC 指令去读取内部的程序码或值。

假设 CPU 现在的程序计数值是 02F0H，内部的数据存储器 30H 的内容为 88H 时，若接下来的汇编语言为 MOV A，30H，而执行完 MOV A，30H 指令后，累加器 ACC 的内容变成 88H。

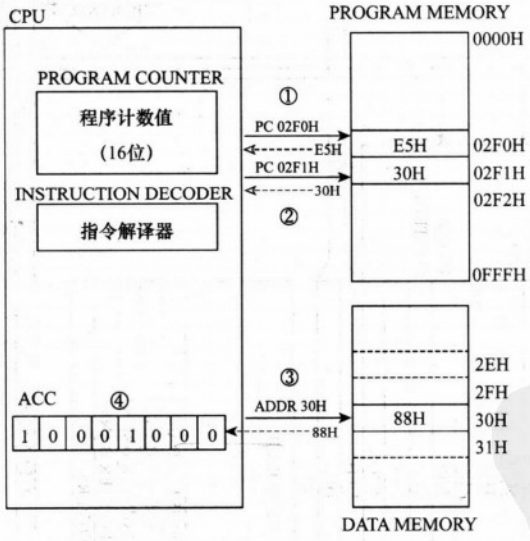


图 5-4　8051 内部局部执行图

注：8051 内部执行描述如下：
步骤 1：送出 PC 值=02F0H，读回第一个程序码 E5H。
步骤 2：送出 PC 值=02F1H，读回第二个程序码 30H。
步骤 3：CPU 针对内部 DATA MEMORY 送出地址 30H，读到 88H 的数据值。
步骤 4：ACC 值的内容被填成 88H。
步骤 5：送出 PC 值=02F2H，读取下个指令的机器码。

这些数据转来转去的操作都在一个机器周期内（约 1μs）完成。在这整个过程中，最为重

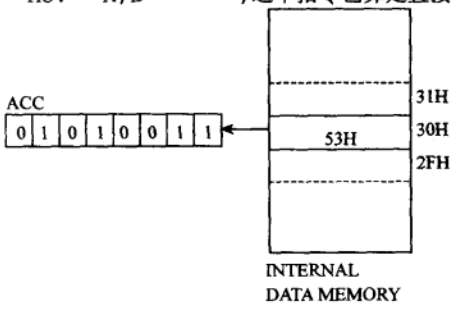
要的是 CPU 取得的 E5H 这个机器码，这个码看似简单，实际上，它已代表了该指令的寻址模式与指令长度，而这也是学习 8051 单片机指令一定要学寻址模式的原因了。

8051 的寻址模式可分成 5 类，分别是直接寻址、间接寻址、寄存器寻址、立即寻址与索引寻址等，接下来的章节就要对这些模式做进一步的说明。

5-2 8051 的直接寻址模式

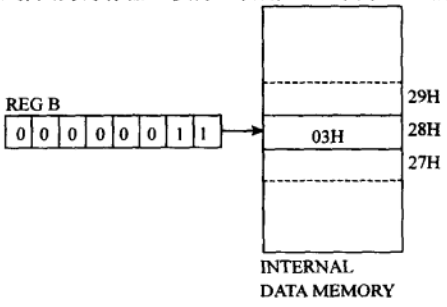
直接寻址模式（Direct Addressing）：8051 直接将某个地址内的值传给 CPU 中的某个寄存器，这种写法最为常见，如：

```
MOV   A,30H      ;把 30H 地址内的数据传给累加器 ACC
MOV   28H,B      ;把寄存器 B 的值存入 28H 的地址内
ADD   A,60H      ;将 ACC 和 60H 地址内的数据相加，结果存在 ACC 中，加法会影响 PSW 上的值
MOV   A,B        ;这个指令也算是直接寻址模式，因为寄存器 B 实际上占用 SFR 中为 F0H 的地址
```



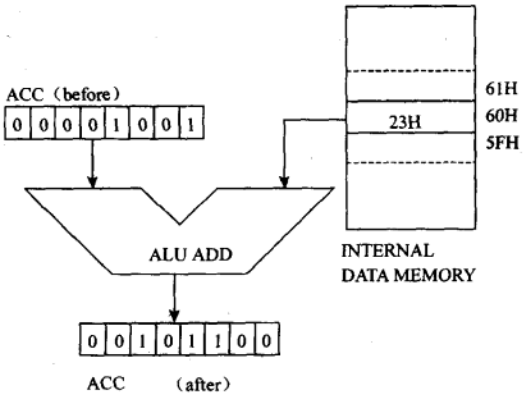
（a）MOV A,30H

注：若内部数据区 30H 的内容是 53H 时，执行完后，ACC 也变成 53H。



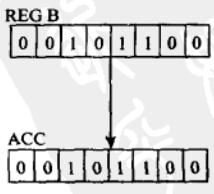
（b）MOV 28H，B

注：把 B 转存到数据存储器 28H 上，所以 28H的内容成为 03H。



（c）ADD A,60H

注：60H 的内容与累加器 ACC 做加法运算，结果存回ACC 上，这个指令会影响 CY 进位标志位。



（d）MOV A,B

注：寄存器 B 的值传递给累加器 ACC。

图 5-5 常见的 4 种 8051 直接寻址模式

5-3 8051 的间接寻址模式

间接寻址模式（Indirect Addressing）：8051 利用寄存器（R0、R1、SP 和 DPTR）当成指针，间接取得该指针内的数据，采用这种寻址模式的指令中，会有一个@的符号，这种模式可以取到所有寄存器及程序存储器或数据存储器中任一个地址的值。间接寻址的例子如下：

```
MOV   R0,#50H        ;寄存器 R0=50H，把 R0 指向 50H
MOV   A,@R0          ;利用间接寻址法，取得 R0 所指的 Data
                     ;Memory 数据存储器中地址为 50H 的内容并存入 ACC 中
MOV   DPTR,#8000H    ;寄存器 DPTR(16 位)=8000H
MOVX  A,@DPTR        ;利用间接寻址法，取得外部(EXternal)数
                     ;据存储器地址为 8000H 的内容，并将该值存入 ACC 中
```

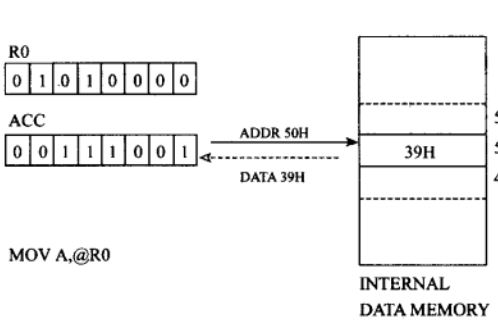


图 5-6 MOV A, @R0

注：R0 事先被设为 50H，当执行 MOV A,@R0 时，CPU 会送出地址线 50H 给内部的数据存储器，并读回一个 39H 的值，这个值稍后又传给累加器 ACC。

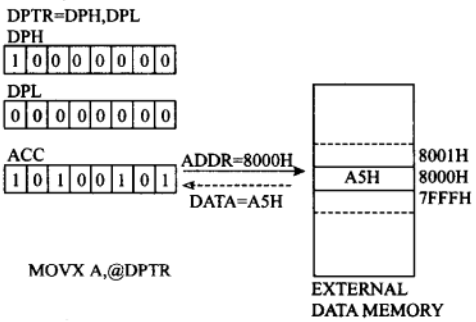


图 5-7 MOVX A, @DPTR

注：DPTR 的内容事先被设定成 8000H，即 DPH=80H，DPL=00H，当碰到 MOVX 指令时，知道接下来要做外部数据存储器的读取，所以接下来会通过 P0 与 P2 端口把完整的地址线送出，并且伴随外部读取的 RD 信号，由图中我们可以看到最后有 A5H 的数据被填入 ACC 累加器上。

```
MOV   DPTR,#6000H    ;DPTR=6000H,假设是 SRAM 外部数据区
MOV   A,OFFSET       ;ACC 等于 OFFSET 内的值
ADD   A,DPL          ;A=A+DPL
MOV   DPL,A          ;DPL=DPL+(OFFSET)
MOV   A,B
MOVX  @DPTR,A        ;利用间接寻址法,将寄存器 B 的值存入外部存储器 SRAM 内
```

当我们的程序要取得一串存在存储器的数据时，可以将指针定在该数据区的起始点，然后借着指针的改变，取得该区的所有数据。

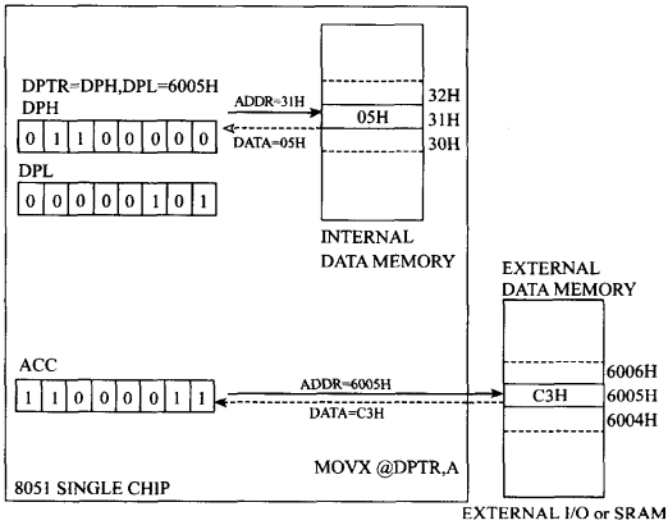


图 5-8　MOVX @DPTR,A

注：假设 OFFSET 的值是 31H，MOV A, OFFSET 后 ACC 的内容成为 05H，这个值又与 DPTR 相加，DPTR 变成 6005H，MOVX 指令执行后，会把 ACC 值送到外部数据存储器 6005H 的位置上。

5-4　8051 的寄存器寻址模式

寄存器寻址模式（Register Addressing）：8051 的 CPU 除了有许多功能各不相同的寄存器外，最常用的寄存器应该是累加器 ACC，而使用率第二高的就是 R7～R0 共 8 个寄存器，在 8051 中共有 4 组 R7～R0 寄存器，使用时靠 PSW 状态寄存器中的 RS1 和 RS0 来决定使用哪一组 R7～R0 寄存器。

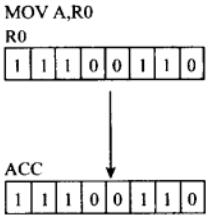
由于 CPU 处理时 R7～R0 的使用几率甚高，所以在 8051 的指令中有许多跟 R7～R0 有关的指令，我们可以通过 R0～R7 共 8 个寄存器做数据的传送或加减等运算，这些指令我们统统归类为寄存器寻址。属于这方面的例子有：

```
MOV   A,R0      ;将 R0 的值传给累加器 ACC
ADD   A,R5      ;将 ACC 值和 R5 值相加,并把结果存放在 ACC 中
MOV   R7,A      ;将 ACC 值放入 R7 寄存器
```

事实上，在 8051 系统中，R7～R0 这 8 个寄存器本身就存在数据存储器中，共占用了 32 字节的空间，所以说，8051 的寄存器寻址法包含于直接寻址法中，举例说：

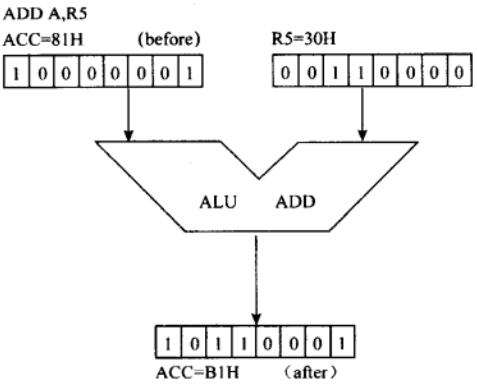
```
MOV   A,R6      ;将 R6 值存入 ACC 中
MOV   A,06H     ;将 06H 地址中的内容存入 ACC 中
```

以上两个指令的效果应当是一样的，但是第一种写法只占 1 字节的空间，不过会因 RS0 与 RS1 值的不同取得不一样的值。第二种属于直接寻址模式，该写法却要花两字节的空间，所以还是有点差异性存在。



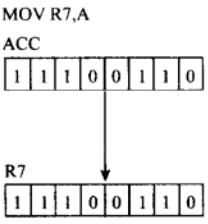
（a）MOV A,R0

注：R0 的内容直接转给 ACC 累加器，这种指令只有 1 字节，执行速度最快。



（b）ADD A,R5

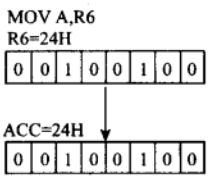
注：ACC 累加器与 R5 的内容相加，结果放在 ACC 上，这类的指令会占用 2 字节的程序空间。



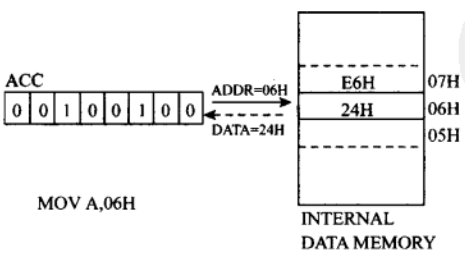
（c）MOV R7,A

注：把 ACC 累加器的值转存到 R7 寄存器上。

图 5-9　8051 寄存器寻址模式



（a）MOV A,R6 只占用 1 字节



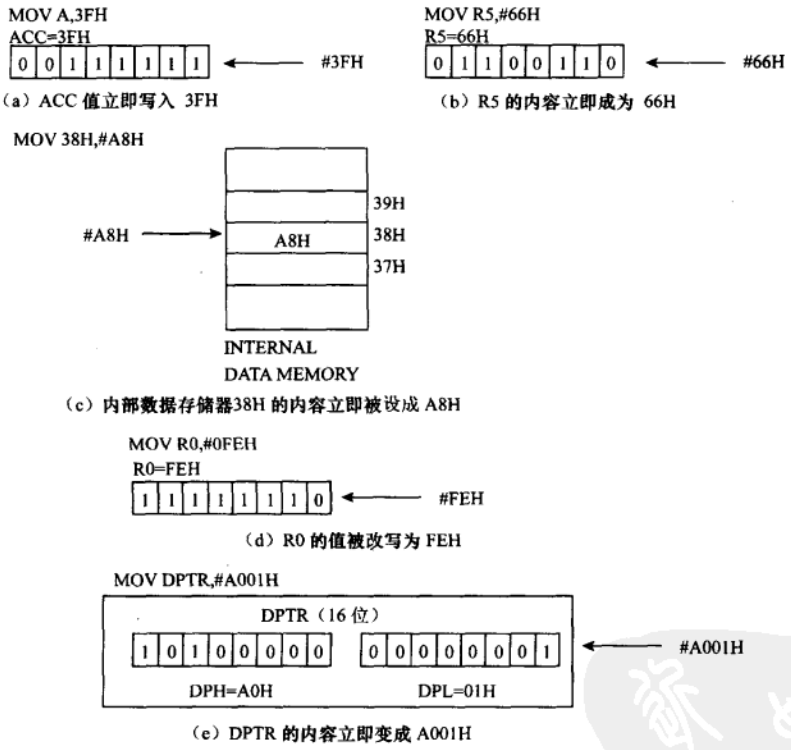
（b）MOV A,06H 占用 2 字节

图 5-10　MOV A, R6 与 MOV A, 06H 指令的比较

5-5 8051 的立即寻址模式

立即寻址模式（Immediate Addressing）：当我们想要让寄存器或某个地址中，立即存入某个值时，就可采用立即寻址法，这种写法是在该值前加上一个“#”符号，由于 8051 系统对 CPU 内部的数据存储器特别照顾，所以可以对这些地址用立即寻址法写入数据，立即寻址法的例子有：

```
MOV   A,#3FH          ;将 3FH 值存入 ACC 累加器中
MOV   R5,#66H         ;将 66H 值存到 R5 寄存器中
MOV   38H,#A8H        ;把 A8H 值存到地址为 38H 的数据存储器中
MOV   R0,#0FEH        ;R0=FEH，有些汇编语言编译程序对于十六进制值一定要求前面加 0
MOV   DPTR,#A001H     ;把 A001H(16 位)值存入 DPTR 寄存器中
```



（a）ACC 值立即写入 3FH

（b）R5 的内容立即成为 66H

（c）内部数据存储器38H 的内容立即被设成 A8H

（d）R0 的值被改写为 FEH

（e）DPTR 的内容立即变成 A001H

图 5-11 8051 的立即寻址模式

5-6 8051 的索引寻址模式

索引寻址模式（Index Addressing）：在 8051 的汇编语言程序中，我们有时候要做查表（TABLE）的操作，首先我们会利用程序计数值（Program Counter）或 DPTR 寄存器

当成基底值，然后将这个基底值加上 ACC 累加器的值，得到一个地址值，接着由此地址值取出其中的内容，完成一次查表的操作，这种寻址法称为索引寻址法，而 ACC 即是所谓的索引值。

在 8051 系统中，限定索引值仅能以 ACC 担任，所以，若以其他值当成索引值时，最后一定要将该值传到 ACC，然后才做索引寻址取到正确的值。这方面的例子仅有两个：

```
MOV    DPTR,#2340H     ;DPTR 等于 2340H 值
MOVC   A,@A+DPTR       ;把 A 值和 DPTR 相加得到一个地址，由此地址中取得一个值并存入 ACC 中
```

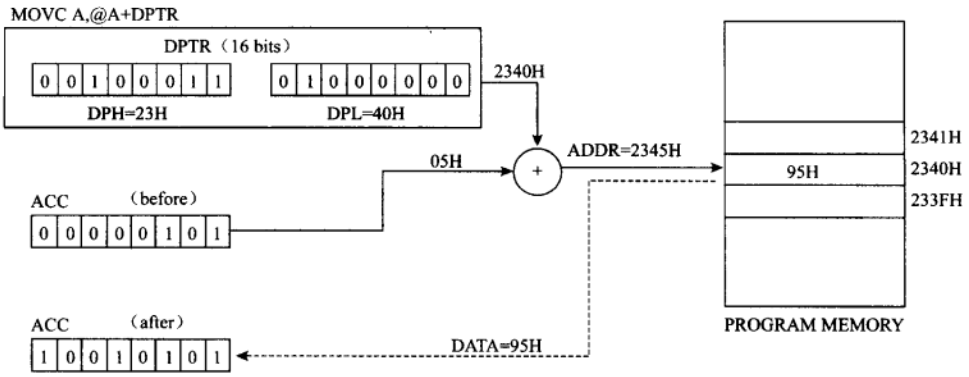


图 5-12　8051 的索引寻址模式 MOVC A,@A+DPTR

注：ACC 先前若是 05H，DPTR 为 2340H，两者相加后为 2345H，CPU 用这个值当成地址值去读取的程序存储器，得到 95H 的值，最后这个值又转给 ACC 累加器。

```
MOVC   A,@A+PC   ;程序现在的 PC 值加 A 值得到一个地址值，并由该地址中取得一个值并存入 ACC 中
```

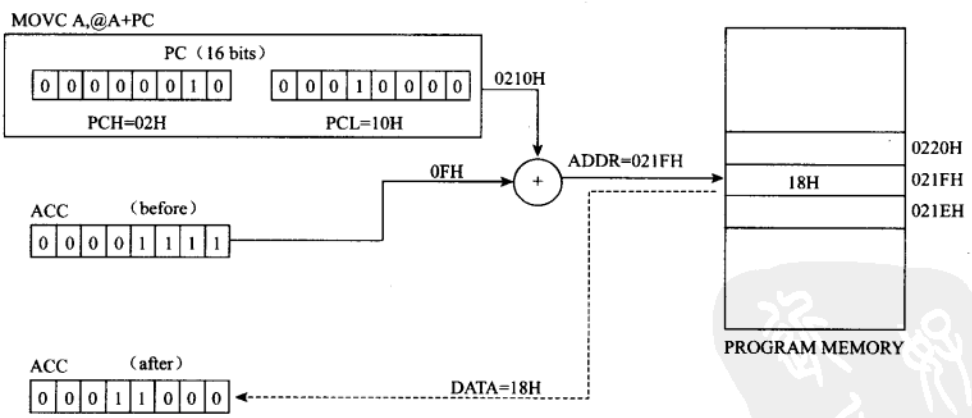


图 5-13　8051 的索引寻址模式 MOVC A,@A+PC

注：ACC 先前若是 0FH，程序计数值 PC 为 0210H，两者相加后为 021FH，CPU 用这个值当成地址值去读取的程序存储器，得到 18H 的值，最后 ACC 累加器取得该值，即 ACC 内容为 18H。

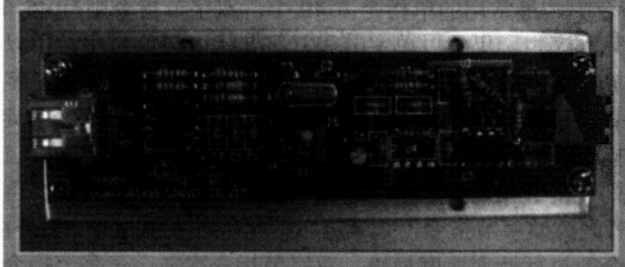
由于索引值 ACC 是 8 位的，因此查表的范围将限定在 0～255 间，在写查表的子程序时，我们应该留意此种寻址方法的特殊限制，以免造成查表操作的错误。使用这类指令时要特别

留意 DPTR 的值，例如你使用的是只有 2048 字节的 AT89C2051，则其 DPTR 值就不可以超过 0800H。

习　题

1. 8051 执行程序时，其操作流程可分为哪些步骤？
2. 8051 的寻址模式有哪几类？
3. 最常用的寻址模式有哪些？试说明其运行原理。
4. 哪种寻址模式可取得所有寄存器及程序存储器或数据存储器中任意一个地址的值？
5. 哪种寻址模式有时候要进行查表的操作？
6. 在寄存器寻址模式中，哪些寄存器是最常被使用的？

6



USB 转 RS485 的接口，它是即插即用的设备，对 Windows 而言，插入后系统增加了一个 COM 端口

第 6 章　8051 指令说明

大多数的人认为指令的介绍与认识是最无趣的，我们觉得事实也是如此。可是如果换另一个角度来看，学习指令只是踏入 8051 单片机的第一步而已，你不必急着弄懂所有的指令，只要先熟悉一些最常用的指令就可以写些简单的程序了。

8051 系列 CPU 中总共包括了 111 个指令，比起其他的 CPU，其指令数并不算太多，在实际的应用上，常用的指令并不会超过 50 个，所以若在学习阶段，可以先浏览 8051 整个指令集一两次，大略知道每个指令的用法，然后在实际写程序时再做指令的加强练习，只要经过三四次的磨炼，应该能够深切掌握 8051 指令的运用。

6-1　8051 指令格式

8051 的指令格式包括一个功能助记符（Function Mnemonic）和操作数（Operand），功能助记符告诉 CPU 要做哪个操作，而操作数则另外提供足够的信息给 CPU，以便完成整个指令的操作，在 8051 的指令集中有些指令不需要操作数，有些指令则需要 1~3 个操作数，CPU 会根据功能助记符的种类来决定取几个操作数。所以我们所写的程序，经过编译后的机器码一定要正确，否则 CPU 执行后，一定不会正常执行，甚至死机。除此之外，利用汇编语言写程序，跳转的地址错误，也会导致死机，这时候通常是 CPU“看错”指令，误把操作数当成指令，又把指令看成操作数，此时程序当然会乱运行，最后导致死机。

学习微处理机的指令的确是个苦差事，你必须学会数十个基本指令后，才能开始写简单的程序做实验，其中可能会花上数日或数星期的时间。本书尽量利用程序范例来解说指令，希望这些一连串的介绍和自我学习方法能让初学者尽快地熟悉 8051 的各项指令和用法。

6-2　8051 指令概述

8051 的 111 个指令中，最常用的单指令占 49 个，双字节指令则有 45 个，剩下的 17 个为三字节指令，在实际程序中，仍然以单字节和双字节指令较为常用，所以在学习阶段，我们应该先将学习重点放在单字节指令上，然后是双字节指令，最后才是三字节指令。

CPU 在执行程序时，它读取第一个字节的数据时，自然地将此字节认定为功能助记符，即此字节一定是一个指令（对 CPU 而言），经过内部的处理运算后，CPU 自动判断该指令是否到此为止，若是，则该字节就是一个单字节指令，若不是，则判断还要再获取一个或两个字节的数据，若是前者则为双字节指令，如果是后者就是三字节指令。另外由 CPU 的执行架构上来看，单字节指令的执行处理速度最快，双字节指令次之，三字节指令再次之。

8051 指令若依执行的功能来区分时，大致上可分成五大类，和传统的 8 位 CPU 较为不同的是，8051 特别加强位的处理能力，我们特别把该类指令归成另一大类，以区别于其他处

理指令，以下就是 8051 的 5 类指令的大略介绍。

1. 数据传送（Data Transfer）指令

此类指令是最常使用的指令，望文生义，所谓的数据传送，就是单片机中数据（8 位）的传送。例如，把累加器（ACC）的值移到其他寄存器上，或是把存储器中某个位置内的值传送某个寄存器上。这类指令中最常用的功能助记符就是 MOV，即所谓传送（MOVE）的意思。属于数据传送的指令，其开头的功能助记符分别是：

MOV：数据传送。

PUSH：数据寄存于堆栈（STACK）中。

POP：数据从堆栈中取回。

XCH：数据互换（Exchange）。

XCHD：数据互换，且只互换低半字节（4 位）。

MOVX：由外部存储器中取得或存入数据。

MOVC：由程序存储器中依照查表（TABLE）方式取得数据。

2. 算术运算（Arithmetic）指令

由于 8051 是属于 8 位的微处理器，所以它在进行算术运算时，也是以 8 位当成一个处理单元，可以提供 8 位没有正负号的算术运算，并且以溢位标志位（Overflow Flag）代表运算后是否有溢位或借位的情形发生。属于算术运算的指令其开头的功能助记符分别是：

INC：对指定的寄存器或地址加 1。

DEC：对指定的寄存器或地址减 1。

ADD：对指定的寄存器或地址加上一个 8 位（0～255）值。

ADDC：对指定的寄存器或地址加上一个 8 位值，另外再加上 Carry 进位标志位值。

SUBB：累加器（ACC）减去一个 8 位值，同时再减去进位值（CY）。

3. 逻辑运算（Logic）指令

逻辑运算就是对数据做逻辑功能的处理，其中包括 AND（与）、OR（或）、XOR（异或）、CPL（取反）及 ROTATE（移位）等等，当程序对某个事件必须做判断时，通常是采用逻辑运算指令将数个条件做逻辑上的判断，而得到一个 1（真）或 0（假）的结果，再依照结果去执行对应的操作。属于逻辑运算的指令，其开头的功能助记符分别是：

CLR：将某个寄存器或地址全部清除为 0。

CPL：将某个寄存器或地址取其取反值，即 1 变成 0，0 变为 1。

RL：将累加器 ACC 的值向左移位 1 位。

RR：将累加器 ACC 的值向右移位 1 位。

RLC：将累加器配合进位标志位向左移位 1 位。

RRC：将累加器配合进位标志位向右移位 1 位。

SWAP：将累加器中的位 0～位 3 和位 7～位 4 对调。

ANL：对寄存器或地址内容做 AND 逻辑运算。

ORL：对寄存器或地址内容做 OR 逻辑运算。

XRL：对寄存器或地址内容做 XOR 逻辑运算。

逻辑运算中必须特别注意的是，移位指令仅对 ACC 有效，其他寄存器无缘使用该指令。

4. 控制跳转（Control Transfer）指令

控制跳转指令是控制程序有条件或无条件地跳转到其他地址，去执行另一项操作，这些

指令通常用在逻辑指令之后，或是该控制跳转指令本身就有逻辑判断式。我们写程序中所使用的 CALL 调用子程序的指令正是一个典型的控制跳转指令。属于控制跳转的指令，其开头的功能助记符分别是：

ACALL：调用近程（2KB 以内）的子程序。

LCALL：调用远程（64KB 以内）的子程序。

RET：子程序执行完毕返回主程序。

RETI：中断服务程序执行完毕后返回原程序。

AJMP：短（2KB 以内）跳转。

LJMP：长（64KB 以内）跳转。

JMP @A+DPTR：查表方式的跳转指令。

JZ：若累加器为 0 则做跳转的操作。

JNZ：若累加器不为 0 则做跳转的操作。

CJNE：比较两个寄存器或地址的值，若不相同则做跳转的操作。

DJNZ：将某个寄存器或地址的内容减 1，若不为 0 则做跳转的操作，此指令使用的机会相当多，请特别注意其用法。

5. 布尔变量操作指令

传统的 8 位微处理器，如 Z80、8085 或 6502 等 CPU 并没有所谓的布尔变量操作指令，即无法针对寄存器或某个地址中的 1 个位的真或假进行处理，所以一个事件的判断结果必须以 1 字节（8 位）来存放，无形之中就浪费了其他 7 个位的空间，而在 8051 系列的 CPU 中，大部分的寄存器都可以用单一位做逻辑上的处理、判断或跳转，无形之中简化了程序的写法，另外在 CPU 中的内部数据存储器，也有 16 个字节的空间可以做单一字节的数据处理，相当于该区可存放 16×8=128 个事件的逻辑状态（仅有 0 与 1 两种情形），此种做法绝对可以减少 RAM 所占用的空间。属于布尔变量操作的指令，其开头的功能助记符分别是：

CLR：把该位清除为 0。

SETB：把该位清除为 1。

CPL：把该位取反，即取补数，原先若为 1，则变成 0，若为 0 时则变成 1。

ANL：把该位和进位标志位做逻辑 AND 的操作。

JC：若进位标志位为 1 则跳转到该地址。

JNC：若进位标志位不为 1 则跳转到该地址。

JB：若该位为 1 时，则跳转到该地址。

JNB：若该位不为 1 时，则跳转到该地址。

JBC：操作和 JB 相同，但条件判断完毕后，自动将该位清除为 0。

6-3 8051 指令集整理

为了方便查询 8051 指令，我们特别将 8051 所有的指令依两种形式列出，第一种方式是依功能划分，总共有 5 个指令功能表，分别是算术运算、逻辑运算、数据传送、布尔变量操作和程序控制跳转等，若对某个指令功能不明白时，查询这 5 类表将可立即知道正确的写法和功能。第二种指令显示的方式，是依机器码值的大小顺序而排列，该表中除了列出指令的正确写法外，同时指出该指令需要多少个字节的空间来存放，这对我们分析别人的程序时非

常有效。例如，当我们从一个未知的 8051 系统中，得到一段程序的机器码，分别是 34H、29H、6FH、73H 等 4 字节数据，那我们如何知道其程序意义呢？

首先我们查表得知，34H 是 ADDC A,#data 指令的第一个字节，所以接下来的 29H 即为 data 的值，所以前两个字节应该是 ADDC A,#29H 的意思，接下来查询 6FH 是 XRL A,R7 指令，且该指令仅占 1 字节，而 6FH 后的 73H 意义为 JMP@A+DPTR，所以由程序中的 4 个字节的内容，我们可以得到其真正的操作情形和程序流向。

```
ADDC    A,#29H      ;把 A 值加以 29H,结果仍存回 ACC 中
XRL     A,R7        ;把 A 值和 R7 寄存器值异或,结果存回 ACC JMP@A+DPTR A 值加 DPTR 后
                    ;得到新地址值,然后程序跳转到该地址继续执行
```

以上 4 个字节的数据竟然可推导出 3 行 8051 指令来，这种分析过程我们称之为逆向工程或还原工程（Reverse Engineering）。我们另外可用程序分析的方式判断出，该段程序的操作情形，以及在某些条件下该程序所反映的状况，我们另外可由此段程序得到一些假设和推论：

（1）此段程序后可能有某段程序区域都是跳转地址表。

（2）该程序要执行跳转前，必须先定义好 ACC 的值。

（3）该程序要执行跳转前，必须先定义好 DPTR 的值，而此 DPTR 值正是跳转表的第一个进入点。

（4）ACC 值和 R7 值做逻辑上的异或运算，所以可断定 R7 应该是某个条件值，而且该条件值应当在此段程序前被定义好。

依此原则及方法，我们可以将一个 8051 系统中的 ROM 程序完全解译出来。

本节所提供的功能区分指令表中，有些符号和写法必须在此做些说明：

Rn ：代表该位置可以是 R7～R0 共 8 个寄存器中的任意一个。

Direct ：代表 8051 中可以直接寻址到的地址值，这包括了 128 个数据存储器，任意一个 I/O 端口和控制/状态寄存器。

@Ri ：间接寻址到数据存储器的地址，在 8051 的指令中，仅有 R0 和 R1 可以有此功能，所以@R5 这种写法是不正确的。

#data ：代表一个 8 位宽度的立即数据值。

#data16：代表一个 16 位宽度的立即数据值，这种写法在 8051 指令中仅有一种，即 MOV DPTR,#addr16。

Bit ：可代表 8051 系统中的 128 个软件标志位、控制 I/O 点及状态位点，只要是 8051 中可以做位寻址的位点在此都如此表示。

Rel ：这是程序跳转时的相对位置（非绝对地址），此值仅有 8 位宽，所以程序能够跳转的距离有限，往前最多可跳转 128 个字节，往回跳转时最多可跳转 127 个字节。

addr16 ：代表 16 位的地址值，由于 16 位的值最大可达 64KB（65535），所以完全涵盖了 8051 所能执行的 64KB 程序空间。这种写法是使用在 LCALL（调用绝对地址）和 LJMP（跳转到绝对地址）指令上。

addr11 ：代表 11 位的地址值，这种用法是使用在 ACALL（调用短程地址）和 AJMP（短程地址跳转）指令上，这种方式只能在相同的 2KB 页上调用或跳转。

ACALL/AJMP 在作用上完全和 LCALL/LJMP 相同，唯一的差异是前者仅占 2 字节空间，后者却要 3 个字节来存放其指令，如果你想缩减程序所占的空间，且程序全部又不超过 2KB 的空间时，就可以尽量采用 ACALL/AJMP 的写法。

在本书的附录里有 8051 指令的总整理图表，分别按功能（Function）与 HEX 码（十六进制码）的顺序来排列，这些图表将是你日后写程序最重要的参考依据（请参阅本书附录 D 及附录 E）。

6-4 影响标志位的指令

凡是曾经做过的必有所影响，当我们以 8051 指令执行程序时，凡是属于算术运算、逻辑运算和某些位寻址的指令，该指令被执行完后，会将其重要的结果存入标志位寄存器中，标志位寄存器就是我们俗称的程序状态字符（PSW），会影响的标志位分别为 Carry 借位标志位、Overflow 溢位标志位和 Aux Carry 辅助借位标志位。其中以 Carry 借位标志位的使用几率最高。8051 影响标志位值的指令，如下表所示。

表 6-1 8051 影响标志位的指令

指 令	影响标志位		
	Carry	Overflow	Aux Carry
ADD	√	√	√
ADDC	√	√	√
SUBB	√	√	√
MUL	0	√	
DIV	0	√	
DAA	√		
RRC	√		
RLC	√		
SETB C	1		
CLR C	0		
CPL C	√		
ANL C,bit	√		
ANL C,/bit	√		
ORL C,bit	√		
ORL C,/bit	√		
MOV C,bit	√		
CJNE	√		

注：①打√代表会受影响；

②运算指令除上述指令外，其余指令均不影响 C、OV 和 AC 三个标志位；

③8051 的标志位当中以 Carry 的使用几率最高，请特别注意其用法；

④Carry 在 MUL 和 DIV 运算指令执行会被清除为 0。

由于 8051 的众标志位中并没有 Zero 零位标志位，所以当我们想判断某个寄存器的值是否为 0 时，通常要改采用 CJNE（比较是否相等再做跳转）的指令，这点在我们写程序时必须特别留意。

除了表 6-1 所列举的指令以外，其他的指令不论其写法如何都不会影响 8051 中 PSW 的 Carry、Overflow 和 Aux Carry 值，例如我们执行了 INC A 或 DEC A 的指令，不论以上指令被连续执行多少次，都不会影响 Carry 和 Overflow 的值，但是执行 MUL 和 DIV 指令时，Carry 却固定被设成 0。类似这种差异性在各种单片机 CPU 的指令中时常看到，若不事先知道这些特点，在写程序时就会暗藏危机，往往会误认为一个看起来绝对没有问题的程序，却一直无法顺利地执行程序。

6-5 8051 指令解析一：算术运算指令

算术运算指令在程序中的使用率相当高，其中又以 ADD（相加）和 INC（加 1）的使用率最高，在前几节中我们曾提及，ADD 指令会影响 CY、OV 和 AC 三个重要的标志位，而 INC（加 1）指令却不会影响上述的标志位。另外 8051 做 MUL 乘法和 DIV 除法指令时，会自动将 CY 进位标志位设置成 0，所以，我们应避免在上述两种指令后，做进位标志位的逻辑判断，因为此时 CY 一定为 0，很有可能造成程序的条件判断错误。

ADD A,<source-byte>

功能：加法。

说明：把指定位置的值与 ACC 累加器的值相加，结果放在 ACC 上。如果结果的位 7 有溢位时，CY 标志位会设成 1，若位 3 有溢位时，AC 标志位也会设成 1。当两个数值做没有正负符号的相加时，若结果有溢位时，CY 标志位会设成 1。

当两个正数相加产生负数的值或两个负数相加产生正数的结果时，OV 标志位会变成 1。

ADD A 指令共有 4 种寻址方式：

```
ADD   A,Rn          ;(A)<-(A)+(Rn)
ADD   A,direct      ;(A)<-(A)+(direct)
ADD   A,@Ri         ;(A)<-(A)+((Ri))
ADD   A,#data       ;(A)<-(A)+#data
```

ADDC A,<source-byte>

功能：包括进位的加法。

说明：ADDC 指令与 ADD 指令类似，唯一的差别是多加了 CY 标志位。

ADDC A 指令共有 4 种寻址方式：

```
ADDC  A,Rn          ;(A)<-(A)+(C)+(Rn)
ADDC  A,direct      ;(A)<-(A)+(C)+(direct)
ADDC  A,@Ri         ;(A)<-(A)+(C)+((Ri))
ADDC  A,#data       ;(A)<-(A)+(C)+#data
```

SUBB A,<source-byte>

功能：包括借位的减法。

说明：ACC 累加器内的值先减 CY 标志位值，再减去指定位置内的值，结果放进 ACC 内。如果有借位情况产生时，CY 标志位会被设成 1，反之则为 0。8051 单片机的减法一定要

通过 ACC 来处理。通常我们在做减法前会先执行 CLR C 指令，把 CY 标志位设成 0，以免影响第一次 SUBB 的结果。

SUBB A 指令有 4 种寻址方式：

```
SUBB A,Rn        ;(A)<-(A)-(C)-(Rn)
SUBB A,direct    ;(A)<-(A)-(C)-(direct)
SUBB A,@Ri       ;(A)<-(A)-(C)-((Ri))
SUBB A,#data     ;(A)<-(A)-(C)-#data
```

INC<byte>

功能：把指定的位置值加 1。

说明：如果该位置的值刚好是 FFH，则 INC 执行完后会成为 00H，但是，这个结果不会影响任何标志位。

INC 指令有 4 种寻址方式：

```
INC   A          ;(A)<-(A)+1
INC   Rn         ;(Rn)<-(Rn)+1
INC   direct     ;(direct)<-(direct)+1
INC   @Ri        ;((Ri))<-((Ri))+1
```

INC DPTR

功能：把 16 位的 DPTR 指针值加 1。

说明：8051 内部的 DPTR 可以拆成 DPH 与 DPL，当执行 INC DPTR 时，DPL 若是 FFH 时，DPL 会变成 00H，同时 DPH 内的值会加 1，这个指令也不影响任何标志位。

```
INC DPTR         ;(DPTR)<-(DPTR)+1
```

DEC byte

功能：把指定的位置值减 1。

说明：若指定的位置内刚好是 00H 时，DEC 指令执行完后会变成 FFH，但是不会影响任何标志位。

DEC 指令有 4 种寻址方式：

```
DEC  A           ;(A)<-(A)-1
DEC  Rn          ;(Rn)<-(Rn)-1
DEC  direct      ;(direct)<-(direct)-1
DEC  @Ri         ;((Ri))<-((Ri))-1
```

DEC DPTR

功能：把 16 位的 DPTR 值减 1。

说明：这是一个不存在的指令，Intel 当初在规划 8051 时就没有这个指令。如果你在汇编程序上写 DEC DPTR 一定会有错误出现。我们不知道当年开发 8051 指令时碰到何种阻碍，不过，结论就是没有 DEC DPTR 指令。如果真的要让 DPTR 减 1 的话，就请改用下面的写法：

```
DPTR_DEC    PUSH    PSW
            PUSH    A
CLR         C
MOV         A,DPL
SUBB        A,#01H
MOV         DPL,A
```

```
MOV         A,DPH
SUBB        A,#00H
MOV         DPH,A
POP         A
POP         PSW
```

MUL AB

功能：乘法运算。

说明：将累加器 ACC 与寄存器 B 进行 8 位无正负符号的乘法运算，产生 16 位的积，高字节（High order byte）存在寄存器 B 中，低字节（Low order byte）则放在 ACC 中。MUL 指令会影响标志位值。当结果大于 FFH 时，OV 标志位会设成 1，另外 CY 标志位则固定被清除成 0。

```
MUL   AB        ;(B)15-8,(A)7-0<-(A)×(B)
```

DIV AB

功能：除法运算。

说明：累加器 ACC 为被除数，寄存器 B 为除数，进行 8 位无正负符号的除法运算。结果的商放在 ACC 累加器上，余数放在寄存器 B。CY 与 OV 标志位都清除成 0。

```
DIV   AB        ;(A)15-8<-(A)/(B)
                ;(B)7-0
```

DA A

功能：累加器 ACC 在加法后的十进制调整。

说明：8051CPU 的加法都是二进制的，如果有两个都是十进制的值要做相加时，我们还是可以用 ADD 或 ADD C 指令，将两个值直接相加，然后再加上 DA A 指令，就可以将 ACC 值调整成十进制的值。

```
DA A            ;IF[[(A3-0)>9]v[(AC)=1]]then(A3-0)<-(A3-0)+6
                ;AND
                ;IF[[(A7-4)>9]v[(C)=1]]then(A7-4)<-(A7-4)+6
                ;DAA 同时调整两位数
```

范例说明 1

```
MOV  A,#12H     ;A 实际上是十进制的 12
MOV  B,#18H     ;B 实际上是十进制的 18
ADD  A,B        ;A=A+B=2AH
DA   A          ;A 的位 3-0>9,DAA 会自动补进 6,2AH+06H=30H
                ;A=30H 十进制的 30
```

范例说明 2

```
MOV  A,#40H     ;(BCD)=40
CLR  C
SUBB A,#10      ;A=A-10=36H
DA   A          ;A=3CH 非预期的 30H
```

当 SUBB 指令执行完后，ACC 值为 36H，且 AC 标志位值为 1，代表辅助进位有效。此时若再执行 DA A 指令，由于 CY=0，且 AC=1 则 ACC 值将加上 06H（而非减少 06H），使得结果成为 36H+06H=3CH，而非原先我们预期的 30H，所以显然 SUBB 之后加 DA A 的写法是不正确的。DA A 仅能根据 PSW 值中 CY 和 AC 标志位而对 ACC 值调整，而且只有加上

00H、06H、60H 或 66H 的功能，没办法做减的操作，因此，若值相减时也做十进制调整时，则要对程序做特别的安排。

范例说明 3

```
MOV   A,#40H          ;十进制的 40
ADD   A,#90H          ;A=A-10(BCD OPERATION)
DA    A               ;A=30H,即十进制的 30
RET
```

上一个例子中我们提到在 SUBB 指令后加 DA A 做十进制调整是不正确的，因为 DA A 仅针对 ADD 和 ADDC 指令有效，所以纵使是减法运算，也要把减的程序写成加法的样子，唯有如此才能获得正确值。如果要减 1，则程序只要写成 ACC 值加 99（100-1），然后再做 DAA 十进制调整即可得到正确值。如果要减 10，则程序必须写成加 90（100-10），再经 DA A 处理后就是正确值。

在本例子中，ACC 值原先是 40（BCD 值），减 10 后得到值 30（BCD），和我们在纸上计算的相同。8051 的 SUBB 指令无法应用在十进制的减法中，但在其他二进制的加减计算系统中，仍有其广泛的用途存在。

6-6 8051 指令解析二：逻辑运算与移位指令

8051 有相当强而有力的逻辑运算能力，可对 8 位数据做 AND、OR、XOR 和数据移位，这些运算的中心仍然是累加器 ACC，所有数据存储器上的数据以及寄存器都可与累加器做逻辑运算，结果则在累加器 ACC 上，在这些运算指令中，较为特殊的指令有 3 个：

```
ANL direct,A          ;某个地址内的值或寄存器和 ACC 作 AND,结果存回原地址上
ORL direct,A          ;某个地址内的值或寄存器和 ACC 作 OR,结果存回原地址上
XRL direct,A          ;某个地址内的值或寄存器和 ACC 作 XOR,结果存回原地址上
```

ANL<dest-byte>,<src-byte>

功能：对相关位置进行逻辑上的 AND 处理。

说明：做完逻辑上的 AND 之后，结果仍然存入 dest-byte 所指定的位置。不影响任何标志位值。ANL 共有 6 种寻址方式。当控制 8051 的输出端口（P0、P1、P2 和 P3）时，常会用到 ANL P1 或 ORL P1 等的写法，这种做法可以直接控制到对应端口的对应位，而且完全不必通过 ACC 累加器的帮忙，这种指令是传统的 8 位 CPU（Z80、6502 或 8085）所缺少的。

```
ANL  A,Rn             ;(A)<-(A)AND(Rn)
ANL  A,direct         ;(A)<-(A)AND(direct)
ANL  A,@Ri            ;(A)<-(A)AND((Ri))
ANL  A,#data          ;(A)<-(A)AND#data
ANL  direct,A         ;(direct)<-(direct)AND(A)结果直接存入 direct 指定的位置
ANL  direct,#data     ;(direct)<-(direct)AND#data
```

ANL direct，A 指令中的 direct 是一个可寻址的地址，所以它可以是一个寄存器、数据存储器中的位置或是一个担任输出入的端口。这类指令的影响是相当深远的，前面所提的 ANL 指令都是以累加器 ACC 为中心，做完 AND 运算后再将值转存到其他寄存器上。可是 ANL direct，A 指令的操作却有所不同，它直接把 direct 地址上的值和 ACC 做 AND，而结果

直接返回 direct 地址上，这些操作完全不经过 ACC 累加器。

ORL<dest-byte>,<src-byte>

功能：对相关位置进行逻辑上的 OR 处理。

说明：做完逻辑上的 OR 之后，结果仍然存入 dest-byte 所指定的位置。不影响任何标志位值。OR 共有 6 种寻址方式。ORL 指令的精神是将寄存器中的一个或数个位设为 1，而且不影响到其他位的状态。

```
ORL    A,Rn          ;(A)<-(A)OR(Rn)
ORL    A,direct      ;(A)<-(A)OR(direct)
ORL    A,@Ri         ;(A)<-(A)OR((Ri))
ORL    A,#data       ;(A)<-(A)OR#data
ORL    direct,A      ;(direct)<-(direct)OR(A)
ORL    direct,#data  ;(direct)<-(direct)OR#data
```

XRL<dest-byte>,<src-byte>

功能：对相关位置进行逻辑上的 XOR 异或处理。

说明：做完逻辑上的 XOR 之后，结果仍然存入 dest-byte 所指定的位置。不影响任何标志位值。XOR 共有 6 种寻址方式。XOR 指令主要是用在数据对比上，如果两者的状态一样，则其结果是 0，反之则是 1。如果两个 8 位的值经过异或逻辑运算后结果为 00H，这代表两个值是完全相等的，若结果不为 00H 时，表示两值绝对不相等。1 在 ANL、ORL 和 XRL 三种指令中，唯有 XRL 指令重复执行时，会有不同的结果出现。

```
XRL    A,Rn          ;(A)<-(A)XOR(Rn)
XRL    A,direct      ;(A)<-(A)XOR(direct)
XRL    A,@Ri         ;(A)<-(A)XOR((Ri))
XRL    A,#data       ;(A)<-(A)XOR#data
XRL    direct,A      ;(direct)<-(direct)XOR(A)
XRL    direct,#data  ;(direct)<-(direct)XOR#data
```

CLR A

功能：把 ACC 累加器清除为 0。

说明：本指令不影响任何标志位值。

```
CLR    A             ;(A)<-0,ACC=0000 0000b
```

CPL A

功能：把 ACC 累加器取反。

说明：寄存器内原先是 1 的变成 0，原先是 0 的变为 1，本指令不影响任何标志位值。

```
CPL    A             ;(A)<NOT(A)
```

RL A

功能：把累加器往左移 1 位。

说明：ACC 的位 7 移位入位 0，其余位均往左移 1 位，本指令不影响任何标志位值。左移指令相当是把 ACC 值乘 2 的运算。

```
RL     A             ;(An+1)<-(An)  n=0～6
                     ;(A0)<-(A7)
```

RLC A

功能：把累加器及 CY 标志位一起左移 1 位。

说明：ACC 的位 7 移入 CY 标志位内，原先的 CY 标志位则移入 ACC 的位 0，本指令会影响 CY 标志位。

```
RLC   A          ;(An+1)<-(An)  n=0～6
                 ;(A0)<-(C)
                 ;(C)<-(A7)
```

RR A

功能：把累加器往右移 1 位。

说明：ACC 的位 0 移位入位 7，其余位均往右移 1 位，本指令不影响任何标志位值。右移指令相当是把 ACC 值除 2 的运算。

```
RR    A          ;(An)<-(An+1)  n=0～6
                 ;(A7)<-(A0)
```

RRC A

功能：把累加器及 CY 标志位一起右移 1 位。

说明：ACC 的位 0 移入 CY 标志位内，原先的 CY 标志位则移入 ACC 的位 7，本指令会影响 CY 标志位。RR A 和 RRC A 是右移指令，8051 的移位指令仅能在累加器 ACC 上实施，其他寄存器则不行。

```
RRC   A          ;(An)<-(An+1)  n=0～6
                 ;(A7)<-(C)
                 ;(C)<-(A0)
```

SWAP A

功能：交换 ACC 累加器的上下各 4 位对调。

说明：SWAP 指令可看成 RR A 四次或 RL A 四次，不影响任何标志位。SWAP 指令是把累加器 ACC 中的位 0～位 3 和位 4～位 7 对调，通常在十进制转换和数值显示的场合下使用。只有 ACC 才支持 SWAP 指令，其他寄存器若要做 SWAP 操作时，要先将数据传送 ACC 上，做完 SWAP 后再回存入原来的寄存器。

```
SWAP  A          ;(A3-0)<-(A7-4),(A7-4)<-(A3-0)
```

6-7 8051 指令解析三：数据传送指令

8051 的数据传送指令是以 MOV 指令为重心，本章讲的所有寻址法对 MOV 指令都适用，学通了 MOV 指令就等于熟悉了一半以上的 8051 指令，另外要特别了解的指令是 MOVX，当我们欲与外部的数据存储器或 I/O 沟通时，MOVX 指令是唯一的通道。

MOV<dest-byte>,<src-byte>

功能：8 位数据的传送。

说明：把<src-byte>所指定位置内的值传送到<dest-byte>的地址内。<src-byte>的内容不会变，本指令也不影响任何标志位。MOV 指令共有 15 种寻址方式，请一定要详加研读。

```
MOV   A,Rn       ;(A)<-(Rn)
MOV   A,direct   ;(A)<-(direct)
MOV   A,@RI      ;(A)<-((Ri))
MOV   A,#data    ;(A)<-#data
```

```
MOV    Rn,A              ;(Rn)<-(A)
MOV    Rn,direct         ;(Rn)<-(direct)寄存器与 Rn 间的数据传送
MOV    Rn,#data          ;(Rn)<-#data
MOV    direct,A          ;(direct)<-(A)
MOV    direct,Rn         ;(direct)<-(Rn)
MOV    direct1,direct2   ;(direct1)<-(direct2)寄存器可以互相传送
MOV    direct,@Ri        ;(direct)<-((Ri))
MOV    direct,#data      ;(direct)<-#data
MOV    @Ri,A             ;((Ri))<-(A)
MOV    @Ri,direct        ;((Ri))<-(direct)
MOV    @Ri,#data         ;((Ri))<-#data
```

MOV DPTR,#data16

功能：16 位 DPTR 指针值的设置。

说明：本指令不影响任何标志位。DPTR 指针又可分为占高位的 DPH，与占低位的 DPL，data16 是一个 16 位的常数值。这个指令通常用在查表的程序或是做外部 I/O 地址寻址用。

```
MOV    DPTR,#data16      ;(DPTR)<-data16
                         ;(DPH)<-data15～8,(DPL)<-data7～0
```

MOVC A,@A+<base-reg>

功能：读入一个 8 位程序码。

说明：<base-reg>是一个 16 位的变数值，它可以是当时的程序计数值 PC（Program Counter），也可以是 DPTR 值，本指令是以 ACC 当成 OFFSET 值，加上 16 位的<base-reg>值，得到一个总和值，然后以此值为地址值，读入该位置的内容，并将结果放在 ACC 内。8051 的程序若有用到查表的功能时一定是使用 MOVC 指令。本指令不影响任何标志位。

```
MOVC   A,@A+DPTR         ;(A)<-((A)+(DPTR))
MOVC   A,@A+PC           ;(PC)<-(PC)+1 这里请特别注意
                         ;(A)<-((A)+(PC))
```

范例说明 1

在 Intel 的指令说明里特别提到使用"MOVC A,@A+PC"时要留意，当程序在执行时 Program Counter 是随时在变动的，请看以下的例子，如果此时累加器的内容是 02H，你认为 RET 时，ACC 的内容是 22H 吗？不是。ACC 的内容竟然是 11H，这是因为 MOVC 指令之后还有一个 RET 指令（占 1 字节），这会使得 TABLE 表的位置与我们预期的刚好相差 1 字节。

```
                         ;ACC=02H
G_TABLE  MOVCA,@A+PC
         RET             ;返回原程序
TABLE    00H             ;5 字节的 TABLE 表
         11H
         22H
         33H
         44H
```

所以正确的写法应该是：

```
                         ;ACC=02H
G_TABLE  INC A           ;A=A+1=03H,补偿 RET 所占用的 1 字节空间
```

```
        MOVCA,@A+PC
        RET             ;返回原程序,ACC=22H
TABLE   00H             ;5 字节的 TABLE 表
        11H
        22H
        33H
        44H
```

或是改用 DPTR 来取得 TABLE 值：

```
                        ;ACC=02H
                        ;ACC 值不要调整
G_TABLE MOV DPTR,#TABLE
        MOVCA,@A+DPTR
        RET             ;返回原程序,ACC=22H
TABLE   00H             ;offset 00H
        11H             ;offset 01H
        22H             ;Offset 02H
        33H             ;offset 03H
        44H             ;offset 04H
```

MOVX <dest-byte>,<src-byte>

功能：外部数据的传送。

说明：8051 单片机与外部的数据存储器或 I/O 的读取或写入，一定是用这个 MOVX 指令来操作。指令助记符上的 X 就是 eXternal 的意思。MOVX 指令只与 ACC 累加器有关，所以要送到外部的数据一定要先放在 ACC 上才行。当本指令执行时，8051CPU 会伴随送出必要的 16 位地址线到 Port2（A15～A8）与 Port0（A7～A0）上，而且 RD（P3.7）与 WR（P3.6）线也会变化，以便读回或写入值。

```
MOVX    A,@DPTR         ;(A) <-((DPTR))外部数据的读回
MOVX    @DPTR,A         ;((DPTR))<-(A)外部数据的写入
MOVX    @Ri,A           ;((Ri))<-(A)这个指令已经很少用
MOVX    A,@Ri           ;(A)<-((Ri))这个指令已经很少用
MOVX    @DPTR,SBUF      ;8051 没有这种指令
MOVX    R0,@DPTR        ;8051 也没有这种指令
```

PUSH direct

功能：将指定的数据推入堆栈中。

说明：这个指令是将 8 位数据送进堆栈（STACK）中，执行后堆栈 SP 值加 1。本指令不影响任何标志位。8051 的堆栈是放在 128 字节的数据存储器上，若程序刚开始时设置 SP=5FH，则堆栈有 32 字节（60H～7FH）之多，如果系统使用中断的机会很多时，就要考虑是否有堆爆（Stack Overflow）的可能，因为中断服务程序在执行前，会做多次数据暂时存入堆栈的操作，有可能两次中断就把堆栈用满了。如果要把 R0 的值存入堆栈中，应该要写成 PUSH 00H，若写 PUSH R0 则在编译时会有错误出现。

```
PUSH  direct            ; (SP) <- (SP) +1
                        ; ((SP)) <- (direct)
```

POP direct

功能：由堆栈区取回存放值。

说明：从堆栈区取回一个 8 位数据值，执行后堆栈 SP 值减 1。本指令不影响任何标志位。在 8051 的汇编语言程序当中，每使用一个 PUSH 指令，一定要对应令一个 POP 指令，而且是 PUSH 在前 POP 在后。使用堆栈指令的要诀是 PUSH 与 POP 两者出现的距离尽量靠近，以免造成误解。PUSH 与 POP 指令必须小心使用，否则系统会死机。

```
POP direct          ;(direct)<-(SP)
                    ;(SP)<-(SP)-1
```

范例说明

```
;使用 PUSH 与 POP 指令前一定要先设置 SP 值
MOV     SR,#5FH     ;堆栈区由 60H 往后延伸
…
…
PUSH    A           ;ACC 暂时存放在堆栈中,(60H)=ACC,SP=60H
MOV     A,P1        ;取得 P1 的状态值
MOV     DPTR,#8000H
MOVX    @DPTR,A     ;储存 P1 的状态值
POP     A           ;取回原先的 ACC,SP=5FH
PUSH    A           ;ACC 存入 STACK 中
POP     B           ;STACK 的值放回 B,相当于 MOV B,A 传送的指令
```

XCH A,<byte>

说明：把 ACC 值与指定位置内的值互换。

功能：XCH 指令限定 ACC 累加器与其他位置内的值互换，本指令也不影响任何标志位。

```
XCH     A,Rn        ;(A)<->(Rn)ACC 与 R0～R7 值互换
XCH     A,direct    ;(A)<->(diect)
XCH     A,@Ri       ;(A)<->((Ri))
```

范例说明

如果要把 ACC 与 20H 内的值互换时，有下面几种写法：

```
PUSH    20H         ;(SP)<-20H
MOV     20H,A       ;(20H)=A
POP     A           ;A=(20H)
```

或是靠堆栈来调整：

```
PUSH    A
PUSH    20H
POP     A           ;(A)<-(20H)
POP     20H         ;(20H)<-A
```

或是直接用 XCH A,20H 指令，而且只要 1 个机器周期就可完成。

XCHD A,@Ri

说明：ACC 的低 4 位（low-order nibbles）与 Ri 所指定地址的低 4 位对调。

功能：交换对调后，高 4 位的值不变，这个指令经常用在数值显示的应用上。本指令也不影响任何标志位。

```
XCHD    A,@Ri       ;(A3～0)<->((Ri3～0))
```

6-8 8051 指令解析四：布尔变量操作指令

8051 的许多指令中，对于单一位的处理指令包括清除、设置、取反、AND 运算和 OR 运算，不过并没有 XOR 异或指令，整个布尔变量操作指令皆以 CY 标志位为重心，而且 AND 和 OR 运算都是和 CY 做处理，除此之外，亦可使用 JB 或 JNB 指令做条件式跳转，对布尔变量操作指令懂得愈透彻，就愈能充分利用内部的数据存储器空间，同时又能有效节省程序空间。

CLR bit

功能：清除位地址的单一位。

说明：只要是 8051 单片机上注明 bit addressable 的位置都可以使用本指令。

```
CLR     C           ;(C)<-0
CLR     bit         ;(bit)<-0
```

SETB bit

功能：将指定的位地址设成 1。

说明：只要是 8051 单片机上注明 bit addressable 的位置都可以使用本指令。

```
SETB    C           ;(C)<-1
SETB    bit         ;(bit)<-1
```

CPL bit

功能：直接寻址位取反。

说明：只要是 8051 单片机上注明 bit addressable 的位置都可以使用本指令。

```
CPL     C           ;(C)<-NOT(C)
CPL     bit         ;(bit)<-NOT(bit)
```

ANL C,<src-bit>

功能：CY 标志位与位地址的单一位做 AND 运算，结果存在 CY 标志位上。

说明：只要是 8051 单片机上注明 bit addressable 的位置都可以使用本指令。

```
ANL     C,bit       ;(C)<-(C)AND(bit)
ANL     C,/bit      ;(C)<-(C)AND(NOT(bit))指定的位取反后才做 AND 运算
```

ORL C,<src-bit>

功能：CY 标志位与可位寻址的单一位做 OR 运算，结果存在 CY 标志位上。

说明：只要是 8051 单片机上注明 bit addressable 的位置都可以使用本指令。

```
ORL     C,bit       ;(C)<-(C)OR(bit)
ORL     C,/bit      ;(C)<-(C)OR(NOT(bit))指定的位取反后才做 OR 运算
```

MOV<dest-bit>,<src-bit>

功能：单一位的位数据传送。

说明：本指令其中一个位必须为 CY 标志位。

```
MOV     C,bit       ;(C)<-(bit)
MOV     bit,C       ;(bit)<-(C)
```

JC rel

功能：如果 CY=I 时，则程序跳转 rel 的位置，反之，则程序继续执行，本指令不影响标

志位值。

JNC rel

功能：如果 CY=0 时，则程序跳转 rel 的位置，反之，则程序继续执行。

说明：在 8051 的各种条件式跳转指令中，JC 和 JNC 使用的机会最高，每当 8051 执行完一个指令后，CPU 会自动调整程序计数器 PC 的值，由于 JC 与 JNC 都是两字节的指令，所以执行完该指令后，PC 值已经加 2 了，这时 CPU 内部会再依 CY 标志位的真伪，再次调整 PC 的值。本指令不影响标志位值。

```
JC    rel       ;(PC)<-(PC) +2
                ;IF(C)=1 THEN(PC)<-(PC)+rel
JNC   rel       ;(PC)<-(PC) +2
                ;IF(C)=0 THEN(PC)<-(PC)+rel
```

JB bit,rel

功能：如果该单一位=1 时，调整 CPU 的 PC 值。

JNB bit,rel

功能：如果该单一位=0 时，调整 CPU 的 PC 值。

JBC bit,rel

功能：如果该单一位=1 时，调整 CPU 的 PC 值，并且清除该位的内容。

说明：直接判断某单一位的状态，如果是真的话，直接对程序做跳转的操作。这些单一位的状态不需要传递到 ACC 累加器，就可直接处理。在 8051 的串行通信中，经常要检查 RI 位，若 RI=1 代表已经有串行数据进来了，这时我们要尽快读回 SBUF，并且立刻清除 RI 的状态，这时就可以用 JBC 这个指令来简化程序。

```
JB    bit,rel   ;(PC)<-(PC)+3
                ;IF(bit)=1 THEN(PC)<-(PC)+rel
JNB   bit,rel   ;(PC)<-(PC)+3
                ;IF(bit)=0 THEN(PC)<-(PC)+rel
JBC   bit,rel   ;(PC)<-(PC) +3
                ;IF(bit)=1 THEN(PC)<-(PC)+rel,(bit)=0
```

6-9 8051 指令解析五：程序分支指令

8051 共有 12 种程序分支的指令，其中较为重要且常用的指令有 LCALL、LJMP 和 CJNE、DJNZ 等指令，熟悉这些指令后可以使我们所写的程序更为简捷。

ACALL addr11

功能：绝对式调用子程序。

说明：本指令只占用两字节，ACALL 它有一个限制：只能调用到与 PC 相同的 2KB 程序空间区域。以 8051 为例，其内部有 4KB 的程序空间，若分成两份则每份有 2KB，分别占用 0000H～07FFH（前段）与 0800H～0FFFH（后段）。如果程序的 PC 值小于 0800H 时，它就没办法调用放在后段的子程序，反之亦然。8051 有这种指令的目的是减少程序的空间，如果你的程序做了 100 次的子程序调用，选用 ACALL 会比 LCALL 少 100 字节的空间。

另一方面要注意的是每做一次调用子程序，会耗掉堆栈两字节的空间，如果程序开头设置堆栈有 32 字节时，连续调用 16 次子程序，就有可能使堆栈爆掉。

```
ACALL  addr      ;(PC)<-(PC)+2
                 ;(SP)<-(SP)+1
                 ;((SP))<-(PC7～0)将要返回的地址存入堆栈中
                 ;(SP)<-(SP)+1
                 ;((SP))<-(PC15～8)将要返回的地址存入堆栈中
                 ;(PC10-0)<-Page Address
```

LCALL addr16

功能：远程调用。

说明：LCALL 调用就没有相同 2KB 位置的限制，所以它可以调用到整个 64KB 的程序空间。LCALL 指令需要 3 字节的存放空间。如果你的汇编语言产生的二进制文件空间小于 2048 字节，那程序的调用可以全部采用 ACALL，若超过 2048 字节时，可能有部分调用要改成 LCALL 才行。这种人工式的判断好像不是很正确。其实我们在写 8051 的汇编程序时，可以把调用全部写成 CALL，让汇编程序自动来帮你做判断。不过，有些汇编程序可能没有此功能，那就要自己判断了。

```
LCALL   addr     ;(PC)<-(PC)+3
                 ;(SP)<-(SP)+1
                 ;((SP))<-(PC7～0)
                 ;(SP)<-(SP)+1
                 ;((SP))<-(PC15～8)
                 ;(PC)<-addr15～0
```

RET

功能：从子程序返回。

说明：不论是 ACALL 或 LCALL 调用的子程序，都以 RET 为返回的指令，CPU 会从堆栈中取得先前存入的 16 位 PC 值，继续执行接下来的程序，RET 执行后堆栈值会加 2。

```
RET              ;(PC15～8)<-((SP))
                 ;(SP)<-(SP)-1
                 ;(PC7～0)<-((SP))
                 ;(SP)<-(SP)-1
```

RETI

功能：从中断服务程序返回。

说明：8051 的中断机制是很复杂的，在稍后的章节里我们会做更详细的分析。当系统允许中断后，只要中断的条件一出现，8051 会自动把当时的程序计数值 PC 存入堆栈中，并且跳转到对应的中断服务程序上，开始进行中断的处理程序，最后才以 RETI 指令当成该服务程序的结束。如果我们误用了 RET 指令，可能会使下个中断永远无法再发生。

```
RETI             ;(PC15～8)<-((SP))
                 ;(SP)<-(SP)-1
                 ;(PC7～0)<-((SP))
                 ;(SP)<-(SP)-1 中断一定要用 RETI 指令
```

SJMP rel

功能：短跳转。

AJMP addr11

功能：2KB 限制跳转。

LJMP addr16

功能：64KB 长距离跳转。

说明：上面 3 个都是跳转的指令，可以依我们实际的需要选择合适的 JUMP 指令。AJMP 与 ACALL 一样有 2KB 的跳转限制。LJMP 可以跳转到程序 64KB 中任一位置，而 SJMP 则只能选择往后 127 字节或往回跳转 128 字节。LJMP 指令占 3 字节，AJMP 与 SJMP 都占两字节，可是执行时，这 3 个指令都要耗掉两个机器周期。

```
AJMP    addr11  ;(PC)<-(PC)+2
                ;(PC10～0)<-addr11
LJMP    addr16  ;(PC)<-addr15～0
SJMP    rel     ;(PC)<-(PC)+2
                ;(PC)<-(PC)+rel
```

JMP @A+DPTR

功能：间接式跳转。

说明：在 8051 的汇编语言程序中，常有一些条件式的跳转写法。例如，如 ACC=0 跳转进入点 1，ACC=1 跳转进入点 2，ACC=2 跳转进入点 3 等等。这种应用就可以采 JMP @A+DPTR 的写法，间接跳转指令是伴随 AJMP 或 LJMP 指令一起使用的。DPTR 先指引到跳转表的最开头，而 ACC 则是条件的类型。

```
JMP @A+DPTR     ;(PC)<-(A)+(DPTR)
```

范例说明

```
    MOV  DPTR,#JMP_TBL
    ADD  A,A        ;A=A*2,因为下面每个跳转表各占 2 字节
    JMP  @A+DPTR ;(PC)=(A)+(DPTR)
;
JMP_TBL
    AJMP  COND0  ;2 字节指令
    AJMP  COND1  ;2 字节指令
    AJMP  COND2  ;2 字节指令
    AJMP  COND3  ;2 字节指令
```

JZ rel

功能：如果 ACC=0 则跳转。

JNZ rel

功能：如果 ACC 不等于 0 则跳转。

说明：8051 并没有 Zero 标志位，所以当要“以 HACC 是否为 0”做跳转时，就可以使用 JZ 与 JNZ 指令。本指令只做判断，但不影响 ACC 累加器的值，也不影响任何标志位。

```
JZ   rel        ;(PC)<-(PC)+2
                ;IF(A)=0 THEN(PC)<-(PC)+rel
JNZ  rel        ;(PC)<-(PC)+2
                ;IF(A)<>0 THEN(PC)<-(PC)+rel
```

CJNE<dest-byte>,<src-byte>,rel

功能：比较两个值，若不相等就做跳转的操作。

说明：这是 8051 比较特殊的指令，CPU 内部会拿<dest-byte>减去<src-byte>，这个结果

会影响 CY 标志位。当相减的结果不为 0 时，就做跳转的操作，反之则程序不对 PC 做调整，通常在此跳转位置再加一个 JC 指令，就可得知何者的值较大。

CJNE 有 4 种寻址模式：

```
CJNE    A,direct,rel    ;(PC)<-(PC)+3
                        ;IF(direct)<(A)THEN(PC)<-(PC)+rel,(C)<-0
                        ;OR
                        ;IF(direct)>(A)THEN(PC)<-(PC)+rel,(C)<-1
CJNE    A,#data,rel     ;(PC)<-(PC)+3
                        ;IF#data<(A)THEN(PC)<-(PC)+rel,(C)<-0
                        ;OR
                        ;IF#data>(A)THEN(PC)<-(PC)+rel,(C)<-1
CJNE    Rn,#data,rel    ;(PC)<-(PC)+3
                        ;IF#data<(Rn)THEN(PC)<-(PC)+rel,(C)<-0
                        ;OR
                        ;IF#data>(Rn)THEN(PC)<-(PC)+rel,(C)<-1
CJNE    @Ri,#data,rel   ;(PC)<-(PC)+3
                        ;IF#data<((Ri))THEN(PC)<-(PC)+rel,(C)<-0
                        ;OR
                        ;IF#data>((Ri))THEN(PC)<-(PC)+rel,(C)<-1
```

DJNZ<byte>,rel

功能：指定位置内容减 1，若结果不是 0 则做跳转。

说明：DJNZ 是 8051 指令中相当好的指令，当某个操作或程序要执行 10 次时，我们可以在数据存储器中找一个不被干扰的位置当计数值，先把它设成 10，每执行操作一次后，就做一次 DJNZ 指令，做完 10 次后就停止。我们若把 DJNZ 写成嵌套的结构，那就可以把计数值提高数千或数万次之多。

```
DJNZ    Rn,rel          ;(PC)<-(PC)+2
                        ;(Rn)<-(Rn)-1
                        ;IF(Rn)<>0 THEN(PC)<-(PC)+rel

DJNZ    direct,rel
                        ;(PC)<-(PC)+2
                        ;(direct)<-(direct)-1
                        ;IF(direct)<>0 THEN(PC)<-(PC)+rel
```

范例说明

假设我们要判断某个地址内的值减 1 后是否为 0，若不为 0 时则做条件式跳转，此时程序的写法应该类似下面的例子：

```
MOV     A,20H           ;取到(20H)内的值
DEC     A               ;A 值减 1
MOV     20H,A           ;减 1 后的值存回(20H)内
JZ      ADDR            ;若 A 等于 0 时跳转到 ADDR 地址
```

若写得更简单一点：

```
DEC     20H             ;(20H)值减 1
MOV     A,20H           ;取到(20H)值
JZ      ADDR            ;若 A 等于 0 时跳转到 ADDR 位置
```

如果你会使用 DJNZ 指令，程序就只剩下一行

```
DJNZ 20H,NEXT          ;(20H)值直接减 1 再做是否等于 0 判断，若条件成立时跳转 NEXT 地址
ADDR: …
```

NOP

功能：不做任何操作。

说明：不做任何操作的指令竟然可以存在!其实 NOP 指令纯粹是用来做延迟的，如果你想在 P1.0 产生一个 10μs 的数字信号，程序开头一定是 SETB P1.0 指令，程序最后则是 CLR P1.0 指令，中间呢?可能是 7～8 个 NOP 指令，至于确实的个数可能要用示波器做进一步时间的确认。

```
NOP                    ;(PC)<-(PC)+1
```

表 6-2　8051 指令一览表（一）

	0	1	2	3	4	5	6	7
0	NOP	JBC bit,rel [3B,2C]	JB bit,rel [3B,2C]	JNB bit,rel [3B,2C]	JC rel [2B,2C]	JNC rel [2B,2C]	JZ rel [2B,2C]	JNZ rel [2B,2C]
1	AJMP (P0) [2B,2C]	ACALL (P0) [2B,2C]	AJMP (P1) [2B,2C]	ACALL (P1) [2B,2C]	AJMP (P2) [2B,2C]	ACALL (P2) [2B,2C]	AJMP (P3) [2B,2C]	ACALL (P3) [2B,2C]
2	LJMP addr16 [3B,2C]	LCALL addr16 [3B,2C]	RET [2C]	RETI [2C]	ORL dir,A [2B]	ANL dir,A [2B]	XRL dir,A [2B]	ORL C,bit [2B,2C]
3	RR A	RRC A	RL A	RLC A	ORL dir,#data [3B,2C]	ANL dir,#data [3B,2C]	XRL dir,#data [3B,2C]	JMP @A+DPTR [2C]
4	INC A	DEC A	ADD A,#data [2B]	ADDC A,#data [2B]	ORL A,#data [2B]	ANL A,#data [2B]	XRL A,#data [2B]	MOV A,#data [2B]
5	INC dir [2B]	DEC dir [2B]	ADD A,dir [2B]	ADDC A,dir [2B]	ORL A,dir [2B]	ANL A,dir [2B]	XRL A,dir [2B]	MOV dir,#data [3B,2C]
6	INC @R0	DEC @R0	ADD A,@R0	ADDC A,@R0	ORL A,@R0	ANL A,@R0	XRL A,@R0	MOV @R0,@data [2B]
7	INC @R1	DEC @R1	ADD A,@R1	ADDC A,@R1	ORL A,@R1	ANL A,@R1	XRL A,@R1	MOV @R1,#data [2B]
8	INC R0	DEC R0	ADD A,R0	ADDC A,R0	ORL A,R0	ANL A,R0	XRL A,R0	MOV R0,#data [2B]
9	INC R1	DEC R1	ADD A,R1	ADDC A,R1	ORL A,R1	ANL A,R1	XRL A,R1	MOV R1,#data [2B]
A	INC R2	DEC R2	ADD A,R2	ADDC A,R2	ORL A,R2	ANL A,R2	XRL A,R2	MOV R2,#data [2B]

续表

	0	1	2	3	4	5	6	7
B	INC R3	DEC R3	ADD A,R3	ADDC A,R3	ORL A,R3	ANL A,R3	XRL A,R3	MOV R3,#data [2B]
C	INC R4	DEC R4	ADD A,R4	ADDC A,R4	ORL A,R4	ANL A,R4	XRL A,R4	MOV R4,#data [2B]
D	INC R5	DEC R5	ADD A,R5	ADDC A,R5	ORL A,R5	ANL A,R5	XRL A,R5	MOV R5,#data [2B]
E	INC R6	DEC R6	ADD A,R6	ADDC A,R6	ORL A,R6	ANL A,R6	XRL A,R6	MOV R6,#data [2B]
F	INC R7	DEC R7	ADD A,R7	ADDC A,R7	ORL A,R7	ANL A,R7	XRL A,R7	MOV R7,#data [2B]
	8	9	A	B	C	D	E	F
0	SJMP REL [2B，2C]	MOV DPTR,#data 16 [3B,2C]	ORL C,/bit [2B,2C]	ANL C,/bit [2B,2C]	PUSH dir [2B,2C]	POP dir [2B,2C]	MOVXA, @DPTR [2C]	MOVX @DPTR,A [2C]
1	AJMP （P4） [2B,2C]	ACALL （P4） [2B,2C]	AJMP （P5） [2B,2C]	ACALL （P5） [2B,2C]	AJMP （P6） [2B,2C]	ACALL （P6） [2B,2C]	AJMP （P7） [2B,2C]	ACALL （P7） [2B,2C]
2	ANL C,bit [2B,2C]	MOV bit,C [2B,2C]	MOV C,bit [2B]	CPL bit [2B]	CLR bit [2B]	SETB bit [2B]	MOVX A,@R0 [2C]	MOVX @R0,A [2C]
3	MOVC A,@A+PC [2C]	MOVC A, @A+DPTR [2C]	INC DPTR [2C]	CPL C	CLR C	SETB C	MOVX A,@RI [2C]	MOVX @RI,A [2C]
4	DIV AB [2B,4C]	SUBB A,#data [2B]	MUL AB [4C]	CJNE A, #data,rel [3B,2C]	SWAP A	DA A	CLR A	CPL A
5	MOV dir,dir [3B,2C]	SUBB A,dir [2B]		CJNE A,dir,rel [3B,2C]	XCH A,dir [2B]	DJNZ dir,rel [3B,2C]	MOV A,dir [2B]	MOV dir,A [2B]
6	MOV dir,@R0 [2B,2C]	SUBB A,@R0	MOV @R0,dir [2B,2C]	CJNE @R0,#data,rel [3B,2C]	XCH A,@R0	XCHD A,@R0	MOV A,@R0	MOV @R0,A
7	MOV dir,@R1 [2B,2C]	SUBB A,@R1	MOV @R1,dir [2B,2C]	CJNE @R1,#data,rel [3B,2C]	XCH A,@R1	XCHD A,@R1	MOV A,@R1	MOV @R1,A
8	MOV dir,R0 [2B,2C]	SUBB A,R0	MOV R0,dir [2B,2C]	CJNE R0,#data,rel [3B,2C]	XCH A,R0	DJNZ R0,rel [2B,2C]	MOV A,R0	MOV R0,A
9	MOV dir,R1 [2B,2C]	SUBB A,R1	MOV R1,dir [2B,2C]	CJNE R1,#data,rel [3B,2C]	XCH A,R1	DJNZ R1,rel [2B,2C]	MOV A,R1	MOV R1,A

续表

	0	1	2	3	4	5	6	7
A	MOV dir,R2 [2B,2C]	SUBB A,R2	MOV R2,dir [2B,2C]	CJNE R2,#data,rel [3B,2C]	XCH A,R2	DJNZ R2,rel [2B,2C]	MOV A,R2	MOV R2,A
B	MOV dir,R3 [2B,2C]	SUBB A,R3	MOV R3,dir [2B,2C]	CJNE R3,#data,rel [3B,2C]	XCH A,R3	DJNZ R3,rel [2B,2C]	MOV A,R3	MOV R3,A
C	MOV dir,R4 [2B,2C]	SUBB A,R4	MOV R4,dir [2B,2C]	CJNE R4,#data,rel [3B,2C]	XCH A,R4	DJNI R4,rel [2B,2C]	MOV A,R4	MOV R4,A
D	MOV dir,R5 [2B,2C]	SUBB A,R5	MOV R5,dir [2B,2C]	CJNE R5,#data,rel [3B,2C]	XCH A,R5	DJNZ R5,rel [2B,2C]	MOV A,R5	MOV R5,A
E	MOV dir,R6 [2B,2C]	SUBB A,R6	MOV R6,dir [2B,2C]	CJNE R6,#data,rel [3B,2C]	XCH A,R6	DJNZ R6,rel [2B,2C]	MOV A,R6	MOV R6,A
F	MOV dir,R7 [2B,2C]	SUBB A,R7	MOV R7,dir [2B,2C]	CJNE R7,#data,rel [3B,2C]	XCH A,R7	DJNZ R7,rel [2B,2C]	MOV A,R7	MOV R7,A

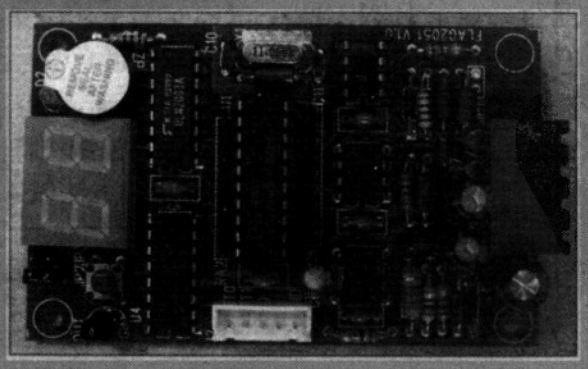
注：[2B]=2 Byte，[3B]=3Byte，[2C]=2Cycle，[4C]=4 Cycle，Blank=1 byte/1 cycle。

习　题

1. 8051 的指令格式的组成是什么？
2. Data Transfer 指令功能是什么？并列举 3 项属于该类别的指令。
3. Arithmetic 指令功能是什么？并列举 3 项属于该类别的指令。
4. Logic 指令功能是什么？并列举 5 项属于该类别的指令。
5. Control Tranfer 指令功能是什么？并列举 5 项属于该类别的指令。
6. 什么是布尔变量操作指令？
7. #data 代表的意义是什么？
8. 利用查表找出下列指令的功能：

```
MOV    A,Rn
LCALL  1000H
XRL    A,@Ri
XCH    A,20H
MUL    AB
```

7



AT2051 控制板上有一颗串行 E^2PROM24LC16，它是采用 I^2C 的传输方式，可储存系统的重要参数值

第 7 章　8051 单片机的引脚说明

单片机的应用是无限的，在本章里我们将对单片机 8051 的引脚与功能做一番详细的说明。8051 单片机比较特殊的是其 P3 端口，除了可做一般的 I/O 来使用外，也可以做特殊的用途。例如：串行传输的输入/输出、外部中断和外部扩展的 WR/RD 信号等等。另外我们也会谈到 AT89C2051 的引脚，虽然只有 20 根引脚，但是它还是保有许多 8051 原有的特性，只是无法进行外部扩充而已。

7-1　8051 单片机的引脚

在市面上最常见的 8051 系列的单片机依序是 AT89C51、AT89C52 和 8051，虽然 8051 内部有 4KB×8 的程序存储器空间，但是仍有存储器空间不足的情形发生，所以在实际的应用场合中，有些设计会因为程序经常修改的缘故，舍去内部程序存储器，而直接采用所谓的外部程序存储器（External Program Memory），不过这种做法将使 8051 端口 0 和端口 2 的功能失效，有所得也有所失，至于何种方法较适宜就看系统上的规划。

以下就是单片机 8051 的详细引脚说明，请特别留意上述芯片在引脚功能上的差异。图 7-1 是 8051 的引脚图，图 7-2 则是 8052 的引脚图。

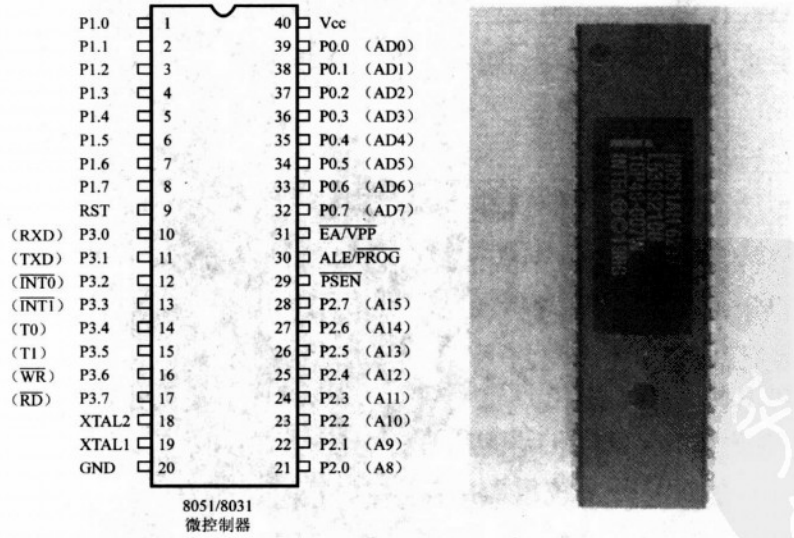


图 7-1　8051 的引脚图

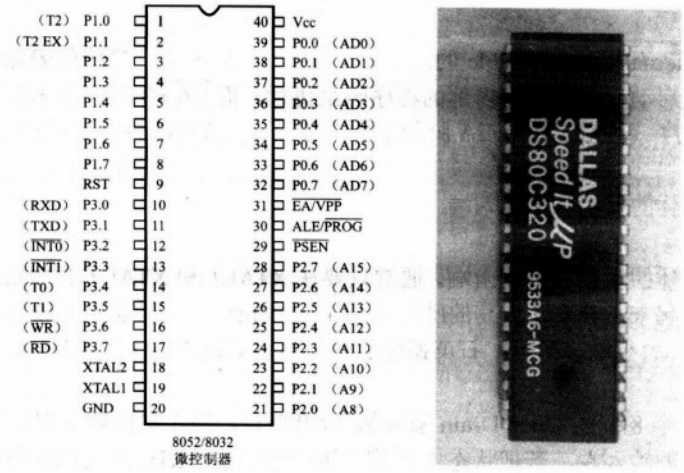


图 7-2　8052 的引脚图

Vcc

电源正端，容许正常工作的电压是+5V±10%，这也就是说由 4.5～5.5V 都能使 CPU 工作。

Vss

电源地端，所有的输入/输出信号都以此点为参考电压点。例如，我们说 P1.0 的输出电压为 4.0V 时，代表此输出端对 Vss 点的电压差为 4.0V。

RST

CPU 的复位 Reset 脚，平常应当在低电位的状态，当要对 CPU 做 Reset 时，只要将此脚升到逻辑 1 的状态，并且保持两个机器周期以上的时间，单片机将立刻进行系统复位的各项操作，并且开始由 0000H 地址上读入程序码。

ALE

ALE 是 Address Latch Enable 的缩写，当 CPU 要读取外部的程序或数据时，必定伴随此信号，Intel 发展单片机 CPU 时，为了有效减少引脚数，所以利用多任务（Multiplex）的方式送出地址值和数据值，并且以 ALE 的状态来指示现在送出的值是 Address 或 Data。如果是 AT89C51 内置有 FLASH 程序型的单片机，该脚在烧录程序时会被当成 PROG 的特殊功能输入脚。在正常情况下，ALE 信号的输出频率约是系统振荡频率的六分之一，所以可以拿来做其他芯片的时钟（Clock）输入用，不过 8051 做外部数据存储器的读写操作时，ALE 信号会暂停输出一个脉冲，关于 ALE 信号的解析请看《8051 单片机彻底研究——实习篇》的“8051 时序彻底研究”一章中的说明。

$\overline{\text{PSEN}}$

这是 Program Store Enable 的缩写，当 8051 单片机被设定成读外部程序存储器时，会送出此信号以便得到程序码。当单片机从外部程序存储器上读取程序码时，$\overline{\text{PSEN}}$ 在一个机器周期内总共变化两次，并且在 $\overline{\text{PSEN}}$ 的上升沿时读入一个 8 位程序运算码或操作码，和 ALE 相同的是当程序做外部数据存储器的读写操作时，$\overline{\text{PSEN}}$ 会一直保持在高电位，以免其他数据的进出和程序的数据码输出相冲突。

$\overline{EA}$

EA 代表 External Access Enable 的意思，$\overline{EA}$ 字母上面加上一横线代表低电位时操作。当 $\overline{EA}$ =1 时，系统启动后一定取用内部的程序码来执行，而 $\overline{EA}$ =0 时，系统会取用外部的程序存储器来执行程序。正常工作时 EA 脚的电位应该是 5V 或 0V，不应该有其他的电压值。

XTAL1

系统振荡晶体的反相放大输入端。

XTAL2

系统振荡晶体的反相放大输出端，通常只要在 XTAL1 和 XTAL2 上并联一颗石英振荡晶体就可让系统动起来，若要让振荡的脉冲更稳定时，Intel 原厂建议在上述两脚与地端间各加入一颗 20～40pF 的小电容即可，石英振荡晶体应尽量靠近 XTAL1 与 XTAL2 脚。

PORT0

端口 0 是一个 8 位的 Open Drain 双向输入/出端口，当作输出功能时，每根脚都能驱动 8 个 LS 的 TTL 的输入端，若把状态 1 送到 PORT0 时，PORT0 则变成高阻抗的输入端口，当 8051 的 $\overline{EA}$ 等于低电位状态时，PORT0 则分别利用多任务的方式送出低位地址和数据值，此时 8051 会利用内部的上拉电路来增加 PORT0 的驱动能力，依照原厂数据的说法约可驱动 8 个 LS 的 TTL 元件，其驱动能力是相当强的。基本上，PORT0 是一个多功能的双向端口。

PORT1

端口 1 是一个双向的 I/O 端口，并且包括内部的上拉电路，此端口可以驱动 4 个标准 LS 的 TTL 元件输入端。只要将 PORT1 的输出状态设成 1 后，PORT1 就可以当成一个输入端口。当芯片是 8032 或 8052 时，PORT1 的 P1.0 和 P1.1 另外有 T2 和 T2EX 的功能。

PORT2

端口 2 是一个有内部上拉电路的双向 I/O 端口，可以驱动 4 个 LS 的 TTL 输入端，当把端口 2 的输出设成 1 时，此端口就能当成输入端口来用，若某个输入点被外部的电路拉成低电位状态时，会有一个小电流经上拉电路流向输入端，通常此电流都小于 1mA。当程序读取外部程序码和执行 MOVX @DPTR 指令时，PORT2 端将送出高地址值和 DPTR 中的 DPH 值。若是输出 1 时另外借用内部增强的上拉电路以得到较佳的输出值。

PORT3

端口 3 和其他 3 个端口相同，都是内部有上拉电路的双向 I/O 端口，也可以驱动 4 个 LS 的 TTL 输入端，当输出端被设成 1 后就可以当成输入端口来用。端口 3 除了标准的双向 I/O 功能外，另外每根引脚还有以下特殊的功能：

P3.0：RxD 串行通信输入端口。

P3.1：TxD 串行通信输出端口。

P3.2：INT0 外部中断输入点。

P3.3：INT1 外部中断输入点。

P3.4：T0 定时/计数器输入点。

P3.5：T1 定时/计数器输入点。

P3.6：$\overline{WR}$ 外部数据存储器的写入信号。

P3.7：$\overline{RD}$ 外部数据存储器的读取信号。

以上各功能涵盖到串行通信、外部硬件中断、定时/计数及外部数据存储器内容的存取等范围，这些实际应用技巧将在稍后各章的“彻底研究”文中做详细的说明。

由上面各个端口的说明我们可以了解到，当初 Intel 在设计 8051 系列单片机时是费了相当的苦心，由于引脚数限制在 40，所以除了某些控制脚的操作无法更改外，其他 4 个扩展端口都尽量设计成双向且多功能的，以满足只要单一芯片就能工作的要求，这些功能基本上都能满足我们电路上的需求，但是只要功能及线路确定后，还是会有以下的限制条件产生。

（1）若做串行通信时，P3.0 和 P3.1 就不能作为其他用途。

（2）若使用外部程序存储器时，P0 和 P2 端口将被用来送出地址值和接送数据值，所以原先的双向 I/O 端口功能也就没了。

（3）若使外部数据存储器时，一定会伴随写入（$\overline{WR}$）和读取（$\overline{RD}$）信号，所以将占用 P3.6 和 P3.7 引脚。

（4）若系统接受外部的硬件中断信号时，也会占用 P3.2 和 P3.3 引脚。

（5）若系统利用外部的定时/计数输入信号时，将会占用 P3.4 和 P3.5 引脚。

如果这些限制都同时存在，8051 系列的单片机将只剩 P1 端口可以做双向的 I/O 端口操作，只比传统的 8 位 CPU 如 Z80、6502 或 8085 好一点点而已，这意味着单片机在功能上较传统的 8 位 CPU 优越许多，当元件数目少时，单片机的效能最佳，最近几年，我们可以在市场上看到与 8051 功能兼容的单片机，且内部的程序空间加大到 32KB 或 64KB 时，系统已经不需要外挂 EPROM 充当程序存储器，单片机 CPU 在 8 位的市场上已具备必胜的筹码。

7-2 认识 AT89C2051

AT89C2051 是一枚 8051 兼容的单片机微控制器，它是 ATMEL 公司继 AT89C51 与 AT89C52 之后另一项非常成功的产品，虽然只有 20 根脚但仍保有 8051 大部分的功能。以下是原厂所提供的特性说明：

（1）与 Intel 的 MCS-51 完全兼容。

（2）内置 2KB 的可编程 Flash 存储器，允许 1000 次的清除与写入，程序数据可保存 10 年以上。

（3）工作电压可从 2.7~6V。

（4）工作频率可以由 0Hz~24MHz，这表示 AT89C2051 执行速度可以快也可以慢。

（5）有两层的程序区 Lock 功能，能保护程序资料不被他人剽窃。

（6）内部有 128 字节的数据存储器空间，与 8051 完全相同。

（7）15 个可编程 I/O 点，分别是 P1 端口与 P3 端口（少了 P3.6）。

（8）两个 16 位的定时/计数器。

（9）5 个中断源，这也跟 8051 相同。

（10）可编程串行通信端口（UART）。

（11）可直接驱动 LED。

（12）内有模拟比较器（Analog Comparator），这项功能为 ATMEL 特有的。

（13）有低耗电（Low Power Idle）与省电（Power Down）功能。

AT89C2051 仅有 20 根引脚，它是属于 CMOS 级的 8 位微控制器，内部有 2KB 的 Flash 存储器（Flash Programmable and Erasable Read Only Memory 简称 PEROM）。它特别适用于嵌

入式（Embedded）的控制应用。图 7-3 是 AT89C2051 的引脚图，图 7-4 则是 AT89C2051 的内部框图，请注意其 P1.0 与 P1.1 还接到比较器的输入点，比较器的输出刚好安排为 P3.6 位。接着来看看 AT89C2051 的引脚说明：

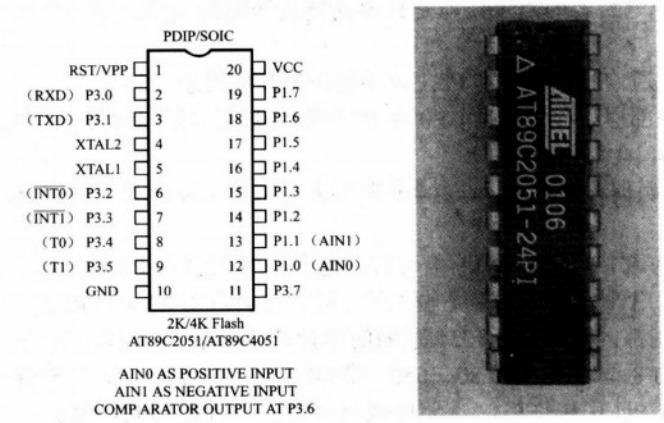


图 7-3　AT89C2051 的引脚图

（1）Vcc：正电源输入端。

（2）GND：电源接地端。

（3）P1 端口：P1 是一个双向的 I/O 端口，P1.2～P1.7 共 6 根引脚有内部的上拉电阻。P1.0 另外为比较器的正相输入端，P1.1 为比较器的反相输入端，当 P1.0 的电压比 P1.1 高时，比较器会输出 1，即内部 P3.6 位会成为 1，反之，P1.1 的电压比 P1.0 高时，P3.6 位会变成 0。P1 端口当输出 0 时，可以吸入（Sink）的电流可达 20mA，所以直接驱动 LED 是没有问题的。当我们把 P1 端口都输出成 1 时，就可以把 P1 端口当成输入端口来用，当其中某一个位被外部电路拉低到 0 时，会有一小电流（<1mA）经由上拉电阻流到地端，ATMEL 在资料上会提到此点的主要原因是，若系统由电池供电，就要把输入状态尽量维持在 Hi 高电位，以免耗掉过多的电量。

（4）P3 端口：P3 也是一个双向的 I/O 接口，并且包括内部的上拉电路。P3 端口当成输入时，也要考虑到经由上拉电阻的小电流，以免耗去太多的能量。P3 也有能力直接驱动 LED，除此之外，P3 端口还有以下特殊的功能：

1）P3.0　RxD（串行输入端口）。

2）P3.1　TxD（串行输出端口）。

3）P3.2　$\overline{\text{INT0}}$（外部中断 0）。

4）P3.3　$\overline{\text{INT1}}$（外部中断 1）。

5）P3.4　T0（外部定时/计数输入点）。

6）P3.5　T1（外部定时/计数输入点）。

（5）RST：系统的 RESET 输入引脚，当 RST 是 1 时，会强迫所有的 I/O 输出都变成 1，系统 RESET 的时间要持续两个机器周期以上才有效。

（6）XTAL1：石英晶体的输入端，它接到 AT89C2051 内部的反向振荡放大器的输入端。

（7）XTAL2：AT89C2051 内部的反向振荡放大器的输出，图 7-5 是 ATMEL 建议的石英

振荡器连接电路，若要直接由外部输入脉冲时，可参考图 7-6 的接法。

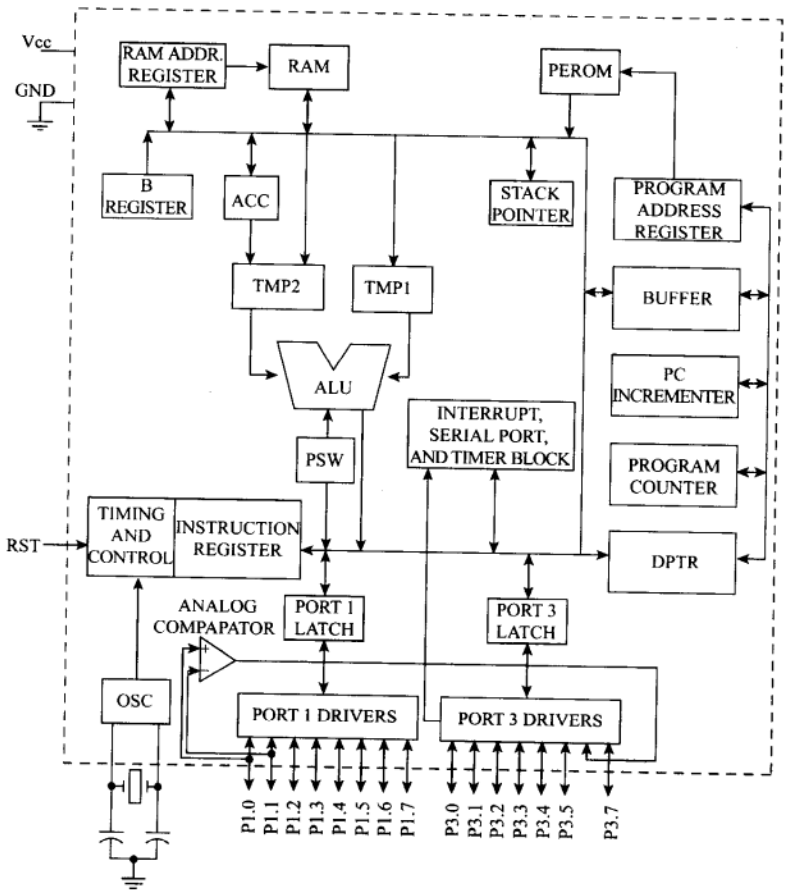


图 7-4　AT89C2051 的内部框图

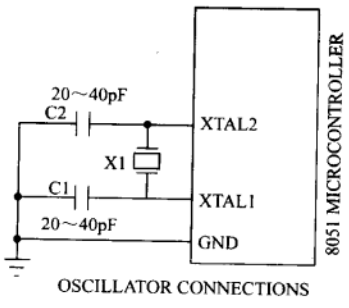


图 7-5　石英振荡器连接电路

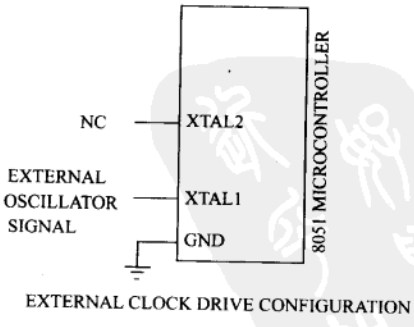


图 7-6　外接振荡信号的接法

AT89C2051 买来的时候，其内部的程序内容完全是空白的，也就是全部为 1 的 FFH，必须要用专用的 Flash 烧录器把程序码烧录进到 AT89C2051 内部。图 7-7 是将 2KB 程序数据烧

录进入 AT89C2051 的示意图，更详细的烧录 DIY 程序请参考《8051 单片机彻底研究——经验篇》中的说明。如果 AT89C2051 烧录并执行后，发现操作不对时，可以把该 IC 放到专用烧录器执行清除操作，只要执行清除命令 10 ms 后，就可以把所有的内容又变成 FFH，我们可以烧录更新的程序进行再次的验证。

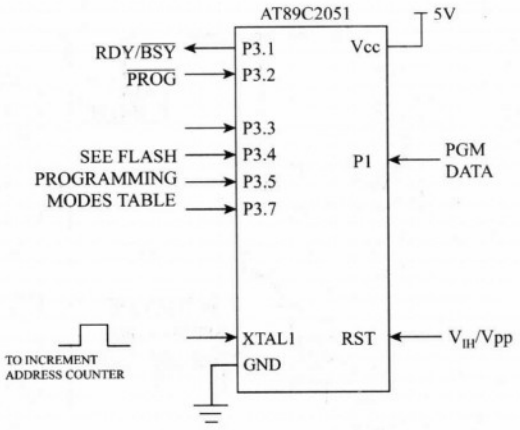


图 7-7　AT89C2051 的烧录示意图

7-3　8051 与 AT89C2051 的差异

复杂 8051 单片机系统除了有串行通信与各式各样的中断外，其程序空间可达 64KB，数据空间也可以到达完整的 64KB。可是如果我们要的只是几个 LED ON/OFF 的操作时，采用 AT89C2051 单片机会最省事也最方便，因为只要接上石英晶体与 RESET 用的 1μF 电容，再写个小汇编语言程序就能让 LED 闪烁得很漂亮了，不过，AT89C2051 还是跟 8051 单片机有点小差异：

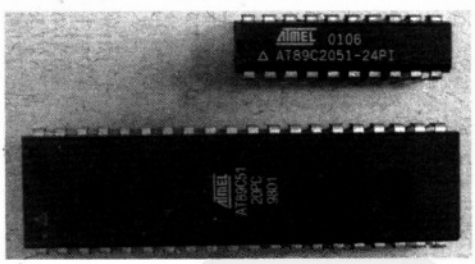


图 7-8　AT89C51 与 AT89C2051 的实物图

（1）无法再做程序空间的扩充，2048 字节的程序空间就是 AT89C2051 的极限。

（2）不能使用 MOVX 的指令。

（3）没有 $\overline{WR}$ 与 $\overline{RD}$ 信号。

（4）当程序数据被 Lock 后，就无法从烧录器上读回，但可以全部清除再烧录新数据。

（5）AT89C2051 只要 3V 就能工作，两个电池串联就可以使该 IC 动起来。

7-4　AT89C 系列的下一步

ATMEL 公司在推出 AT89C51 与 AT89C52 深获好评后，接下来的 AT89C2051 也是相当成功的，软件工程师特别关心程序空间的大小，空间当然越大越好，可是价格也会因空间增加而增加，但绝对不是加倍的，这也就是说，AT89C52 的程序容量是 AT89C51 的两

倍，但前者的价格不会是后者的两倍。ATMEL 在这些 Flash 单片机流行之后，陆续推出了数种不同用途的 8051 兼容单片机，程序空间更大，额外的附加功能更多，唯一相同的是全部采 FLASH 作为其程序存储器空间。更详细的 IC 资料情形请到 http://www.atmel.com 网站上查询。

AT89C55：40 引脚，程序空间 20KB。

AT89S8252：40 引脚，程序空间 8KB，内部还有 2KB 的 E^2PROM，可以存放系统参数或重要设定值。

AT89C4051：20 引脚，程序空间 4KB。

AT89C1051U：20 引脚，程序空间 1KB，原先的 AT89C1051 是没有串行接口，但是后来又加入 UART，所以编号最后又加上一个 U 的代号。

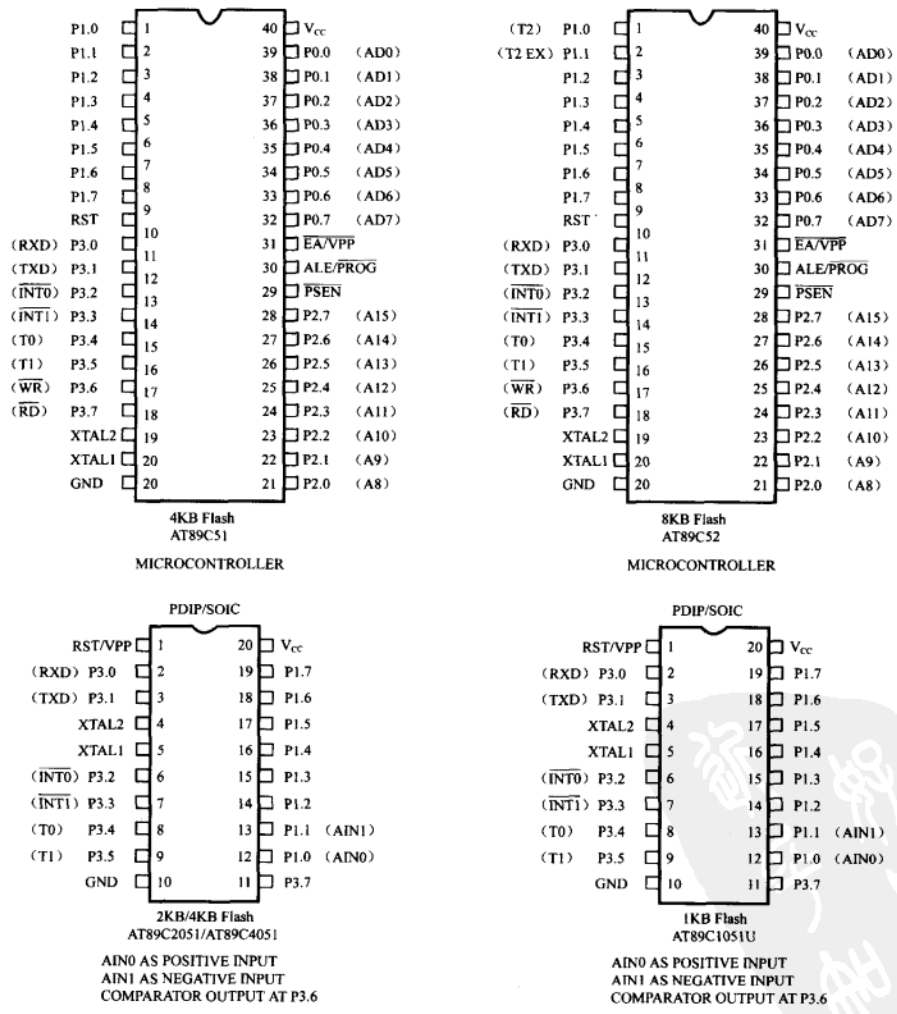
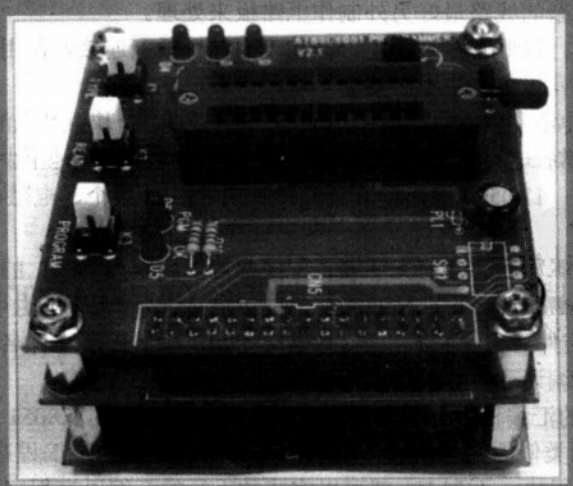


图 7-9　ATMEL 常用 8051 微控制器的引脚图

习　题

1. 试绘出 AT89C2051 的引脚图。
2. RST 引脚有什么用途？其操作原理是什么？
3. XTAL 引脚有什么功能？输入/输出的引脚有哪些？
4. 在 AT89C51 中，PORT3 的各引脚名称叫什么？
5. 承上题，在应用 PORT3 的各引脚时有哪些限制？
6. AT89C2051 与 AT89C51 的引脚有何不同？功能上有哪些差异？
7. AT89C2051 的 P3.6 有何用途？其脚位是什么？
8. AT89C55 的程序空间容量有多大？

8



USB 烧录器烧录一颗 AT89C2051 的时间可在 10s 内完成，这个速度与 ICE 下载的速度相当

第 8 章　8051 基本程序练习

在前面的章节里，我们学习了 8051 单片机的基本介绍及各种指令的用法，你是否跃跃欲试，想立即看到自己所写的程序运行的情况呢？本章设计一些简单的程序，只要一个 AT89C2051、一块“面包”板、几个简单的电子元件，就可以马上看到你学习的成果，让你对 8051 单片机的应用有更进一步的体验。

8-1　工具的准备

编写 8051 的汇编语言时，除了一台 PC 机外，应该有哪些配备呢？下面我们把这些必备的工具分为软硬件两个方面来讨论。

1. 硬件方面

（1）PC 机：Windows 或 DOS 的操作系统均可，这台 PC 要能执行 8051 汇编程序，而且还要能与 AT89C2051 烧录器连通，传送烧录内容后进行烧录相关的所有操作。

（2）AT89C2051 烧录器：万用烧录器或任何一款可烧录 AT89C2051 的烧录器均可，本书所有的例子都以 AT89C2051 为主。

（3）AT89C2051 单片机：最好准备两块或两块以上，可以交替烧录，减少等待的时间。

（4）“面包”板：不用焊接电路与元件，直接用单芯线连接，我们建议简单的电路可以用此方法验证，但是复杂的线路最好另外制作电路板来处理。

（5）单芯线：少许，最好有多种颜色，插在“面包”板上连接元件用，系统石英振荡器的部分线路应尽量缩短，以免产生不必要的干扰。

（6）电源供应器或电池：以能输出+5V 为原则，也可以用 3 块碱性电池串成 4.5V 供电，AT89C2051 本身就是非常省电的 IC，除非是做 LED 的显示，3 块碱性电池至少可使用半个月以上。

（7）其他电子元件：11.0592MHz 的石英晶体一块、LED 数个、电阻与电解电容数个。

2. 软件方面

（1）程序文件的编辑程序。任何文字编辑程序都可以用来写 8051 的汇编语言程序，例如 DOS 版下的 EDIT、Windows 下的 Notepad、WordPad 或 Word 都行，也可以加中英文注解，8051 汇编语言的扩展文件名为 ASM，只要存储时指定正确的扩展文件名后，就可以由汇编语言对你写的程序进行编译。

（2）单片机 8051 的汇编程序。在本书中我们是以 2500AD 的 X8051 做操作的示范，你也可以在因特网上找到类似的共享软件，使用大都相同，这类的汇编程序大都还是以 DOS 环境为主，编译的速度很快，不到 10s 就可以完成数千行的汇编语言程序。

（3）AT89C2051 烧录器的驱动程序。这个程序也有 DOS 与 Windows 版本之分，如果你的汇编程序是 DOS 版的，烧录器也是 DOS 版的，那么开机时就可以直接用 DOS 启动，以免系统进

入 Windows 后又返回 DOS，浪费许多等待的时间。若汇编程序是 DOS 版，但烧录器必须在 Windows 环境下工作时，那就要让 PC 先进入 Windows 后，再开一个 DOS 窗口来执行汇编程序。

（4）HEX2BIN 程序。通常汇编程序在链接时，可以指定输出的格式，最常见的是二进制的 TSK 文件和十六进制的 HEX 文件，前者是真正要烧录到 AT89C2051 的文件，如果你的程序长度有 100 字节，则 TSK 文件也是 100 字节长。TSK 文件的内容只有 8051 单片机才看得懂。HEX 文件的内容全部是 ASCII 的格式，可以用 Notepad 或 Word 就可以打开来看。有些 AT89C2051 烧录器只能接受 TSK 文件的格式，因为它收到后不需要再做任何转换就可以进行烧录，可是，如果你使用的汇编程序又只能产生 HEX 文件时，就需要 HEX2BIN.EXE 这个程序来做转换。你可以到旗威科技公司网站上下载 HEX2BIN 执行文件，它产生的 TSK 文件的长度会比较短，当你写的程序长度快要接近 AT89C2051 的 2048 字节程序长度极限时，就会发觉它的优点所在。

8-2 8051 汇编程序 X8051 与 LINK4 的操作

本节所要示范的汇编程序是由 2500AD 这家公司所开发的软件，它是目前在市面上所能看到较好用的 DOS 版汇编程序，假如你想要找视窗版的汇编程序，那么网络上所能下载的汇编程序都有程序长度与功能上的限制，必须向开发的公司购买取得序列号后才能完整使用。可惜的是 2500AD 这家公司几年前被并购（http://www.avocetsystems.com）后，市场上已找不到比 2500AD 更好的汇编语言程序编译程序。

使用该软件的步骤很简单，请确认 X8051.EXE、LINK4.EXE 与你写的汇编语言在同一个目录上，假设是 C:\2500AD>，请确认你所写的汇编语言程序扩展文件名为.ASM，在 DOS 模式进到 C:\2500AD 的路径下，执行 X8051 A（A.asm 为你所储存的文件名），此时文件夹会产生一个 A.obj 的文件，这样就好了吗？别急，在同样的路径下执行 LINK4，出现询问 FILE NAME 的信息，只要键入 A，指定要把该 OBJ 中间文件变成 8051 可执行的机器码，然后其他的信息都按 Enter 键跳过，直到询问文件类型要转成何种类型时：Options（D，S，A，M，Z，X，H，E，T，1，2，3，<CR>=Default），键入 X，这时在文件夹里会出现一个 A.TSK

```
C:\2500AD>X8051A
                8051 Macro Assembler-Version 4.05b
        Copyringht(C) 1985 by 2500 A D.Software, Inc

              *****Active Comands*****
                Ctrl S=Stop Output
                Ctrl Q=Stop Output
                Esc C=Stop Assembly
                Esc T=Terminal Output
                Esc P=Printer Output
                Esc D=Disk Output
                Esc M=Multiple Output
                Esc N=No Output

        2500 A.D. 8051 Macro Asssembler-Version 4.05b
        ---------------------------------------------
                Input Filename: A.asm
                Output Filename: A.obj

        Lines Assembled: 17          Assembly Errors: 0
```

图 8-1 X8051 编译的画面

```
C:\2500AD>LINK4

2500 A.D. Linker Copyright (C) 1985-Version 4.05b

Input Filename: A
      Enter Offest For 'CODE'
Input Filename:

Output Filename:

Library Filename:
Options (D,P,S,A,M,N,Z,X,H,E,T,1,2,3,<CR>=Default): X

                Linder Output Filename: A.tsk
                Disk Listing   Filename: <None Specified>
                Symbol Table Filename: <None Specified>

Link Errors: 0          Output Format: Executable
```

图 8-2　LINK4 链接的画面

二进制的文件，把这个文件烧录到 AT89C2051 芯片里就行了。

本书的最后一章上有旗威科技公司开发的 Windows 版 AT89C2051 烧录器的操作示范说明，该烧录器是采用 USB 接口与 PC 连接，所有电量供应来自 USB，不需要任何外加电源即可进行烧录，也可以单机自行烧录并且加上 Lock bit 保护。

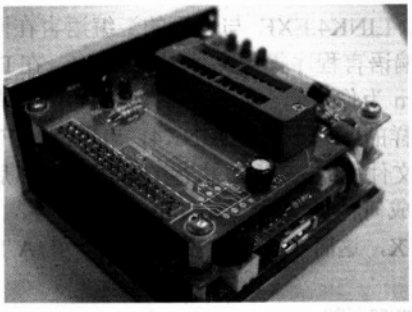


图 8-3　旗威科技公司的 USB 接口烧录器

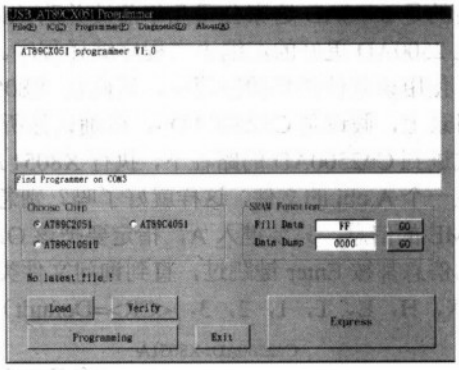


图 8-4　烧录的画面

8-3　基础范例一：LED 的亮与灭

目的：熟悉 AT89C2051 的引脚并学习 SETB、CLR 的用法。

硬件要求

- AT89C2051 烧录器。
- AT89C2051。
- “面包”板。
- 1μF/16V 及 10μF/25V 电解电容、11.0592MHz 石英晶振、LED、20pF 电容×2。

操作说明

SETB 使引脚输出电位为 High，CLR 使引脚输出电位为 Low。

线路接法

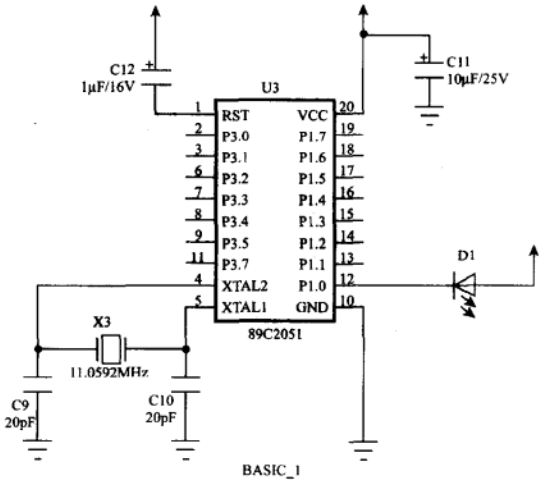


图 8-5　让 LED 明灭的线路图，所采用的是负逻辑（Negative Logic）接法

程序范例：CH8_LED.ASM

```
LED     REG     P1.0        ;设置 LED 所接的脚位
;
        ORG     0000H
START   MOV     R1,#00H     ;在系统开始前先做一小段延迟
$       DJNZ    R1,$        ;让系统的运行先就绪
        MOV     SP,#60H     ;声明 stack
;
        CLR     LED         ;LED on,此时 LED 正极接电源,负极接 P1.0
        CALL    DELAY       ;做一个段延迟
        SETB    LED         ;LED off
        CALL    DELAY       ;做一小段延迟
;
DELAY   MOV     R0,#00H
$1      MOV     R1,#00H
$2      DJNZ    R1,$2       ;内循环
        DJNZ    R0,$1       ;外循环
        RET
```

程序说明

一个正确的 8051 程序，在系统一开机 RESET 后，一定会加入一小段延迟的时间，让系统的硬件先就绪后，才正式开始处理主程序所要进行的操作，由于 P1.0 输出端是接 LED，而每一次 LED 状态改变的时候也要加入适当的延迟时间，让它有充分的时间可以把状态持续，否则 LED 亮灭的速度太快，肉眼是无法察觉它在变化的操作。

讨论：

如果将 LED 的正极接 P1.0，而负极接地时，LED 会产生怎样的变化呢？除了明灭的顺序颠倒之外，亮度上也会变暗许多，这是因为由 AT89C2051 的 P1.0 输出是 1 时所送出来的电流较小（<0.5mA），LED 当然就亮不起来，但 P1.0 输出 0 时它可以吸入多达 20mA 的电流，而 LED 要点到很亮时只要 10mA 的电流就够了。所以，正确的连接方法应该像电路图一样接

成负逻辑（Negative Logic）的状态，CLR LED 时 LED 点亮，SETB LED 时 LED 熄灭，刚好跟我们的习惯相反。

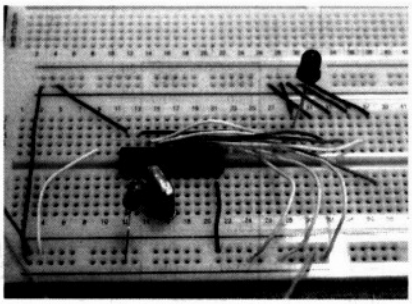


图 8-6 “面包”板接上 LED 明灭线路

注：石英晶振的引脚应离 AT89C2051 的 XTAL1、XTAL2 引脚越近越好

8-4 基础范例二：蜂鸣器的使用

目的：学习 MOV direct,#data 指令的用法。

硬件要求:

- AT89C2051 烧录器。
- AT89C2051。
- “面包”板。
- 规格 5V 的蜂鸣器（BUZZER，内置振荡器型）。
- 其他元件同范例一（LED 不使用）。

线路接法

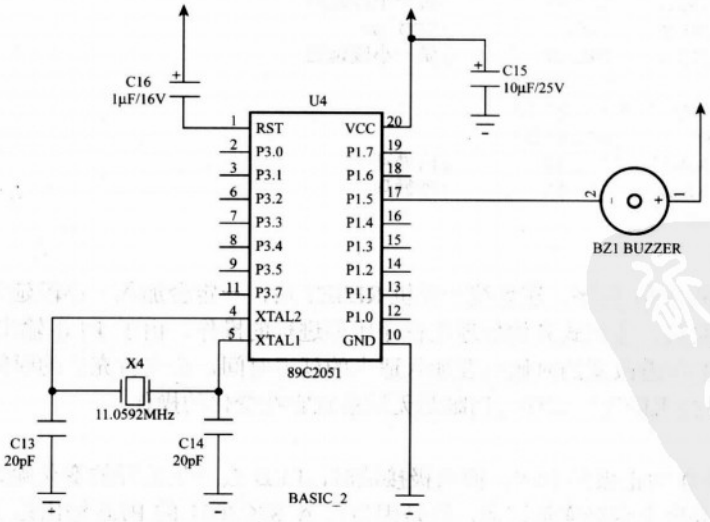


图 8-7 蜂鸣器鸣叫的线路图

注：使用负逻辑接法可使鸣叫声较宏亮

操作说明

当 P1.0 被设置成 1 时，引脚输出状态为 High，P1.0 被设成 0 时，引脚输出变为 Low。此时 BUZZER 的正端应该接电源正端，负端接 P1.0 端口。

程序范例：CH8_BUZZER.ASM

```
        ORG     0000H
START   MOV     R1,#00H     ;在系统开始前先做一小段延迟
$       DJNZ    R1,$        ;让系统稳定后
        MOV     SP,#60H     ;声明 stack
;
        MOV     P1,#FFH     ;BUZZER off
        CALL    DELAY       ;做一小段延迟
        MOV     P1,#00H     ;BUZZER on
        CALL    DELAY       ;做一小段延迟
;
DELAY   MOV     R0,#00H
$1      MOV     R1,#00H
$2      DJNZ    R1,$2
        DJNZ    R0,$1
    RET
```

程序说明：

本程序所要学习的要点在于如何输入有用的 DATA，使设备能够驱动，在此范例中所有 P1.0 端口的任一个位接 BUZZER 都会产生哔哔的间断鸣叫声。

讨论：

如果要让 BUZZER 的鸣叫声持续较久的时间，程序应该如何修改呢？是增加延迟的时间，还是有更好的做法？又如果要让 BUZZER 的鸣叫声的间隔时间加长，除了加长关闭的时间延迟外，是不是有更好的做法？先想想看，在稍后章节里会有更深入的探讨。

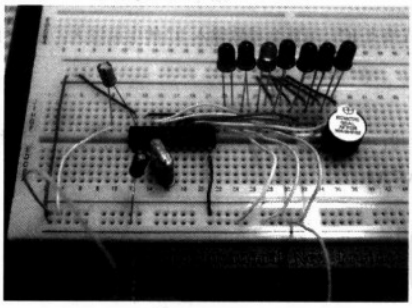


图 8-8 “面包”板接上蜂鸣器鸣叫线路实物图

注：搭配 LED 与蜂鸣器可做多种信号的变化

8-5 基础范例三：指示灯

目的：了解 ACC 累加器的用途及学习 RR、RL 的用法。

硬件要求

- AT89C2051 烧录器。
- AT89C2051。
- “面包”板。
- 8 个 LED。
- 其他元件同范例一。

线路接法

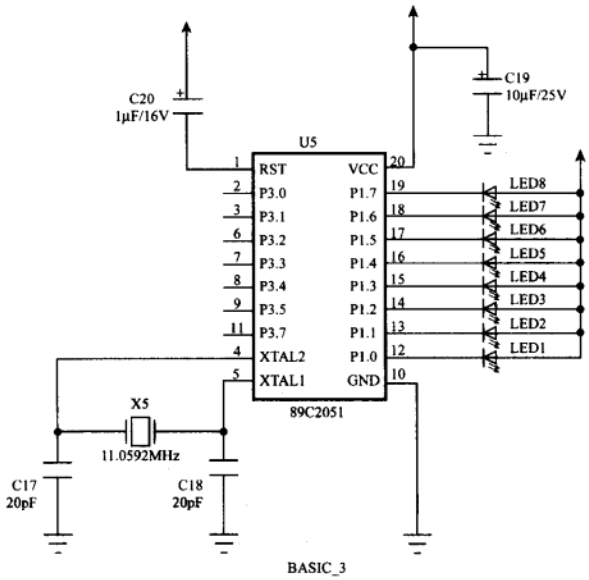


图 8-9　指示灯线路图

注：多个 LED 同时以负逻辑的方式接上 IC，可能会导致输入 IC 的电流过大，建议可在 LED 上串联一个 470~1000Ω 的电阻作限流的处理

操作说明：

8 个 LED 会来回地轮流明灭，LED 的正端要接电源，负端接到 P1.0 端口上。

程序范例：CH8_PILI.ASM

```
        ORG     0000H
START   MOV     R1,#00H
$1      DJNZ    R1,$1
        MOV     SP,#50H
LOOP    MOV     A,#FEH              ;累加器存入'1111 1110B'的 DATA
LOOP_1  MOV     P1,A                ;将累加器的数据送给 P1 端口
        CALL    DELAY               ;延迟一小段时间
        RL      A                   ;将'1'向左移位=>'1111  1101B'
        CJNE    A,#7FH, LOOP_1      ;当 DATA 值不等于 7FH 时回 LOOP_1
LOOP_2  MOV     P1,A
        CALL    DELAY
```

```
        RR      A                       ;向右移位
        CJNE    A,#FEH, LOOP_2
        SJMP    LOOP                    ;跳转回到 LOOP 重新执行
;
DELAY   MOV     R0,#00H
$1      MOV     R1,#00H
$2      DJNZ    R1,$2
        DJNZ    R0,$1
        RET
```

程序说明：

当 RL A 的指令运算时，会将放在 ACC 累加器中的值向左移动 1 位。举例来说，如果 ACC 累加器中的 DATA 为 00010110B（16H），那么 RLA 后的 DATA 应该为 00101100B（2CH），相当于数学运算的乘 2，而 RRA 则为 00001011B（0BH），相当于数学运算的除 2。要注意的是这两个运算指令只适用于 ACC 累加器。

讨论：

在单片机的世界里，有许多处理的方式是和人的习惯有所不同的，在 8051 汇编语言里就可以看出其中的差异，8051 的汇编语言程序中，要叫它做乘 2 或除 2 都是很容易的，只要执行 RL A 或 RR A 就行了。但是要把某个值乘 3 时，最快的方法是先乘 2 再加上自己，就变成乘 3 了。

图 8-10 是用 8 个 LED 及 AT89C2051 组成的指示灯实物图。

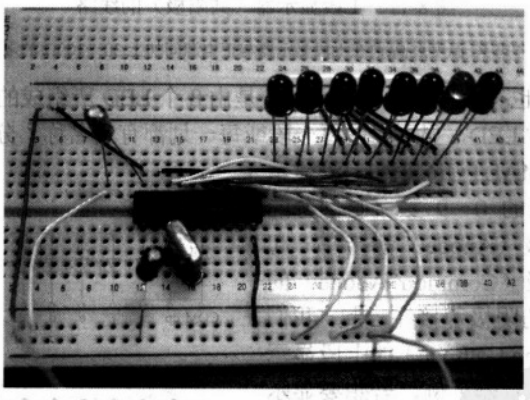


图 8-10　8 个 LED 及 AT89C2051 组成的指示灯

8-6　基础范例四：七段显示器的使用

目的：了解 8051 如何驱动七段显示器。

硬件要求

- AT89C2051 烧录器。
- AT89C2051。
- “面包”板。

- 1 块 0.5 共阳极的七段显示器（简称单 8）。
- 其他元件同范例一。

线路接法

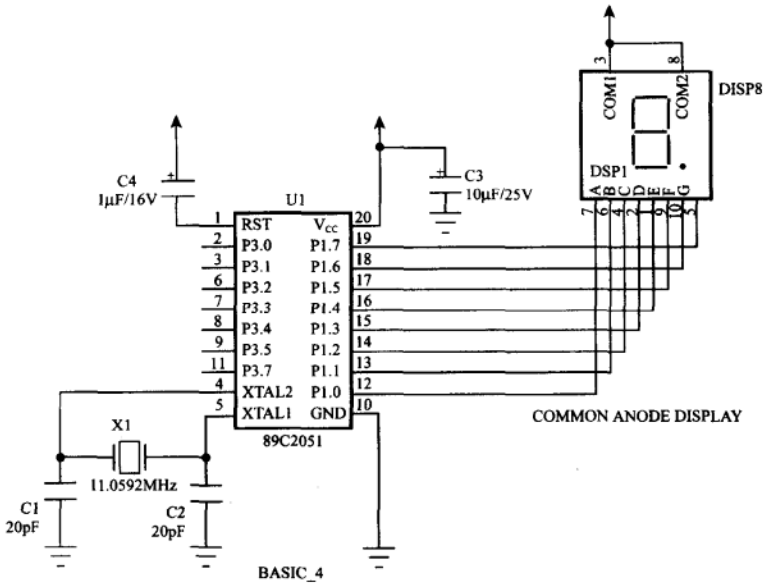


图 8-11　七段显示器（共阳极）的线路

注：如果考虑 IC 温度过高,请在七段显示器与正电源间加上 30～100Ω 的限流电阻。

每个七段显示器（也有人称为七节显示屏）是由 8 个 LED 所组成的，其中包括 7 个细长条形的 LED 及小数点形的 LED，显示器的每一段或每一划都有其名称，分别是英文小写的 a～f，以及小数点 dp（DECIMAL POINT）。七段显示器可以显示包括小数点的 0～9 数字与部分的英文字母。如果其阳极（Anode）都接在一起时，我们称之为共阳极（Common Anode）的七段显示器，若 LED 的阴极都接在一起，我们称之为共阴极（Common Cathode）的七段显示器。以共阳极的七段显示器为例，若想要显示数字 1 时，就要使 b 划与 c 划点亮，在电路上我们会把共同点接到正电源端（+5V），标示 b 与 c 的地方经过限流电阻到地，就可以显示数字 1。在本范例中，七段显示器是直接点亮的，亦即用 8 位去驱动一个七段显示器。如果有多位数字要显示时，可以用扫描的方式显示，以节省控制脚位。

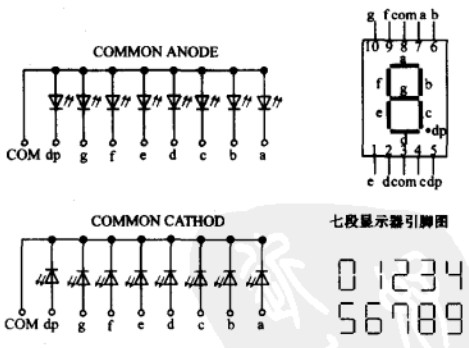


图 8-12　七段显示器内部线路及的引脚图

操作说明

显示数值由 0 变化到 9。

程序范例：CH8_DISP_8.ASM

```
        ORG     0000H
START   MOV     R1,#00H
$1      DJNZ    R1,$1
        MOV     SP,#50H
;
LOOP    MOV     A,#3FH      ;累加器存入'00111111B'的 Data,显示值为数字 0
        CPL     A           ;取反
        MOV     P1,A        ;将累加器的数据送给 P1 端口
        CALL    DELAY

        MOV     A,#06H      ;累加器存入'00000110B'的 Data,显示值为数字 1
        CPL     A
        MOV     P1,A
        CALL    DELAY

        MOV     A,#5BH      ;累加器存入'01011011B'的 Data,显示值为数字 2
        CPL     A
        MOV     P1,A
        CALL    DELAY

        MOV     A,#4FH      ;累加器存入'01001111B'的 Data,显示值为数字 3
        CPL     A
        MOV     P1,A
        CALL    DELAY

        MOV     A,#66H      ;累加器存入'01100110B'的 Data,显示值为数字 4
        CPL     A
        MOV     P1,A
        CALL    DELAY

        MOV     A,#6DH      ;累加器存入'01101101B'的 Data,显示值为数字 5
        CPL     A
        MOV     P1,A
        CALL    DELAY

        MOV     A,#7DH      ;累加器存入'01111101B'的 Data,显示值为数字 6
        CPL     A
        MOV     P1,A
        CALL    DELAY

        MOV     A,#27H      ;累加器存入'00100111B'的 Data, 显示值为数字 7
        CPL     A
        MOV     P1,A
        CALL    DELAY

        MOV     A,#7FH      ;累加器存入'01111111B'的 Data,显示值为数字 8
        CPL     A
        MOV     P1,A
```

```
        CALL    DELAY
        MOV     A,#6FH          ;累加器存入'01101111B'的 Data,显示值为数字 9
        CPL     A
        MOV     P1,A
        CALL    DELAY
        SJMP    LOOP            ;跳转回到 LOOP 重新执行
;
DELAY   MOV     R0,#00H
$1      MOV     R1,#00H
$2      DJNZ    R1,$2
        MOV     R2,#00H
$3      DJNZ    R2,$3
        DJNZ    R0,$1
        RET
```

程序说明

这个程序分别把 0～9 共 10 个位数的显示码送给七段显示器，再配合适当的时间延迟，就可以看到数字的显示了。另外我们也可以用 MOVC 查表指令，来做相同的工作，而且程序长度也较短。

CH8_DISP8-2.ASM

```
CNT     EQU     30H
        ORG     0000H
        MOV     P1,#BFH
START   MOV     R1,#00H
$       DJNZ    R1,$
        MOV     SP,#50H
;
        MOV     CNT,#00H        ;计数值先清为 0
LOOP    MOV     A,CNT           ;读取计数值至累加器中
        MOV     DPTR,#TABLE     ;定义字形表的起始地址
        MOVC    A,@A+DPTR       ;查询计数值所对应的字形码
        CALL    OUTPUT
        INC     CNT             ;计数值加 1
        MOV     A,CNT           ;将计数值存到累加器中
        CJNG    A,#10,LOOP      ;判断累加器中计数值等于 10 则离开循环
        MOV     CNT,#00H        ;将计数值填成 0
        SJMP    LOOP
;
;7 SEGMENT DISPLAY TABLE
TABLE   DB      3FH             ;0
        DB      06H             ;1
        DB      5BH             ;2
        DB      4FH             ;3
        DB      66H             ;4
        DB      6DH             ;5
        DB      7DH             ;6
        DB      27H             ;7
        DB      7FH             ;8
        DB      6FH             ;9
OUTPUT
```

```
        CPL     A                   ;输出时因为电路设计的缘故,所以要做取反处理
        MOV     P1,A
        CALL    DELAY
        RET
;
DELAY   MOV     R0,#00H
$1      MOV     R1,#00H
$       DJNZ    R1,$
        DJNZ    R0,$1
        RET
```

程序说明

这个程序的运行也是很简单的，它先由 CNT 位置取得一个 0～9 的计数值，然后用 MOV C 指令去拿到真正的七段显示码，再做一次取反后才到 P1 端口上，就可以做正确的数字显示，第二个写法才是比较好的程序写法，我们还可以依实际的需要加上 A～F 的非数字显示，而程序需要修改的部分只有查表这里而已，在 8051 真正的应用当中，我们经常会使用到 MOVC 指令来做查表的操作，这也是这个程序实现所要传达的主要概念。

讨论

在 8051 的应用邻域上，除了 LED 显示外，就属七段显示器的应用最常见，它相当于 8 个 LED 的总合，虽然体积比较大，但是能够显示的信息远较 LED 来得多，许多仪器内部都有类似的电路，不过，这不是给用户观看的，而是当机器出问题时，维修工程师看七段显示器上显示的代码，就可以了解故障的原因。

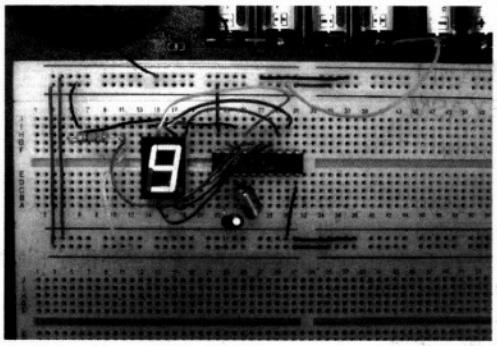


图 8-13　接上七段显示器线路的“面包”板

注：七段显示器在通电后会从 0～9 反复显示。

8-7　基础范例五：按键的使用

目的：了解机械按键的特性与用法。

硬件要求

- AT89C2051 烧录器。
- AT89C2051。
- “面包”板。

- 1 块 0.5 共阳极的七段显示器（简称单 8）。
- 1 块机械式的按键、1 个 10kΩ电阻。
- 其他元件同范例一。

线路接法

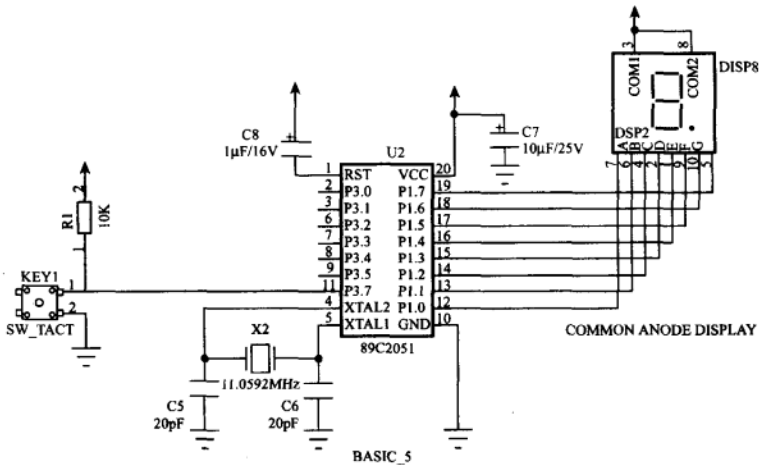


图 8-14　测试按键功能的线路图

注：如果顾虑 IC 温度过高，请在七段显示器与正电源间加上 30～100Ω的限流电阻。

操作说明

每按一次键，显示数值加 1。

程序范例：CH8_KEY.ASM

```
        ORG     0000H
;
        MOV     R2,#00H         ;设置 R2 为计数寄存区并清为 0
LOOP    MOV     P3,#FFH         ;设置 P3 为输入端口
$       JB      P3.7,$          ;当按键输入时,离开循环执行下面的程序
        MOV     DPTR,#TABLE
        MOV     A,R2
        MOVC    A,@A+DPTR
;
        MOV     P1,A
        CALL    DELAY
;
        MOV     A,R2
        ADD     A,#1
        DA      A
        ANL     A,#0FH
        MOV     R2,A
        JMP     LOOP
;
DELAY   MOV     R0,#00H
$1      MOV     R1,#30H
$2      DJNZ    R1,$2
        DJNZ    R0,$1
```

```
        RET
;
TABLE
        DB      C0H             ;0
        DB      F9H             ;1
        DB      A4H             ;2
        DB      B0H             ;3
        DB      99H             ;4
        DB      92H             ;5
        DB      82H             ;6
        DB      D8H             ;7
        DB      80H             ;8
        DB      90H             ;9
```

程序范例：CH8_KEY-2.ASM

```
CNT     EQU     30H             ;定义计数器的地址
KEY     REG     P3.7            ;定义按键的输入脚
;
        ORG     0000H
        MOV     P1,#BFH         ;设置通电后起始状态
        MOV     P3,#FFH
START   MOV     R1,#00H
$       DJNZ    R1,$
        MOV     SP,#50H
;
        MOV     CNT,#00H
        SETB    KEY
WAIT    JB      KEY,WAIT
;KEY PRESS
        MOV     A,CNT
        MOV     DPTR,#TABLE
        MOVC    A,@A+DPTR
        CALL    OUTPUT
        INC     CNT
        MOV     A,CNT
        CJNE    A,#10,NEXT
        MOV     CNT,#00H
NEXT    JNB     KEY,NEXT        ;WAIT UNTIL KEY RELEASE
        SJMP    WAIT
;
;7 SEGMENT DISPLAY TABLE
TABLE   DB      3FH             ;0
        DB      06H             ;1
        DB      5BH             ;2
        DB      4FH             ;3
        DB      66H             ;4
        DB      6DH             ;5
        DB      7DH             ;6
        DB      27H             ;7
        DB      7FH             ;8
        DB      6FH             ;9
;
OUTPUT  CPL     A
        MOV     P1,A
```

```
        CALL    DELAY
        RET
;
DELAY   MOV     R0,#00H
$1      MOV     R1,#00H
$       DJNZ    R1,$
        DJNZ    R0,$1
        RET
```

程序说明

这里我们共列了两个程序，你可以看出其中的差异吗?第一个例子只检查按键按下的操作，若按下的时间久一点时，显示的数字还会一直往上加。第二个程序除了检查按键被按下外，还多了一道检查，确定按键已经被放开，程序多了这道手续后，不论按键被压多久，显示值都只会改变数次而已。

讨论

这是程序处理机械式的开关输入经常会碰到的问题，更标准的做法是：

（1）判定按键是否被压下，不是的话就离开。

（2）至少延迟 10 ms 的时间。

（3）再做一次按键的判断，确定按键真的被压下时才执行操作。

（4）开始执行按键特定的操作。

（5）执行完后，若此时按键仍旧被按着，则等到按键被放开后才离开。

8051 对于按键的处理程序可以很轻易地带过，也可能需要相当慎重地处理，这都要看实际的应用需求而定，某些控制系统中的按键有些需要每按数次操作，有些按键则允许一直按着，直到特定值后才放开。

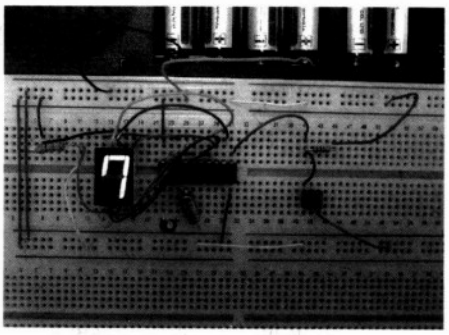


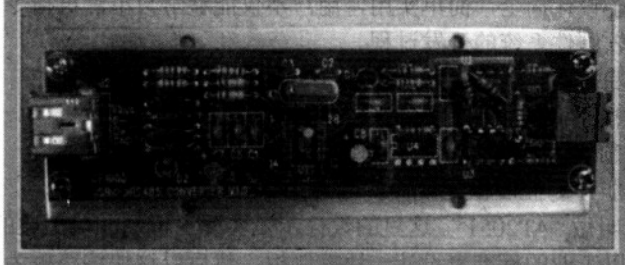
图 8-15　加上按键测试线路的“面包”板

注：按键 Tact Switch 在图的右侧。

习　题

1. 试编写一段小程序控制 LED 的明亮。
2. 试编写一段小程序使蜂鸣器每间隔一秒发出声响一次（写出间隔一秒的计算方式）。
3. 在基础范例中，何种接法可使 LED 的亮度提高？请说明原因。
4. 什么是 Negative Logic?
5. 试编写一段程序使指示灯以同时亮两个 LED 的方式来回运行。
6. 试绘制一个七段显示器的引脚图。
7. 试编写一个以七段显示器显示英文字母的程序（利用查表法）。
8. 试编写一程序使七段显示器的显示数字，每按键一次可加 3。

9



USB 转 RS485 接口板用 9600 b/s 的速度收送串行数据，这个速度对一般的仪器设备应该足够

第 9 章　8051 控制板线路说明

AT2051 控制板是一块标准的工业用控制板，它上面有温度感测 IC 与显示屏，也可以即时与 PC 机连接。我们早已成功地将此设备应用在房间温度调节监控与商场环境监控等诸多领域。

9-1　如何选用控制板

在第 8 章我们已经练习过几个 8051 的范例，然而要进一步做更实用、更完整的系统，“面包”板上的布线是不够用的。因此我们有必要建立一个良好的学习环境及平台，以便进行更深入的学习。我们首先要做的就是设计一块 8051 控制板，然后用这块控制板作为下面的学习平台。

许多 8051 学习板大都强调学习性高，为了把所有的学习项目都包含进来，间接地造成 PC 板面积的增大，学习 8051 一段时间后，该 8051 学习板就束之高阁了，对于学习 8051 的程序而言，这些似乎已经全然足够了，不过当你正式面对实际的设计时，往往还要经过一连串的软件与硬件考验，这样就说明了 8051 的学习是一回事，而实际应用又是另一回事。这是因为真正在实际应用时，要考虑的事情更多了，精确度更高了。这其中包含：

（1）产品的数量。
（2）生产产品时元件的质量控制。
（3）产品的规范与说明文件。
（4）硬件本身的安全性与保护性。
（5）防止高频电磁干扰（EMI）。
（6）程序更新与错误反馈。
（7）售后服务与产品维修。

因此，与其在学习板上做“模拟”的学习，不如脚踏实地的真正做一块“真实的”8051 控制板，只要这块板子设计周全，简单实用，用户不仅能够学习单片机指令、程序，并且还参与了解整个单片机软硬件系统的开发过程。

我们希望达到的目标有：

（1）开发一块价格不高的多用途控制板。
（2）这片板子必须实用且符合工业应用的要求。
（3）读者的程序能够和控制板配合而实际可用。

我们把这块板子叫做“AT2051”。这里的设计示范实际上已经涵盖上述的几个重要观念，其实，我们的构想是以工业的标准来设计一块控制板，然后以这块线路板为主题，逐一地把各种汇编语言程序加进来，这也包括与 PC 机的连接程序，最后以数十种实例来实现 8051 单片机的系统应用。这种安排让学习者全程观看整个软硬件的设计流程，配合各种仪器设备的

操作与确认，进而对 8051 单片机有更深刻的认识。

9-2 AT2051 控制板的特点

我们的 AT2051 控制板包含有：

（1）AT89C2051 单片机：Atmel 公司内置 2K Flash 程序空间的单片机。

（2）两位七段数字显示器。

（3）SMART 温度感测 IC。

（4）串行 EEPROM 24LC16。

（5）RS485 通信线路。

（6）按钮输入与蜂鸣音响输出。

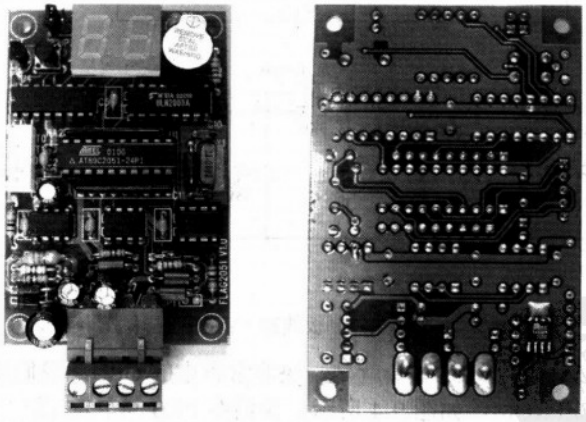
（7）供应电压范围宽广（DC 6～34V）。

（8）PC 板的面积仅有半个巴掌大（52×80mm）。

（9）分散式控制概念的导入。

在 AT2051 控制板上，我们必须要特别强调的是，分散式控制观念已经加到系统程序当中，当 AT2051 独自工作时，它是一个控制器，当它与 PC 机实时连通，它是一个简易的控制系统。当许多个 AT2051 控制板同时与微机并联通信时，它们就是如假包换的工业监控系统了。

在使用初期，AT2051 控制板可以拿来学习 8051 的汇编语言，你写好程序并且编译成二进制文件的格式，经过烧录器转到 AT89C2051 Flash 内部，再把该 IC 插入控制板上即可进行操作验证。你可以用数字把状态显示出来，当然也可以通过控制板上的 RS485 接口将信息传给 PC 机或其他具有 RS485 的设备。



（a）元件面实物图　　（b）焊接面实物图

图 9-1　AT2051 的完整图

9-3 线 路 分 析

AT2051 控制板的线路共分为 3 部分，第 1 部分是 AT89C2051 CPU 与稳压电路部分，

见图 9-2，右侧还特别接了一个三根引脚的温度感测 IC，另外还有一个按键输入。第 2 部分则是七段数字显示与蜂鸣器驱动部分，见图 9-3，为何不多一位数改用三位数字？主要原因是考虑到 PC 板不要过大，以免不方便携带。第 3 部分是 RS485 通信的电路，见图 9-4，SIO 与 $\overline{\text{SIO}}$ 最后接到板边的端子上。接下来，我们以功能单元来说明 AT2051 控制板的线路原理。

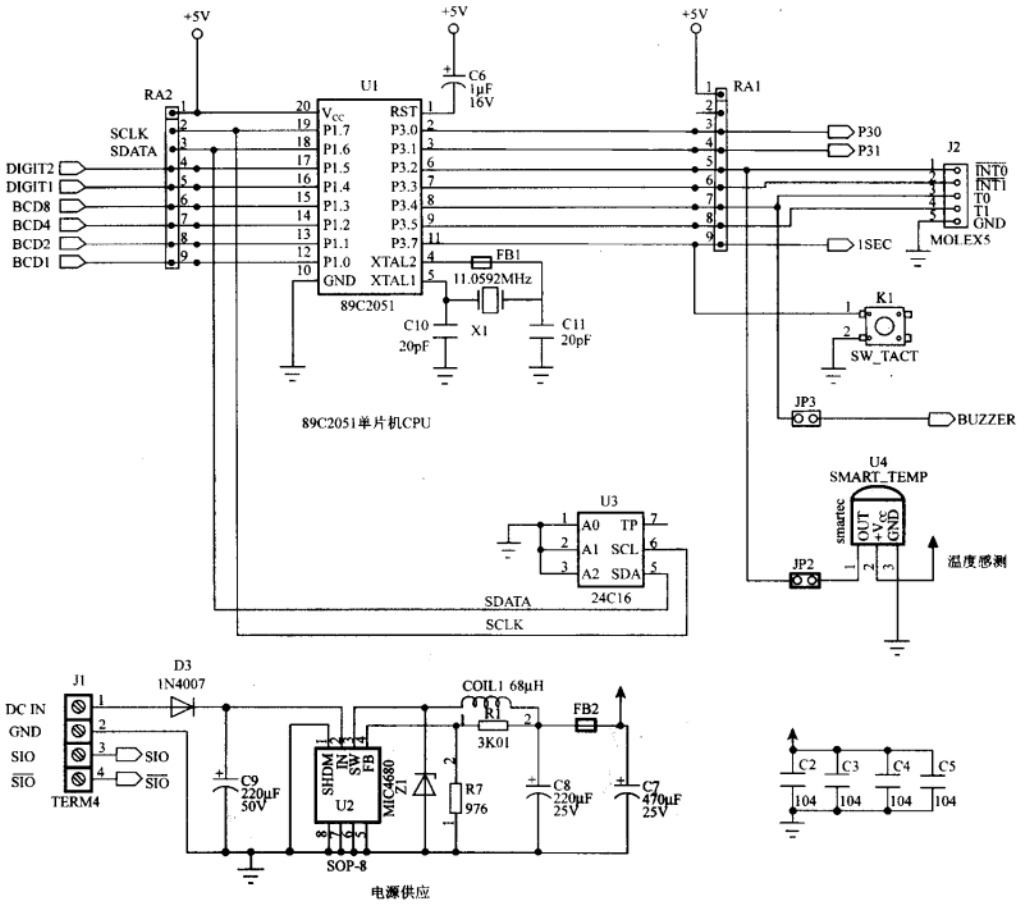


图 9-2　AT2051 控制板线路一：CPU 与稳压电路

（1）CPU AT89C2051：该 IC 本身就是一块非常省电的 CPU，我们规划 P1.0 端口作为数字显示驱动与串行 EEPROM，如图 9-4 所示。我们在 P1.0 与 P3.0 端口上都加上上拉电阻，让这两个端口不论是输出或输入都有正确的电压电平。AT89C2051 的第 4 引脚与第 5 引脚为石英晶体的输入点，照原厂的建议：加 20pF 右的小电容到地端。同时我们在 XTAL2 端加上一个磁珠（Ferrite Bead），这可抑制部分的高频信号，亦即压抑 11.0592MHz 的奇次谐波。

（2）串行 E^2PROM 24LCl6 的读写端口：24LC16 的功能与我们在网卡上常见的 93C46 类似，但是前者是标准 I^2C 规格（后述）且只要两根引脚就能读写，而后者需要 4 根引脚才能运作，读写的波形较为烦杂，理所当然的程序占的空间也较多。因此，我们当然选择较好的 24LC16。

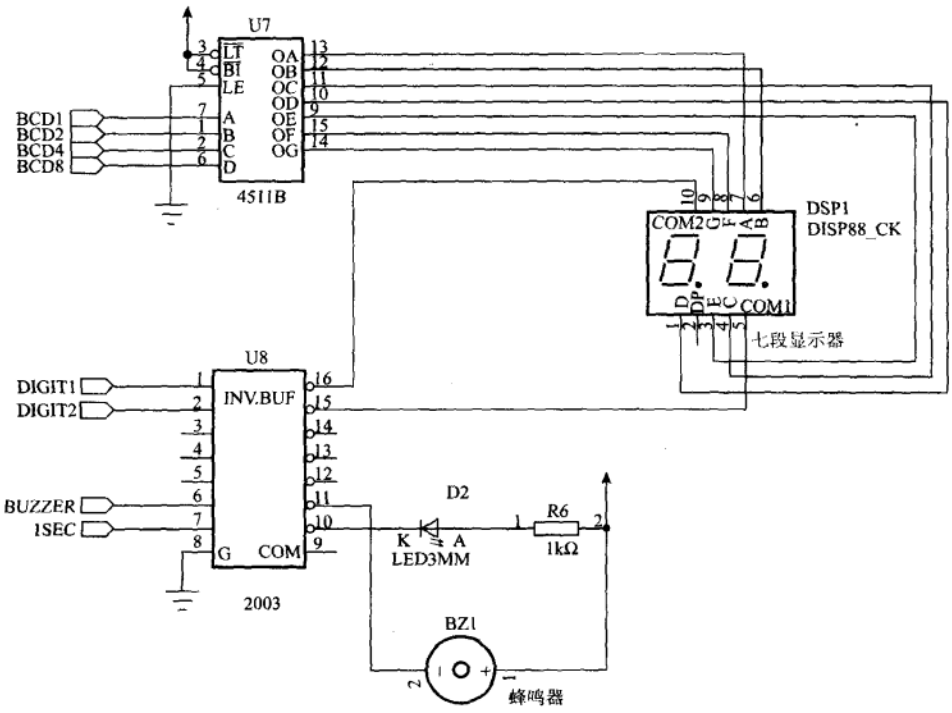


图 9-3　AT2051 控制板线路二：显示与蜂鸣器部分

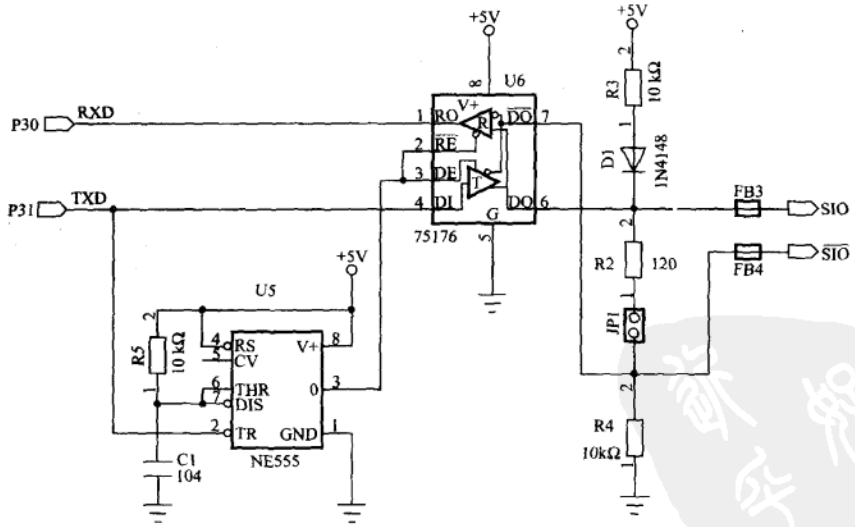


图 9-4　RS485 串行通信部分的电路图

（3）温度感测 IC 部分：这是一个温度感测元件，我们称为 Smart Sensor，它把测到的温度值以数位脉冲的方式表现出来，我们只要读取该脉冲的负载周期（Duty Cycle），再经过公式转换，即可得到摄氏温度值，这一方面用汇编语言来处理。为什么要设计一个温度

感测器呢?因为 Smart Sensor 的线路十分简单，但实用面十分地广泛，而一般电子方面的书籍又较少介绍，因此特别加以介绍。当然你也可以加装其他的感测元件，单片机的 I/O 能力是无限宽广的。

（4）电源稳压部分：大部分的 8051 学习板都是用 7805 配合散热片做电源稳压，这种做法成本最低，但是电源使用效率也低，如果输出入电压差过大时，散热片会很烫，几乎会有一半的电量都变成热量散失掉。在 AT2051 控制板上我们改用交换式的稳压 IC MIC4680 Switcher 开关。它是一个工作在 200kHz 的 1A 稳压器，只要配合电容电感，快速二极管以及回授分压电阻就可以组合成一个体积超小的稳压电路，值得一提的是 MIC4680 的转换效率高达 90%，就是用电池供电也是可行的。MIC4680 的输入电压非常宽广，从 6～34V 的直流都可以接受。

（5）按键部分：从线路图上可以看到，AT2051 的按键与 LED 显示是占相同的位（P3.7），当按下 K1 键时，LED 灯是一定不亮的。从设计上的观点来谈，LED 显示的机会始终比按下按键的机会还高，所以与其分别占用一位，不如把输入/输出点合并在一起。当输入/输出点数有所限制时，我们习惯用这种方式有效地降低实际的 I/O 数。

（6）RS485 通信电路部分：请再参考图 9-4，75176 的第 2 脚与第 3 脚分别控制 RS485 的收送状态，在电路当中，我们用一颗 555IC 做成 lms 的单击触发，当 AT89C2051 的 Start 位开始时，就把 75176 切换成输出的模式，以方便把串行信号送到传输线上。线路上我们加入电阻与二极管，主要用意是提供适当的电平以及传输线上有异常电压时，可适度保护 485 的 IC 不致损坏。

（7）七段显示电路部分：AT2051 控制板上有两个七段显示器。显示的数值由 P1.0～P1.3 决定，这 4 个位再经过 4511，把 0～9 的 BCD 码转换成七段显示码。而显示的十位数与个位数则由 P1.5 与 P1.4 来选择。当 P1.4=1 且 P1.5=0 时十位数字点亮，当 P1.4=0 且 P1.5=1 时个位数字点亮，若 P1.4 与 P1.5 都为 0 时，显示屏就不做任何指示。这个显示电路是属于扫描（SCAN）的方式，也就是说：我们要不断且定时去分别点亮这两位数字，才能很清楚看到这两个字。这部分程序将在定时中断的章节中提及。

（8）蜂鸣器驱动电路：AT2051 控制板上有一个直径为 12mm 的蜂鸣器，我们这里选定的是自我振荡型的蜂鸣器，只要在蜂鸣器两端加上超过 3V 的电压，蜂鸣器就会叫个不停。蜂鸣器是通过 2003 电流放大 IC 来控制。当 P3.4=1 且 JP3 接通时，蜂鸣器就操作，反之则停止鸣叫。

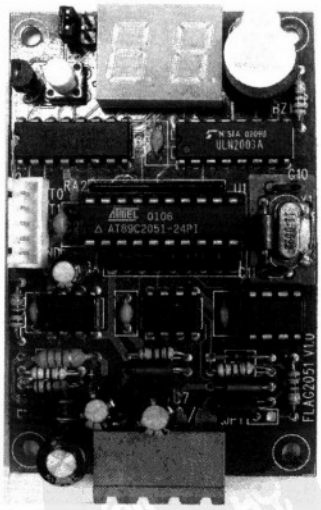


图 9-5　控制板的实体放大图

9-4　AT2051 控制板的应用与学习方向

因为 AT2051 配备有简单而实用的温度检测 Smart Sensor，所以很简单就可以做出许多温控的应用实例。例如：

（1）温度监视器：你可以把 AT2051 控制板安置在书房的墙壁上，随时观察温度的变

化。

（2）温度变化记录器：AT2051 控制板上已经有 RS485 的连线功能，除了温度的即时显示外，你还可以在 PC 个人电脑上写一个简单的程序，定时去读取 AT2051 控制板上的温度，即时记录某个环境的温度值。

（3）温度监控器：AT2051 控制板上有蜂鸣器，我们的程序可以针对温度值设定上下限值，当温度超过上限值时，启动蜂鸣器的声响。反之，当温度小于下限值时，点亮 LED 以提醒人们留意。

（4）温度控制器：AT2051 板结合大同电锅，将电锅的温度维持在 42℃，同时把牛奶与酵母菌混合在一起，8 个小时后就可以制造出一整锅的纯优酪乳。

（5）AT2051 控制板的深度应用一：定时读取温度值以便判断是否启动风扇或冷气机，控制的若是后者就不能经常开开关关。

（6）AT2051 控制板的深度应用二：养殖环境的温室控制，结合 PC 机及相关设备的控制，维持该温室的温度在某个范围内。

（7）AT2051 控制板的深度应用三：实验环境的温度变化监视，同时结合 Excel 绘出相关的图表与温度变化的情况。

（8）AT2051 控制板的深度应用四：恒温箱的控制，配合 AT2051 控制板温度值的读回，监视温度控制的反应速度。

实际上温控是 AT2051 的众多应用之一而非全部，在本书往后章节以及相关书本中您还会看到更多的应用案例。当然，真正的应用还是要依您的需求及创造力来实现的。

学习方向

也许会有许多人纳闷：只用一块控制板再加上一台 AT89C2051 的烧录器，少了 ICE 及 ROM 模拟器，这样写得了汇编语言吗?其实，这样真的就可以开始写程序了。只要我们的程序都经过模块化的处理，控制板上又有串行通信接口，再搭配示波器及万用电表，这样就有足够的除错工具了。你另外要学习一些基本除错的技巧，就能派上用场了。我们预期你会学到以下相关的知识：

（1）8051 硬件分析技巧。
（2）8051 汇编语言的熟悉。
（3）中断服务程序的技巧。
（4）AT89C2051 烧录程序的认识。
（5）七段显示器的程序技巧。
（6）按键输入的技巧。
（7）串行式 E^2PROM 的使用。
（8）I^2C 的程序写法与验证。
（9）省电模式的探讨。
（10）串行通信的探讨。
（11）RS485 通信的认识。
（12）温度测量。
（13）温度的校正与调整。
（14）相关仪器的使用。

9-5 AT2051 元件表及元件照片

AT2051 控制板上几个重要元件的实体照片号及备注说明如图 9-6～图 9-19 所示。

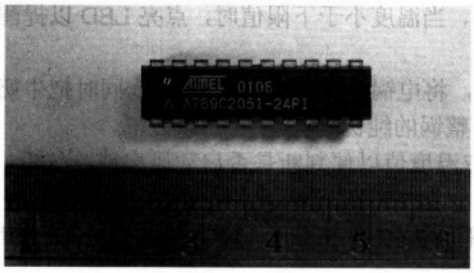


图 9-6 AT89C2051

注：ATMEL 公司的 2KB Flash 微机控制芯片。

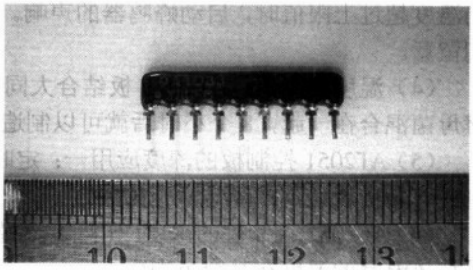


图 9-7 电阻

注：电阻的共同点上会有一个圆圈记号。

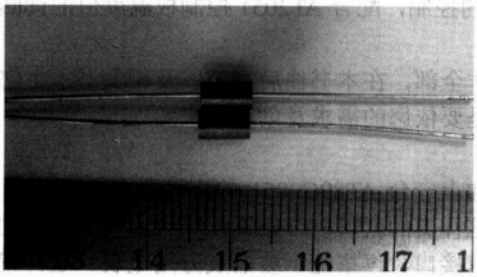


图 9-8 磁珠

注：磁珠可以滤掉高频信号。

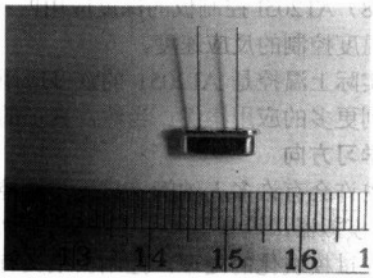


图 9-9 石英晶体

注：AT2051 控制板要做串行通信所以使用的频率是 11.0592MHz。

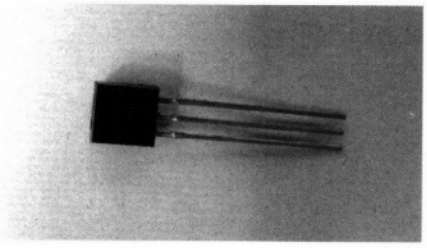


图 9-10 温度感测 ICSMT160-30-92

注：只有三根脚的温度感测 IC，以方波信号代表其度量到的温度值。

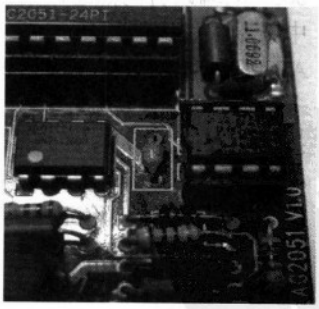


图 9-11 24LC16

注：16KB 的串行 E^2PROM，以 I^2C 的方式存取数据。

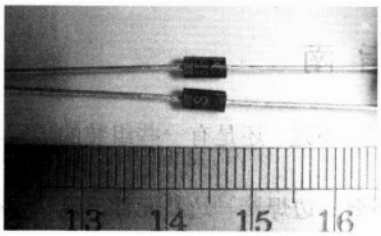


图 9-12　快速二极管

注：快速二极管样子与整流二极管相似，其工作频率约在 50kHz 以上交换式稳压 IC。

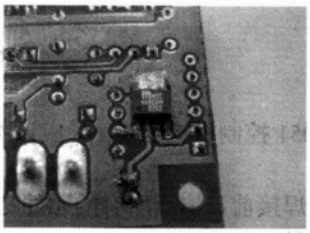


图 9-13　MIC4680B

注：交换式稳压 IC 的好处是输入范围广、转换效率高。

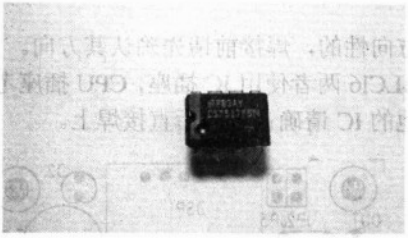


图 9-14　75176

注：RS485 串行通信最常用的传送接收 IC。

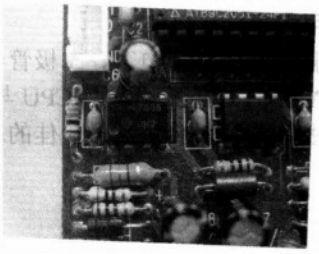


图 9-15　555

注：相当常用的计时 IC，我们称之为 Timer。

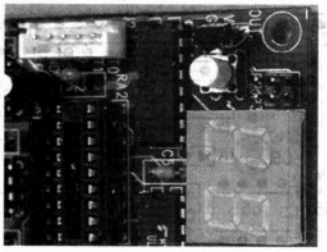


图 9-16　4511B

注：七段显示器专用的驱动 IC。

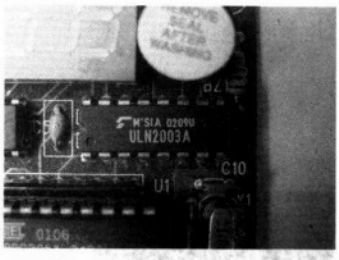


图 9-17　ULN2003

注：电流放大 IC，我们用这颗 IC 去驱动 LED、RELAY 或小型电动机。

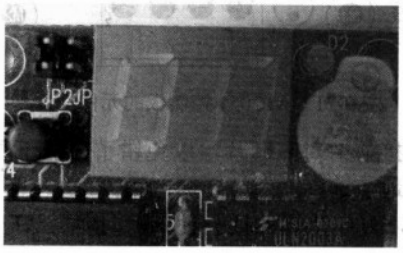


图 9-18　两位数七段显示器

注：这里使用的显示器是共阴极的，即其内部的所有负端是接在一起的。

图 9-19　蜂鸣器

注：内置振荡型的蜂鸣器只要加上额定电压即可发出声响。

9-6 组 装 指 南

AT2051 控制板自己装上，应该不会有很大的问题，不过，还是有一些组装的步骤与原则要遵循：

（1）焊接前，清点所有的元件是否备齐，千万不要一边焊接一边找元件，这一定会影响焊接的质量。

（2）从元件高度最低的开始焊接，所以应该从 U2 MIC4680 SMD 元件开始，这个稳压 IC 要焊在“焊接面”上，SMD 焊接时先让电烙铁吃一点锡，把 SMD 元件放正，然后对角各焊一根引脚，确认其他 IC 脚都对齐后，再焊接其他脚。为了使其散热系数更好，请把第 5 到第 8 脚全面用焊锡包括起来。

（3）除了电阻外，电容、二极管与 IC 都是有方向性的，焊接前请先确认其方向。

（4）AT2051 控制板上只有 CPU 与 EEPROM 24LC16 两者使用 IC 插座，CPU 插座考虑到要经常插拔，最好是使用质量较佳的 IC 插座，其他的 IC 请确认方向后直接焊上。

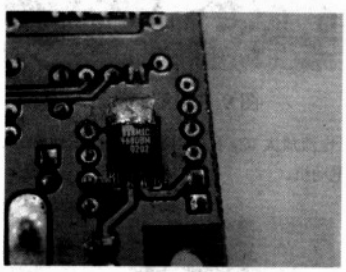


图 9-20 MIC4680 焊接实物图

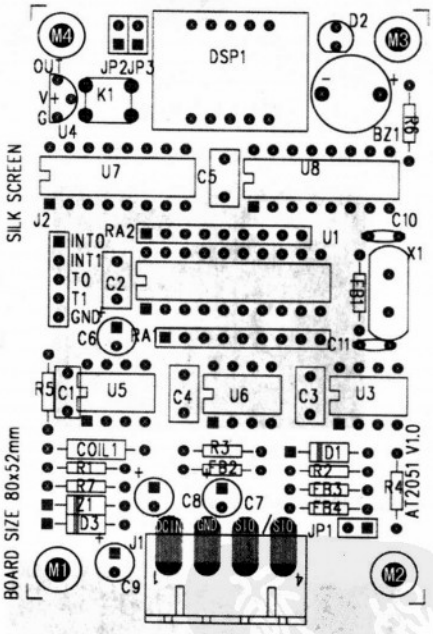


图 9-21 AT2051 控制板的元件位置图

（5）七段显示器与蜂鸣器都是有方向性的，焊接前请对照图片，以免错了要拆下重焊。

（6）石英晶体焊好后，请再用焊锡将其外壳与地点连在一起。

（7）温度 SMART Sensor 的引脚请对照线路图，方向错误的话会使该 IC 损坏。

（8）扩展接头 J2 焊接时也请遵循图片的方向。

（9）整块 PC 板焊接好后，一定要用目视的方式检查所有的焊点，不能有任何虚焊的情形发生。

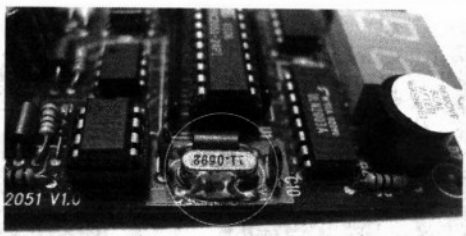


图 9-22　石英晶体外壳另外用焊锡与地 GND 接通图

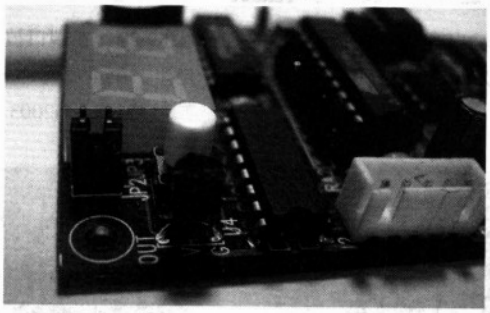


图 9-23　温度感测 IC 实物图

9-7　组装的测试步骤

AT2051 控制板焊接完后，一连串的硬件检查操作是少不了的，这是在确保我们写程序前硬件是完全正常的。我们把测试步骤分成两大部分，首先是控制板不放上 AT89C2051 CPU，直接接上直流电源，用数字电表检查各个重要的测试点。第二部分则是放上内置测试程序的 AT89C2051，然后通电检测，由于此时 CPU 已经有程序在运行了，可能要用数字式示波器观察测试点的信号波形了。

不放上 AT89C2051 CPU，接上+12V 直流电压，短路跳线 JP 都不接，请确认图 9-24～图 9-33 所示的情况。

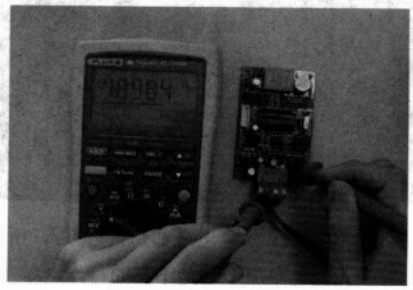


图 9-24　稳压之后的输出电压是+5V，测量值为+4.898V，误差应该在±5%以内

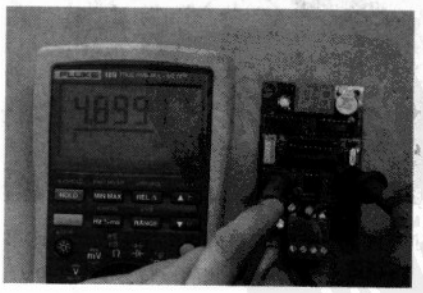


图 9-25　所有 IC 的电源脚都是+5V，测量值为+4.899V，接近+5V

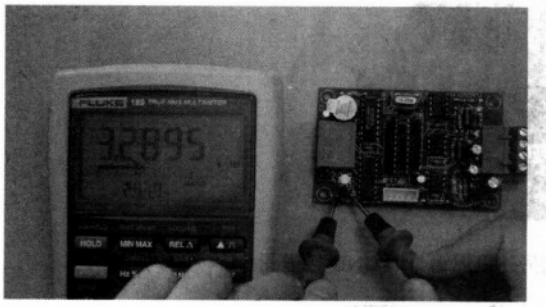


图 9-26　温度 IC 的 OUT 输出有脉冲产生，脉冲的频率测量值为 3.289kHz

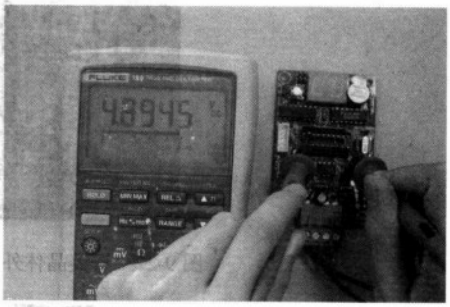


图 9-27　P1 端口被上拉电阻提到接近+5V 的电平，测量值为+4.894V，但 P1.5 与 P1.4 因为接到 ULN2003，其电压只有 2V 上下

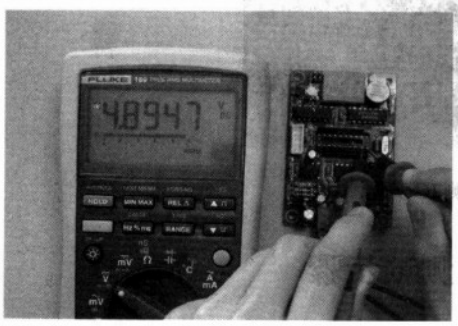


图 9-28　P3 端口被提升电阻提到接近+5V 的电平，测量值为+4.894V，P3.7 也接到 ULN2003，其电压只有 2V 左右

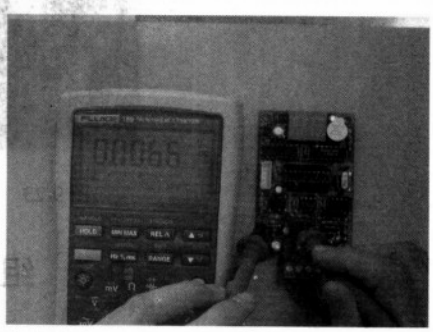


图 9-29　U5 555 的第 3 脚 OUT 输出始终在 0V 左右，测量值为+0.007V

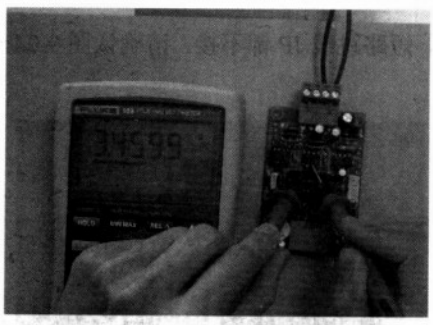


图 9-30　U6 75176 的第 6 脚的电位始终比第 7 脚高一点点，所量测的电位差为 3.459V

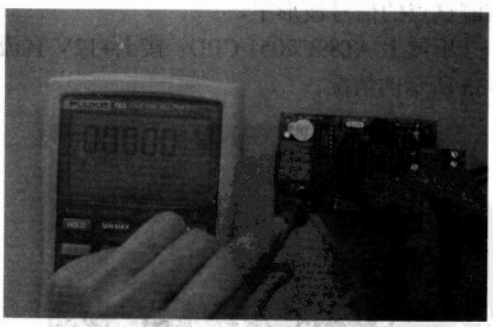


图 9-31　U7 4511 的输出端都是接近 0V，所以七段显示器是不亮的

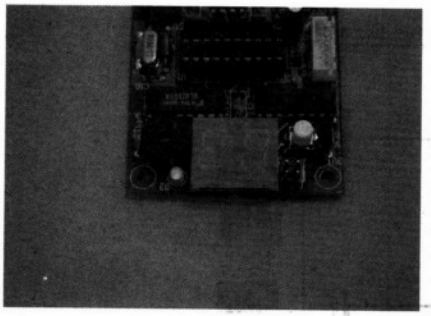


图 9-32　LED 是亮的

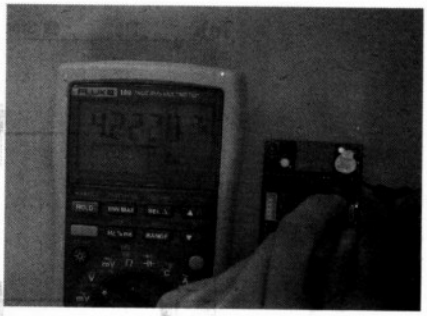


图 9-33　如果把跳线 JP3 接通，蜂鸣器就开始鸣叫，测量值为 4.223V

烧录一块含有测试程序的 AT89C4051 CPU，这个二进制程序文件的文件名为 TEST_ALL.TSK，占用的程序空间较大，约有 3KB 之多，所以需要用 Atmel AT80C4051 才能放入所有的程序码，烧录完后将该 IC 插到 AT2051 控制板，再接上+12V 直流电压，短路跳线，即 JP2 与 JP3 都接通：

（1）稳压电路的输出是否为+5V，如果不是，那就有可能你把 CPU 插反了。

（2）请再次检查所有 IC 的电源脚是否都是+5V。

（3）石英晶体的振荡频率是 11.0592MHz。

（4）七段显示器会显示出现在所测到的温度值。

（5）LED 每闪烁一次，代表 AT2051 更新一次温度值。

（6）蜂鸣器每秒鸣叫一次，这个速度与 LED 点亮的速度明显不同。

（7）AT2051 控制板每更新一次温度显示值时，会把温度值转成 ASCII 码，并从 RS485 串行端口送出。

以下硬件验证需要示波器及转换板配合：

（1）用示波器观察 555 的 OUT 输出，应该有 1ms 的脉冲出现。

（2）用示波器观察 75176 的输出，应该有一连串信号出现，而且 SIO 与 $\overline{\text{SIO}}$ 的信号刚好反相。

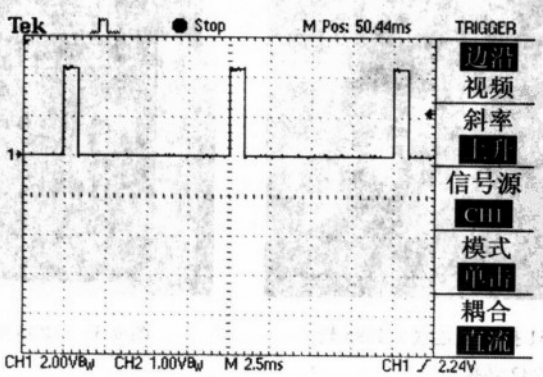


图 9-34　由示波器观察 555 的输出，每隔 10ms 有一个串行值送出

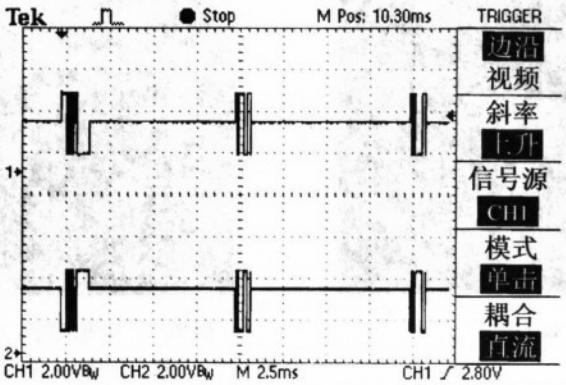


图 9-35　观看 RS485 串行端口上，SIO 与 $\overline{\text{SIO}}$ 的信号，也是 10ms 出现一次

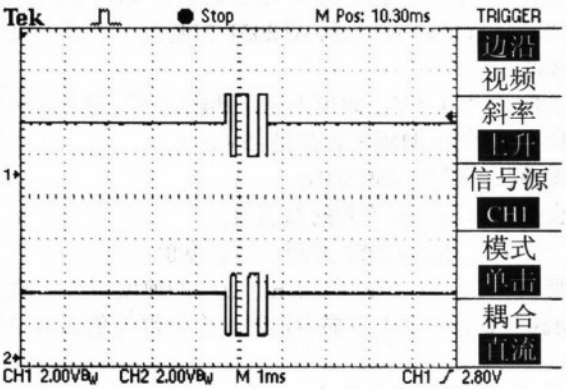


图 9-36　放大 SIO 与 $\overline{\text{SIO}}$ 的信号，两者是完全反相的

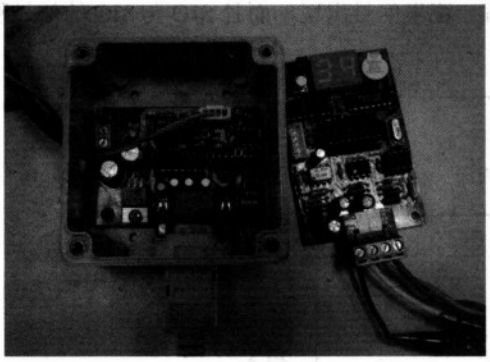


图 9-37　AT2051 控制板连接 RS485 转 RS232 转换板

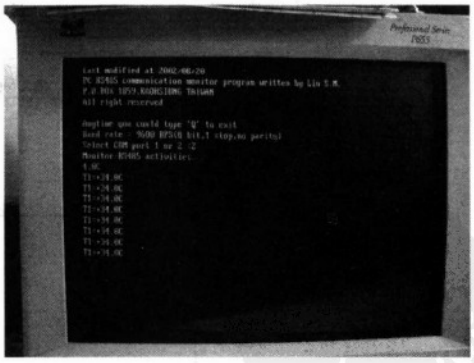
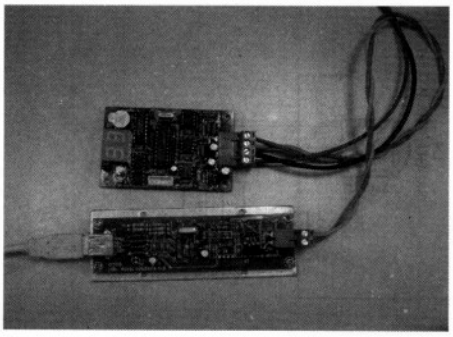


图 9-38　DOS 画面上回传的温度值

（3）PC 端若有加上 RS485 转 RS232 转换板，在 DOS 的环境下，执行 MONI_485．EXE

（RS485 端口的监视程序），应该可以在屏幕上能看到 AT2051 回送的 ASCII 温度值。

（4）PC 端若有加上 RS485 转 USB 转换板，并已安装 USB 的驱动程序，执行 VB 写的 AT2051.EXE 程序，应该能在 Windows 下看到正确的温度值。



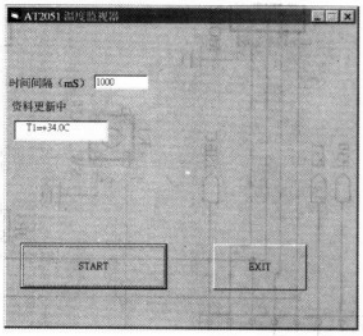


图 9-39　AT2051 控制板连接 RS485 转 USB 转换板，右图为接收到温度值的画面

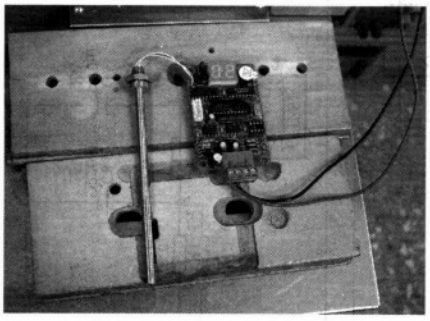


图 9-40　我们把温度感测 IC 埋入不锈钢管中，就变成标准的工业用温度测棒

图 9-41　恒温槽的温度设置是 10℃，用 AT2051 控制板量到的温度是 9℃

（5）用手接触温度 IC，应该能看到温度一直上升，升到 34℃或 35℃。

（6）装冰水的杯子接近温度 IC 时，显示的温度应该下降到 10℃以下。

9-8　测试点的电平与波形观察

我们在 AT2051 控制板上总共列出了 11 个测试点，当你把 PC 板装好后，首先要确认的是稳压电路的+5V 输出是否正确，接下来烧录一个 TEST_PGM.TSK 文件到 AT89C2051 当中，并将此 CPU 插到 AT2051 控制板（注意方向）上，这个程序只做温度量测及显示，并不把数据送到 RS485 端口上。你可以用示波器与电表来检测这些测试点的信号。只要你测得的信号与我们列出的相符，你可以认定：硬件一定是没问题的，你可以放心地开始写 8051 的汇编程序了，当你怀疑硬件有问题时，可以随时烧录该文件进行再次的确认。

以下是 11 个检查点的电表读值或示波器信号波形图示。

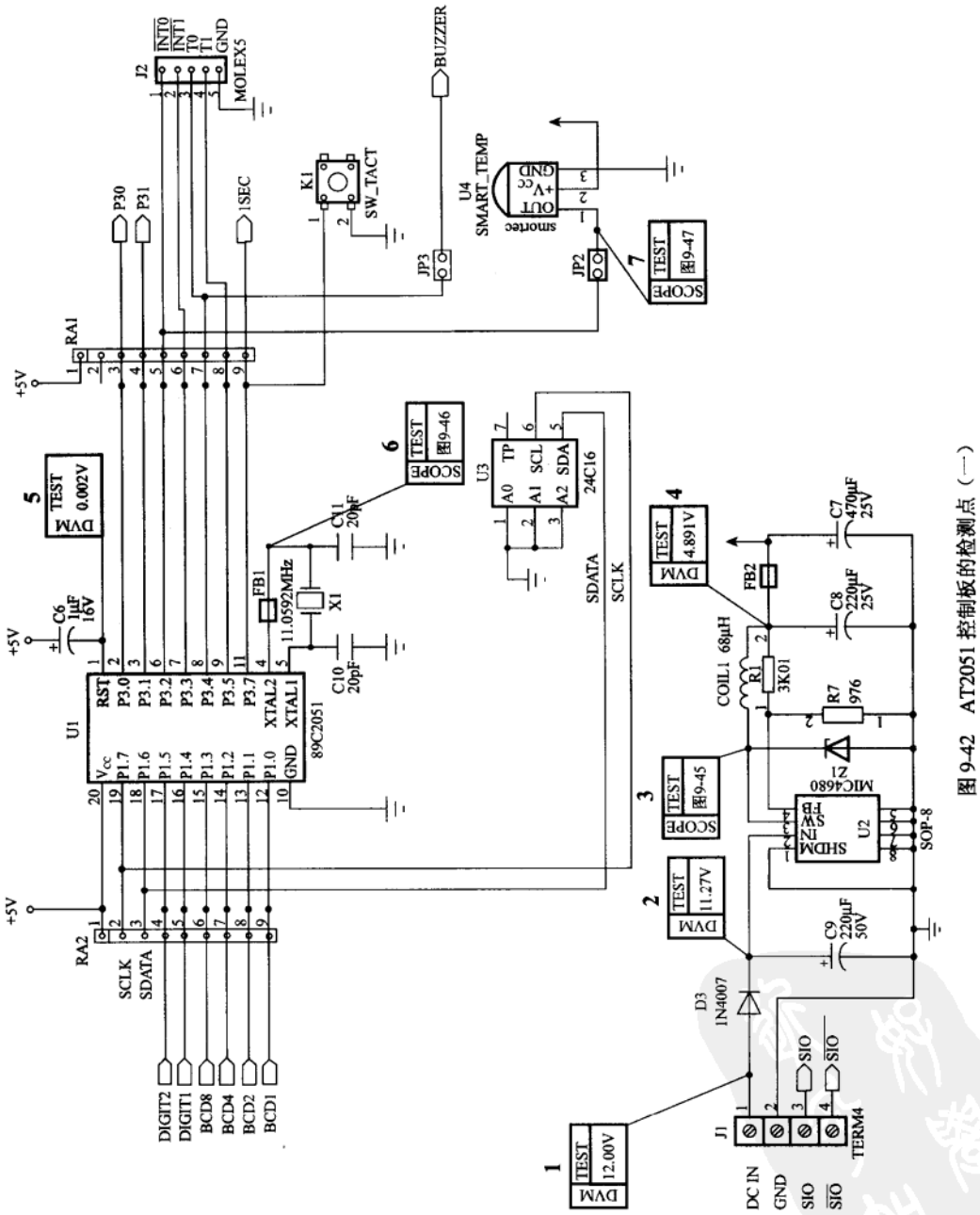


图 9-42　AT2051 控制板的检测点（一）

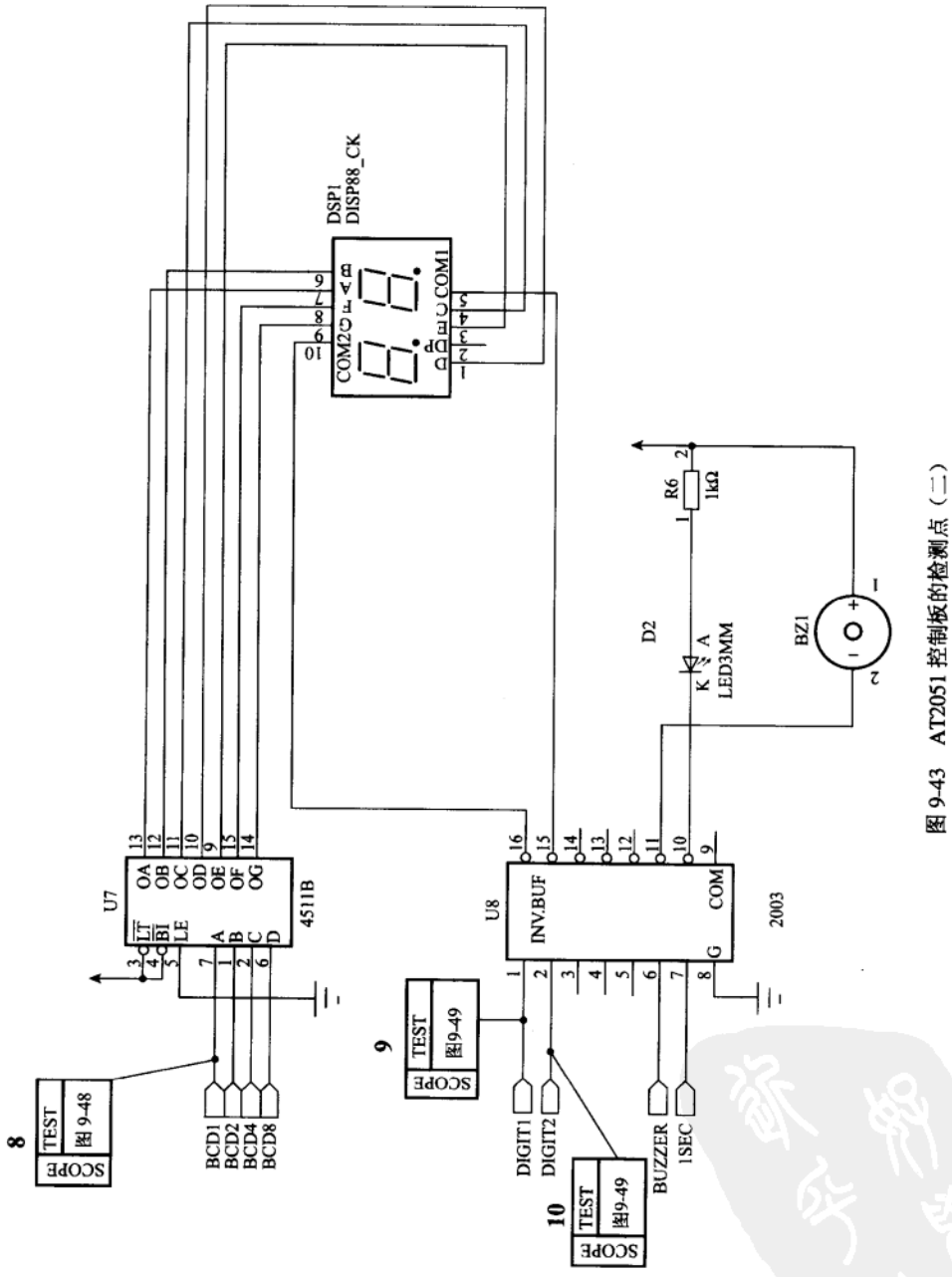


图 9-43 AT2051 控制板的检测点（二）

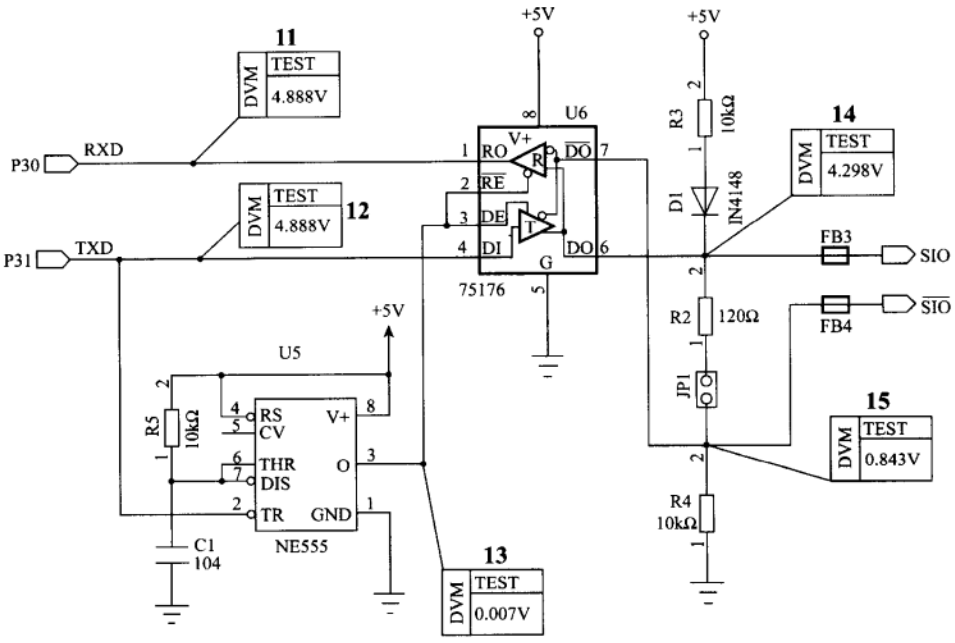


图 9-44　AT2051 控制板的检测点（三）

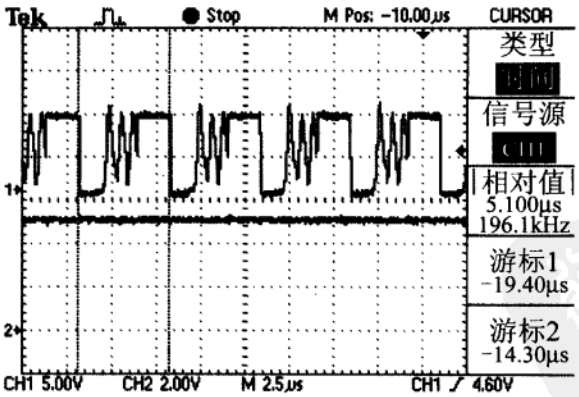


图 9-45　MIC4680 交互式稳压 IC 的输出，其工作频率约在 196kHz 左右

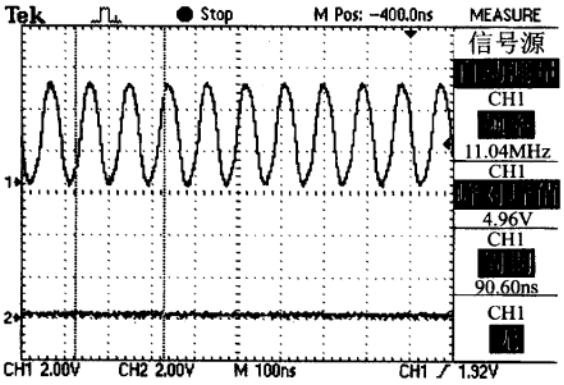


图 9-46　AT2051 控制板的石英振荡频率约为 11.04MHz（测量值）

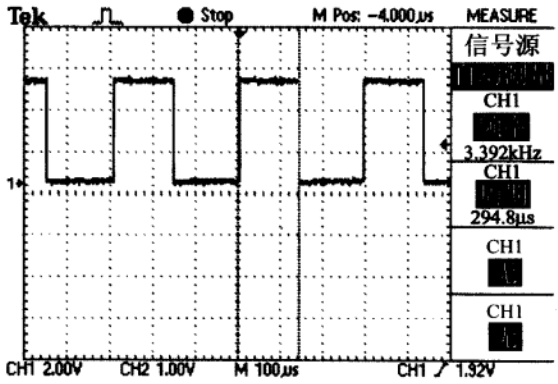


图 9-47　温度感测 IC SMT160 的方波输出信号

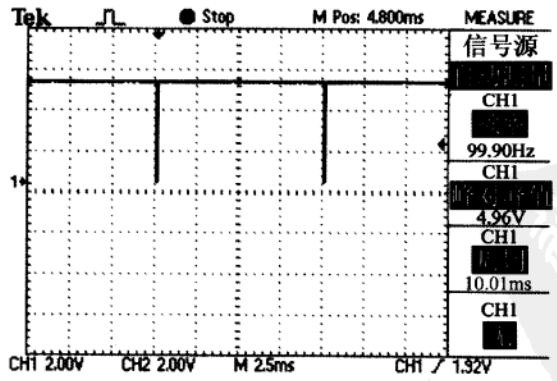


图 9-48　每隔 10ms 时间，CPU 对 4511 送出两组显示的数字显示值

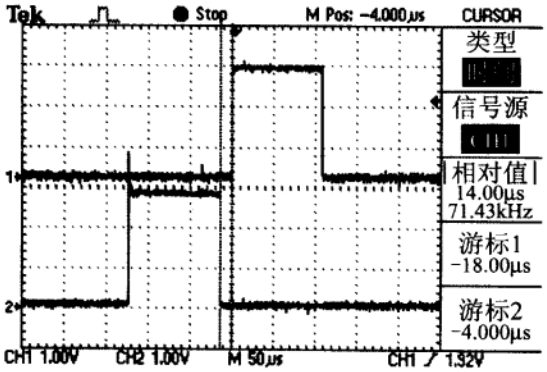


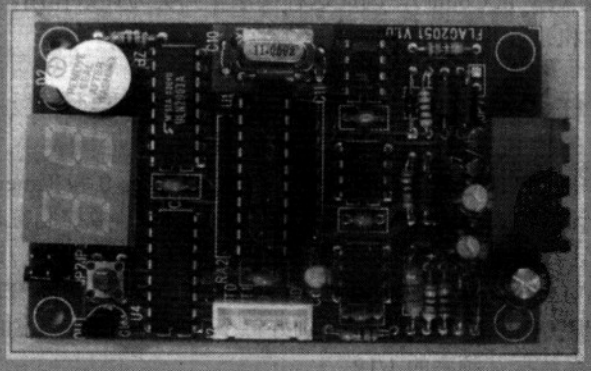
图 9-49　DIGIT1 与 DIGIT2 的输出波形，请注意两者间有一小段空格，即数字与数字间要先熄灭后再点亮

习　　题

1. 8051 在实际应用上应具备哪些重要观念？请列举 5 种。
2. AT2051 控制板有哪些特点？请列举 5 项。
3. AT2051 控制板所用的稳压 IC 和常用的 7805 有何不同？
4. AT2051 控制板可做哪些方面的应用？请列举两项并详加说明。
5. AT2051 控制板所用的电流放大 IC 有什么用途？
6. 如何焊接 SMD 元件？
7. 简要说明 AT89C2051 的电源脚与接地脚的脚位。
8. 测试中 75176 的第 6 引脚及第 7 引脚比较，哪根引脚的电位较高？

彻底研究篇

10



AT2051 控制板上有标准的 RS485 收发 IC：75176，它的传输速率最快可达 10Mb/s，但我们只用到 10kb/s 位，还用这个速率可以很方便地与 PC 连通

第 10 章　8051 定时/计数彻底研究

定时与计数是 8051 单片机的专长之一，我们可以利用这个功能来产生定时中断，对待测信号定时或计数，8051 有两组 Timer，8052 则有三组 Timer。本章将探讨定时与计数相关寄存器的设置与用法，Timer 的操作是 8051 使用中非常重要的一个环节，将其配合中断与串行传输，即成为 8051 相当成功的典范。

10-1　什么是定时/计数

在单片机的控制领域中，定时和计数的应用占了相当重要的地位。例如，我们想让 CPU 每 10 ms 就做一次中断，这就是定时器（Timer）的典型应用；又比如，我们想对某数字信号的发生次数做计数，则是计数器（Counter）的一种应用。较早问世的 CPU 如 Z80 或 6502 内部都不包括定时/计数器的电路，必须靠其他专用的 IC 来达成定时/计数的功能。8051 单片机的 CPU 由于号称使用单一芯片就完成控制的目的，所以都把定时/计数的线路并入 CPU 中，以方便就近控制，这种安排不仅减少了线路板的面积，而且使得定时/计数的控制灵活性发挥得淋漓尽致。

定时和计数有何不同呢？对单片机 8051 而言两者都是一视同仁的，主要的差别是信号源的不同罢了，8051 都是以内部的 Counter 来接受这些信号，假如信号源是 CPU 内部的 OSC 振荡源时，由于振荡频率已知，则 Counter 上的值单位为秒，所以称这种接法是定时作用。假使信号源为外部输入的数字信号，频率未知且不一定保持稳定，所以 Counter 上的值仅能称为该信号的事件次数（Event Count），因此这种接法就归类为计数作用。在 8051 中，所称的 Timer 都同时有定时和计数的两项功能。

10-2　8051 定时器和计数器安排

8051 系列的 CPU 内部包含两个定时/计数器，分别是 Timer 0 和 Timer 1，这两个 Timer 统一由 TMOD（定时/计数模式控制寄存器）和 TCON（定时/计数控制寄存器）来掌握，而 8052 还多了一个定时/计数器 Timer 2，这个 Timer 则交由 T2CON（定时/计数器 2 控制寄存器）来控制。当这些 Timer 被设置成定时功能时，内部的 Counter 值随着每个机器周期而每次加 1，若系统使用的振荡频率是 12MHz 时，刚好每个机器周期是 1μs 的时间，所以此时的定时分辨率是 1μs，即系统振荡频率的 1/12。

当 Timer 被设置成计数功能时，它通过外部输入点 T0 和 T1 信号，每当这些信号由高电位降到低电位的负沿时，内部的 Counter 值加 1，这些 Counter 加 1 的操作完全以硬件方式做成，与我们的软件程序毫不相干，但是该计数信号仍有一些限制存在，在 8051 的英文手册指出，最高可接受的外部信号输入频率是系统振荡频率的 1/24，约为 500kHz 左右，这也就是

说，若外部的输入频率超过 500kHz 时，可能会造成 Counter 值的错误，这种错误可能会使得 Counter 上的值比正确值还少。8051 是在每个机器周期的 S5P2 阶段检查 T 0 和 T 1 的状态，若此时的状态为高电位，而下一个机器周期的 S5P2 阶段为低电位时，内部的 Counter 值自行加 1，所以输入的外部信号不论其周期比（Duty Cycle）为多少，其中较短的状态必须保持一个机器周期以上的时间，才得以让内部的 8051 计数器操作。

8051 的 Timer 0 和 Timer 1 分别有 4 种模式可供切换使用，而 8052 的 Timer2 则另外有 3 种特定的操作模式：自动获取（Capture），自动载入（Auto-Reload）和波特率发生器（Baud Rate Generator），这些功能将在另一本《8051 单片机彻底研究——实习篇》中再做详细介绍。

10-3 定时/计数器相关的寄存器

8051 的两个定时/计数器统一由两个寄存器来控制，寄存器 TMOD 控制 Timer 的定时或计数模式，该寄存器各以 4 个位指定 Timer 0 或 Timer 1 的工作模式，必须留意的是本寄存器无法进行位寻址。TCON 寄存器内部指示两个 Timer 的工作结果，此 8 个位都可以进行位寻址，我们可以通过设置 TCON 的状态，来停止或启动 Timer。当定时或计数器因溢位而产生中断时，会将 TCON 中的部分位设置成 1，以便程序检查这些位。由于 8051 将 Timer 的功能复杂化，若不对 TMOD 及 TCON 两个寄存器彻底了解，将无法有效地控制 Timer，所以彻底理解这些控制位绝对是有必要的。

谈到定时/计数器时，最好靠图示的方法了解 Timer 真正的操作原理，如下图所示，这是 8051 Timer 操作的框图。

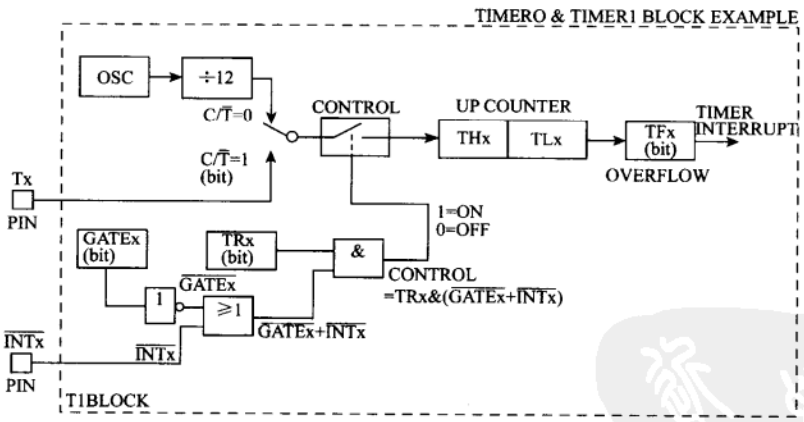


图 10-1　8051 Timer 操作框图

图 10-1 是 Timer 0 或 Timer 1 的操作解析图，由左到右分别列出了信号源信号控制方式，计数器和溢位后所设置的位，当我们指定 Timer 为定时模式时，其 C/T 位就必须为 0，以每次 1 个机器周期的频率接受输入信号，若系统所使用的振荡晶体是 12MHz 时，Timer 所接受的信号频率正好是 1MHz。若指定 Timer 为计数模式时，C/T 位成为 1，Timer 接受由外部输入的数字信号（T0 和 T1），不论 Timer 属于定时或计数模式，我们都需要一个控制闸（管制

点），来决定是否接受定时或计数的数字脉冲，8051 总共安排了 TRx、$\overline{\text{GATEx}}$ 和 $\overline{\text{INTx}}$ 三个控制点。前两者是属于软件可控制的位，最后一个则是外部硬件控制点。由图中我们可以看出总控制点的逻辑状态是：

CONTROL=（$\overline{\text{GATEx}}$ + $\overline{\text{INTx}}$ ）&TRx

'+' 代表逻辑上的 OR，'&' 代表逻辑上的 AND，如果允许 Counter 操作必须 CONTROL 值等于 1，只有在 TRx 为 1 而且 GATE 为 0 或 INTx 为 0 时，CONTROL 才等于 1，这意味着若要让 Counter 计数时，必须用程序将 TRx 设成 1，另外再借助 GATEx 软件位或 INTx 硬件控制打开允许计数之门，当计数器算到其最大值时，若再有一个脉冲信号进入 Counter 值，就会导致 TF0 或 TF1 变为 1，该位刚好可作为对 CPU 中断要求的信号，若 CPU 处理该中断时，TF0 或 TF1 则会自动清除为 0。

图 10-1 中的 Counter 可以是 8 位、13 位或 16 位的宽度，主要视设置的模式而定，同时 Counter 会因设置模式的不同，而有自动载入和不能自动载入之分，若能自动载入则只需对计数器做一次载入的操作即可，反之时，必须在每次溢位后，在程序中重新对 Counter 载入计数值，否则得到的 Counter 值将是错误的。

1. TMOD 模式控制寄存器解析

8051 上的 Timer0 和 Timer1 的功能几乎是相同的，这两个定时/计数器的工作模式各由 TMOD 寄存器上的 4 个位来控制。C/T 决定 Timer 是计数或是定时，$\overline{\text{GATE}}$ 则是允许或禁止 Timer 操作的控制点，剩下的 M1 和 M0 两个位决定 Timer 的工作模式。

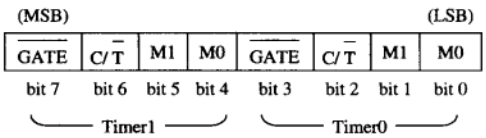


图 10-2　TMOD 模式控制器寄存器内部安排

注：TMOD 寄存器不可做位寻址。

M1=0，M0=0：模式 0，Counter 被设置成 8 位的计数器，高位组计数器（THx）占 8 个位，低位组计数器（TLx）则占 5 个位，这个工作模式与 8048/8049 的 Timer 工作模式类似。

M1=0，M0=1：模式 1，Counter 被设置成 16 位的计数器，THx 和 TLx 串接，计数值最高可达 65536。

M1=l，M0=0：模式 2，Counter 被设置成 8 位可自动载入的定时/计数器，实际操作的 Counter 是 TLx（8 位），每当 TLx 计数到产生溢位时，THx 上的值会自动载入 TLx 上，以方便 TLx 继续计数。

M1=1，M0=1：模式 3，本模式被设置后，Timer 1 将停止操作，而 Timer 0 被分割成两个 8 位计数器，TL0 由 Timer0 的 GATE 控制，而 TH0 则改由 TRl 位来控制。

2. TCON 控制寄存器解析

Intel 的手册将此寄存器称为 TCON，代表是 Timer 的控制寄存器，不过由其内部的功能看来，应该再加上 Status（状态显示）的功能才对，因为其中有些位是我们必须设置的（TRx 和 ITx），而有些位（TFx 和 IEx）却是在某些计数器条件成立下才被硬件自动设置的，所以后者可视为 Counter 的状态位（Status Bit）。

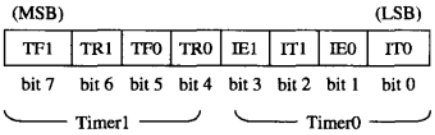


图 10-3　TCON 寄存器的安排

TRx：定时/计数器的软件控制位（Run Control Bit），我们可以在程序中直接寻址本字节（TR 0 或 TR 1），以便控制 Timer 的开始定时或计数。

ITx：外部中断的控制位（Type Control Bit），此位不属于定时/计数器的功能。它指定外部输入的中断要信号是属于下降负沿触发（Falling Edge）或是低电平触发（Low Level）。

TFx：定时/计数器上数到溢位发生时，本字节被设成 1，请参看图 10-1 的右方，本位实际上是一个对 CPU 要求中断的信号，当 CPU 反应并处理该中断时，TFx 会自动清除为 0。

IEx：外部中断成立的反应标志位，本位也是一个对 CPU 请求中断的信号，当 CPU 对此中断有所反应时，该 IEx 会自动清除为 0。

TCON 寄存器上共有 4 种功能：设置或禁止 Timer 工作、Timer 溢位时的状态显示、外部中断信号的定义以及外部中断信号成立时的状态指示。

10-4　8051 的 Timer 定时/计数器设置步骤

Timer 0 和 Timer 1 的设置步骤几乎是相同的，我们分别要定义 TMOD、TCON、TH0、TL0、TH1 和 TL1 等寄存器后，定时计数器方能正确操作，虽然这些操作转换成指令后，仅需数行汇编语言即可解决，但在程序刚规划的阶段时，每个 Timer 的操作和控制模式都必须考虑清楚后才开始写程序，这样才可以使我们准确地掌握各个 Timer 的操作。

- 设置步骤 1：确定是定时或计数，定时的话，TMOD 中的 C/$\overline{T}$ 位为 0，计数时 C/$\overline{T}$=1。
- 设置步骤 2：确定定时/计数时是否受外部的硬件信号控制。若不受外部 INT0 或 INT1 引脚控制时，TMOD 中的$\overline{GATE}$ 位为 0，否则需设成 1。
- 设置步骤 3：决定计数器的模式，模式 0 时最大计数值为 8192 且不能自动载入，模式 1 时最大计数值为 65536，同时计数器也没有自动载入的功能，模式 2 时最大计数值为 256 可自动载入计数值，模式 3 时有两个计数器，其最大计数值都是 256，各种模式的详细操作说明请看下一节的说明。
- 设置步骤 4：将前 3 个步骤的结果组合起来，成为一个给 TMOD 寄存器的设置值。
- 设置步骤 5：决定计数器（TH0、TL0、TH1 和 TL1）的设置值，由于 8051 上的计数器都属于可载入式的进位计数器（Up-Counter），所以正确的设置值应该是该计数器的最大可计数值减去欲计数值，然后将该结果分成两部分后，再传到 THx 和 TLx 两个计数器。
- 设置步骤 6：在适当的时机中将 TCON 中的 TR1 和 TR0 位设成 1，开始让 Timer 操作。

图 10-4 列出 Timer0 和 Timer1 有关的寄存器，若要正常操作所必须设置的寄存器和位，唯一要注意的是 TMOD 寄存器无法进行位寻址，所以必须两个 Timer 模式同时设置。

(MSB)							(LSB)	
				GATE	C/$\overline{T}$	M1	M0	TMOD
		[TF0]	TR0					TCON
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	TH0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	TL0

图 10-4（a） 与 Timer0 有关的寄存器

(MSB)							(LSB)	
GATE	C/$\overline{T}$	M1	M0					TMOD
[TF1]	TR1							TCON
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	TH1
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	TL1

图 10-4（b） 与 Timer1 有关的寄存器

10-5 Timer 模式 0 彻底研究

当 Timer 被设置成模式 0 工作时，计数器的宽度只有 13 位，其中 THx 占 8 位，TLx 则仅占 5 个位，所以真正要存入的计数值应该是：

TLx=（8192-计数值）除 32 取其余数值

THx=（8192-计数值）除 32 取其商数值

设置一个计数器的值看来颇不方便，但在 8051 的汇编语言中，可借用反汇编程序中的伪指令，自动做这些计算，以下就是这种最为方便的写法：

```
MOV TL0, #(8192-计数值).MOD.32
MOV TH0, #(8192-计数值)/32
```

图 10-5 说明了模式 0 上的工作框图，只要我们定义好 TMOD 值、计数器载入值及 TRx=1 后，就可让计数器往上数，最后到达计数器上限值时产生溢位并使 TFx 变成 1。以下的程序例即在示范此过程，由于 Timer1 与 Timer2 的设置方式都相同，所以本章大部分程序范例都是以 Timer 0 做示范。

目的：试验 Timer 在模式 0 下的定时操作情形。

程序名称：CH10_T1.ASM

辅助硬件

示波器或者逻辑分析仪。

程序操作说明

当程序烧录到 AT2051 控制板后，接上电源并且将测试棒夹到 P3．2 上，我们看到一串变化很快的数字信号串，该脉冲信号为 1 的时间很短，脉冲的周期约是 5.418 ms，任何时间量测时都得到相同的结果，图 10-6 是用示波器所测到的信号。

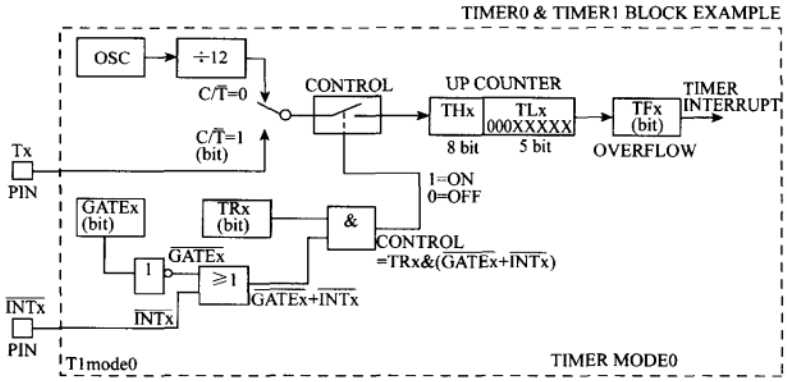


图 10-5　MODE 0 的工作模式

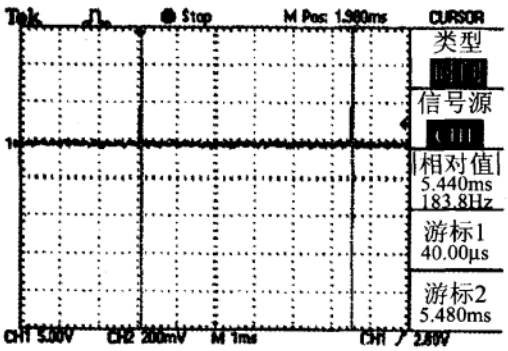


图 10-6　MODE 0 时，计数值位 5000，在 P3.2 所看到的输出信号

```
;PROGRAM NAME:T1.ASM
;TEST TIMER0 MODE 0:13 BIT COUNT UP
;
COUNT       EQU     5000
CHECK_BIT   REG     P3.2
;
            ORG     0
            MOV     R0,#00H
            DJNZ    R0,$          ;WAIT
            CLR     CHECK_BIT
            MOV     A,TMOD
            ANL     A,#11110000B
            CLR     ACC.3         ;GATE=0
            CLR     ACC.2         ;C/T=0
            CLR     ACC.1         ;TIMER0 M1=0
            CLR     ACC.0         ;TIMER0 M0=0
            MOV     TMOD,A
            MOV     TH0,#(8192-COUNT)/32
            MOV     TL0,#(8192-COUNT).MOD.32
;
            CLR     TF0           ;CLEAR TF0
```

```
            SETB    TR0          ;SET TIMER0 START TO COUNT
;
LOOP        CLR     CHECK_BIT
            JNB     TF0,LOOP
;TF0=1
OVER        MOV     TH0,#(8192-COUNT)/32
            MOV     TL0,#(8192-COUNT).MOD.32
            SETB    CHECK_BIT;P3.2=1
            CLR     TF0          ;CLEAR TF0 AGAIN
            AJMP    LOOP
;
```

图 10-6 当 8051 的 Timer 被设置成定时后，若系统使用的晶体为 12 MHz 时，其定时的精确度为 lμs，如果改采用 11.0592MHz 时，其定时精度变成 1.085μs。在程序中我们设置的计数值为 5000，所以 5000×1.085μs=5.440 ms，这跟我们在仪器上所得到的值 5.418 ms 相距不大。如果我们想把定时功能改成计数，只须将程序中的 CLR ACC.2 改为 SETB ACC.2 即可，然后由外部送数字脉冲信号进入 T0 点上，其余程序不做任何变更。在此时我们也发现了几个疑点，这是 Intel 手册中所没有提到的，或是说成没讲清楚的地方：

疑点 1：当计数器溢位后，除了 TFx 被设成 1 外，计数器只要 TRx 为 1 时，仍旧继续计数，而不是原先我们所以为的停止计数。

疑点 2：Timer 在 MODE 0 下计数时，手册上明白地指出 TLx 中仅有 5 个位有效，但是当这 5 个位都是 1 时，若还有另 1 个脉冲进入时，您认为会如何变化呢？如果

TH0=11100000B 、TL0=00011111B，当计数值再增加 1 时，TH0=11100001B 、TL0=00000000B 或是 TH0=11100001B、TL0=00100000B 呢？经过我们验证后应该是第二者才对，虽然 TL0 仅有 5 个有效位，除了本身会对 TH0 加 1 外，本身也会加 1。

疑点 3：计数器溢位后，计数值清除成 0 又重头计数，相当于重新载入最大的计数值，这种安排少了自动载入的功能，反而处处要程序依中断要求重新设置计数值，实用效果会打一些折扣。

疑点 4：综合以上数点我们可以断定，当初规划此模式的工程师内定想要有个定时/计数模式与 8048/49 兼容，才会如此设计，否则 THx 和 TLx 加起来共有 16 位宽，计数器的计数宽度应该是愈大愈好才对。

Timer 的模式 0 仅有 13 位宽，实用价值并不高，若真想做定时/计数应用时，请改用模式 1，其计数器宽度为 16 位，其余功能均与模式 0 相同。

10-6 Timer 模式 1 彻底研究

8051Timer 的模式 1 与模式 0 几乎相同，唯一的差别是 Counter 的宽度改成 16 位宽，其余特点都保留，图 10-7 是模式 1 的工作框图，由于在此模式下，TLx 和 THx 都是 8 位宽，所以其计数器载入值的写法应该是：

```
MOV TLx,#(65536-计数值).MOD.256
MOV THx,#(65536-计数值)/256
```

在模式 1 下计数器的最大值是 65536，若系统使用的频率是 12MHz 时，则定时可达 65.536 ms

之久，实际应用上应该足够，不过模式 1 仍与模式 0 相同，计数器均没有自动载入的功能，如果我们要做重复性的定时操作时，这时就必须靠程序强迫载入，以下的例子在验证模式 1 下的定时操作的精度如何。

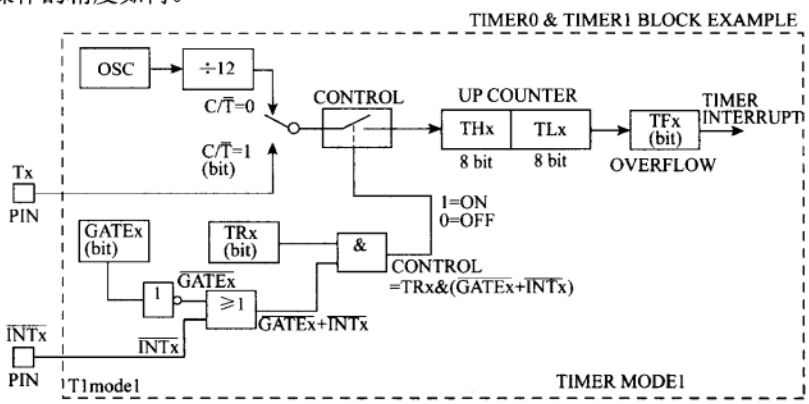


图 10-7　MODE 1 的工作模式

目的：试验 Timer 0 的 Mode 1 下定时精确度的测试。

程序名称：CH10_T2.ASM。

辅助硬件

数字示波器一台，测试点在 8051 的 P3.2 脚上。

程序操作说明

本程序将 8051 的 Timer 0 设置成模式 1 下的定时功能，每 10 ms 的时间下变换 P3.2 的状态一次，然后用同步示波器观察此段时间是否正确。由于 P1.0 的状态必须每隔 10 ms 变化，所以程序必须重复对 Timer 0 做载入的操作。

```
;PROGRAM NAME:T2.ASM
;TEST TIMER0 MODE 1:16 BIT COUNT UP
;
COUNT    EQU     9217
CHECK_BIT REG    P3.2
;
         ORG     0
         MOV     R0,#00H
         DJNZ    R0,$            ;WAIT
         CLR     CHECK_BIT
         MOV     SP, #40H
;
         MOV     A,TMOD
         ANL     A,#11110000B
         CLR     ACC.3           ;GATE=0
         CLR     ACC.2           ;C/T=0
         CLR     ACC.1           ;TIMER0 M1=0
         SETB    ACC.0           ;TIMER0 M0=1
         MOV     TMOD,A
         ACALL   CNT_RELOAD
;
LOOP     JNB     TF0, LOOP
```

```
;TF0=1
OVER    CLR     TR0
        ACALL   CNT_RELOAD
        CPL     CHECK_BIT       ;P3.2=1
        AJMP    LOOP
;
CNT_RELOAD
        MOV     TH0, #(65536-COUNT)/256
        MOV     TL0, #(65536-COUNT).MOD.256
        CLR     TF0             ;CLEAR TF0
        SETB    TR0             ;SET TIMER0 START TO COUNT
        RET
;
```

机器周期计算：

1/11059200×12=1.085μs

10 ms 的计数值=10 ms/1.085μs=9216.58（取整数 9217）

程序执行后，我们预期应该可以在示波器上看到一个相当稳定的方波，低电位的时间约是 10.0 ms，高电位的时间也是 10.0 ms，可是，从结果看来只对了一半。低电位的操作是对的，可是预期高电位时，看到的是一连串的脉冲信号，请看图 10-8 更详细的图示。我们进一步检查线路，发现 P3.2 也是温度感测 IC 的输出，若把 JP2 跳线去掉，就可以看到非常漂亮的 10 ms 方波了。

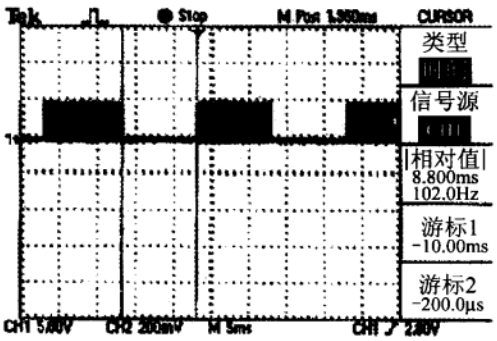


图 10-8　MODE 1 所看到的信号,时间间隔是对的，但是高电位的状态却不是可以预期的

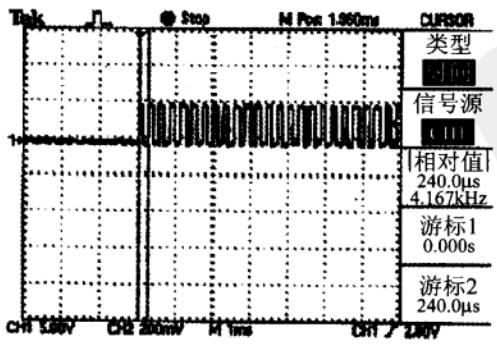


图 10-9　把图 10-7 信号再做放大，竟然在 HI 时伴随有 4.1kHz 左右的脉冲出现

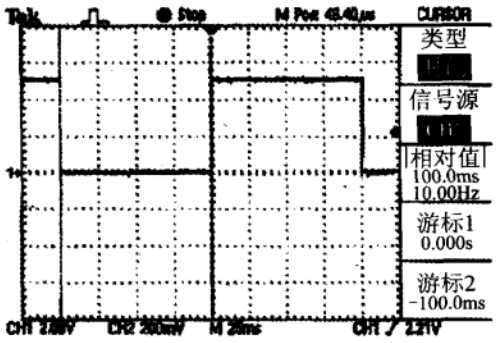


图 10-10　把连接感测 IC 的跳线移开后，就可以看到漂亮的 100 ms 波形了

目的：试验摸式 1 下 100μs 的方波信号输出。

程序名称：CH10_T3.ASM。

辅助硬件

数字式示波器，最好附有测量时间差的指针（Cursor）功能。

程序操作说明

执行后的结果是输出脉冲每 107μs 才变化一次，这也就是说，8051 执行时也是需要时间的，多了 7μs 的时间，相当于多做了好几个指令似的。我们把控制 P3.2 操作的指令往前移，或许可以把时间误差再降一些。

```
;PROGRAM NAME:T3.ASM
;TEST TIMER0 MODE 1:16 BIT COUNT UP
;
COUNT    EQU      92
CHECK_BIT REG     P3.2
;
         ORG      0
         MOV      R0, #00H
         DJNZ     R0, $            ;WAIT
         CLR      CHECK_BIT
         MOV      SP, #40H
;
         MOV      A, TMOD
         ANL      A, #11110000B
         CLR      ACC.3            ;GATE=0
         CLR      ACC.2            ;C/T=0
         CLR      ACC.1            ;TIMER0 M1=0
         SETB     ACC.0            ;TIMER0 M0=1
         MOV      TMOD, A
         ACALL    CNT_RELOAD
;
LOOP     JNB      TF0, LOOP
;TF0=1
OVER     ACALL    CNT_RELOAD
         CPL      CHECK_BIT        ;P3.2 COMPLENENT
         AJMP     LOOP
;
```

```
CNT_RELOAD
        MOV     TH0, #(65536-COUNT)/256
        MOV     TL0, #(65536-COUNT).MOD.256
        CLR     TF0             ;CLEAR TF0
        SETB    TR0             ;SET TIMER0 START TO COUNT
        RET
;
```

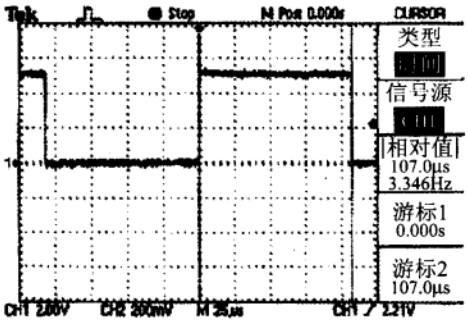


图 10-11　设置 100μs 信号输出，结果是 107μs 的信号，这种问题通常要靠程序来修正

目的：试验模式 1 下 100μs 定时的精度，并试图修正其输出的精度。

程序名称：CH10_T4.ASM。

辅助硬件

数字式示波器或逻辑分析仪。

程序操作说明

从执行后得知误差值还是有 6μs 之多，除非把 COUNT 值再减少一些，由 92 降到 87 时，输出的误差只有 1μs 左右。定时时间愈长时（>10 ms），程序处理判断的时间差是可以忽略不计的，若定时时间小于 100μs 时，程序及指令所花的时间对定时的精度有绝对的影响。

```
;PROGRAM NAME:T4.ASM
;TEST TIMER0 MODE 1:16 BIT COUNT UP
;
COUNT   EQU         87
CHECK_BIT REG       P3.2
;
        ORG         0
        MOV         R0, #00H
        DJNZ        R0, $           ;WAIT
        CLR         CHECK_BIT
        MOV         SP, #40H
;
        MOV         A, TMOD
        ANL         A, #11110000B
        CLR         ACC.3           ;GATE=0
        CLR         ACC.2           ;C/T=0
        CLR         ACC.1           ;TIMER0 M1=0
        SETB        ACC.0           ;TIMER0 M0=1
        MOV         TMOD, A
        MOV         TH0, #(65536-COUNT)/256
```

```
        MOV         TH0, #(65536-COUNT).MOD.256
        CLR         TF0             ;CLEAR TF0
        SETB        TR0
;
LOOP    JNB         TF0, LOOP       ;TF0=0
;TF0=1
OVER    CPL         CHECK_BIT       ;P3.2 COMPLENENT
        MOV         TL0, #(65536-COUNT).MOD.256
        MOV         TH0, #(65536-COUNT)/256
        CLR         TF0             ;CLEAR TF0
        SJMP        LOOP
;
```

目的：学习模式 1 下 100 ms 定时器的写法。

程序名称：CH10_T5.ASM。

辅助硬件

数字式示波器，测量 P3.2 输出的脉冲。

程序操作说明

若在程序中想让 P3．2 每隔 100 ms 变化一次，最好的方法就是直接设置定时器的载入值，不过，8051 的 Timer 在模式 1 下最大值只有 65.536 ms，显然不合我们的需要，变通的方法是先让定时精度设置在 10 ms，并且在程序中另外指定一个字节当成软件计数器，如果该计数值到达 10 时，即变化 P3.2 的状态一次，类似这样的规划安排可以让定时器做更长时间的定时。

```
;PROGRAM NAME:T5.ASM
;TEST TIMER0 MODE 1:16 BIT COUNT UP
;
MS_100      EQU     30H
COUNT       EQU     9217            ;10ms FOR XTAL=11.0592MHz
CHECK_BIT REG       P3.2
;
            ORG     0
            MOV     R0, #00H
            DJNZ    R0, $           ;WAIT
            CLR     CHECK_BIT
            MOV     SP, #40H
            MOV     MS_100, #00H    ;CLEAR(MS_100)
;
            MOV     A, TMOD
            ANL     A, #11110000B
            CLR     ACC.3           ;GATE=0
            CLR     ACC.2           ;C/T=0
            CLR     ACC.1           ;TIMER0 M1=0
            SETB    ACC.0           ;TIMER0 M0=1
            MOV     TMOD, A
            MOV     TH0, #(65536-COUNT)/256
            MOV     TL0, #(65536-COUNT).MOD.256
            CLR     TF0             ;CLEAR TF0
            SETB    TR0
;
```

```
LOOP        JNB     TF0, LOOP       ;TF0=0
;TF0=1
OVER        CLR     TF0             ;CLEAR TF0
            MOV     TH0, #(65536-COUNT)/256
            MOV     TL0, #(65536-COUNT).MOD.256
            INC     MS_100          ;INCREMENT COUNT
            MOV     A, MS_100
            CJNE    A, #10, $NEXT
            CPL     CHECK_BIT       ;P3.2 COMPLENENT
            MOV     MS_100, #00H    ;CLEAR COUNT
$NEXT       SJMP    LOOP
;
```

目的：学习 Timer 模式 1 下 LED 定时闪烁的写法。

程序名称：T6.ASM。

辅助硬件

数字式示波器，测量 P3.2 输出的脉冲与 LED 闪烁观察。

程序操作说明

程序中定义一个 10 ms 的地址，每经过 10 ms 时该地址就增加 1，直到加到 100 为止。程序另外判断 MS_10 这个地址的内容，若小于 10 时点亮 LED 灯，反之介于 11～100 时熄灭 LED。当我们观察 LED 时，就会发现 LED 每秒闪烁一次，每次 ON 的时间要比 OFF 的时间差很多。

```
;PROGRAM NAME:T6.ASM
;TEST TIMER0 MODE 1:16 BIT COUNT UP
;
MS_10       EQU     30H
COUNT       EQU     9217            ;10ms FOR XTAL=11.0592MHz
CHECK_BIT   REG     P3.2
LED         REG     P3.7
;
            ORG     0
            MOV     R0, #00H
            DJNZ    R0, $           ;WAIT
            CLR     CHECK_BIT
            MOV     SP, #40H
            MOV     MS_10, #00H     ;CLEAR(MS_10)
;
            MOV     A, TMOD
            ANL     A, #11110000B
            CLR     ACC.3           ;GATE=0
            CLR     ACC.2           ;C/T=0
            CLR     ACC.1           ;TIMER0 M1=0
            SETB    ACC.0           ;TIMER0 M0=1
            MOV     TMOD, A
            MOV     TH0, #(65536-COUNT)/256
            MOV     TL0, #(65536-COUNT).MOD.256
            CLR     TF0             ;CLEAR TF0
            SETB    TR0
;
LOOP        JNB     TF0, LOOP       ;TF0=0
;TF0=1
OVER        CLR     TF0             ;CLEAR TF0
```

```
                MOV     TH0, #(65536-COUNT)/256
                MOV     TL0, #(65536-COUNT).MOD.256
                INC     MS_10           ;INCREMENT COUNT
                MOV     A, MS_10
                CJNE    A, #10, $TEST
$TEST           JNC     $LEDOFF
                SETB    LED             ;LED ON
                SJMP    $CONT
$LEDOFF         CLR     LED             ;LED OFF
;
$CONT           MOV     A, MS_10
                CJNE    A, #100, $NEXT
                CPL     CHECK_BIT       ;P3.2 COMPLENENT
                MOV     MS_10, #00H     ;CLEAR COUNT
$NEXT           SJMP    LOOP
;
```

10-7 Timer 模式 2 彻底研究

在 8051 Timer 的模式 2，计数器宽度仅有 8 位宽度，最大计数值只到 256 而已，但是有自动再载入的功能，当此模式一经设置后，计数操作由 TLx 负责，而 THx 不负责计数但储存重新载入的定时值（8 位），当 TLx 计数到最大计数值而产生溢位时，除了 TFx 会被设成 1 外，CPU 内部自动将 THx 上的值载入 TLx 内部，继续让 TLx 计数。多了这项功能会使得利用程序载入的时间减少了，也让定时或计数的时间误差降到最低。

以下这个程序控制 P3.2 位，每隔 100μs 产生一次状态上的变化，由于模式 2 下的定时操作，最大的时间间隔只到 256μs 左右（以系统频率是 12MHz 为例），若需要更长的时间间隔时，必须另外再加 1 个字节做软件计数器，其写法和上一个程序范例类似。

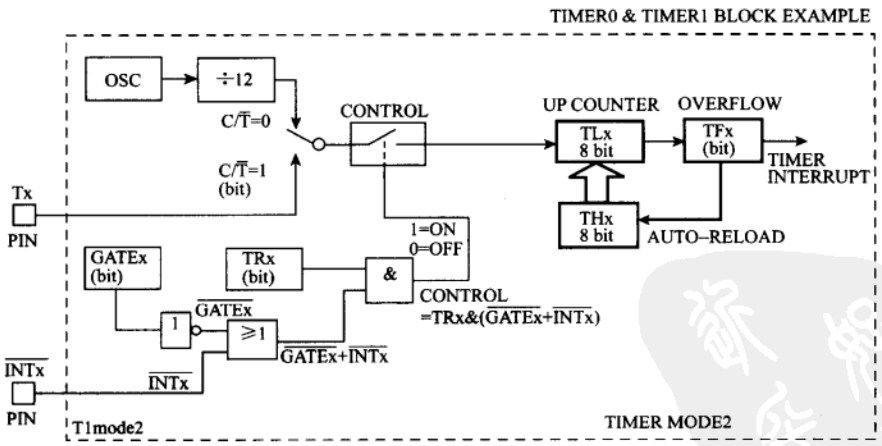


图 10-12 MODE2 的工作模式

目的：学习模式 2 下 100μs 定时器的写法。

程序名称：CH10_T7.ASM。

辅助硬件：数字式示波器，测量 P3.2 输出的脉冲。

程序操作说明

这个程序将 Timer 0 设成 MODE 2 的定时操作，并且只作一次计数值载入的操作，当 TF0 标志位溢位时，操作简化成只将 P1.0 状态改变以及清除 TF0 标志位两项而已。利用示波器观察 P3.2 的状态时，确实时间间隔为 99.0μs，比起先前模式 1 的方法误差小了许多，而且程序也较为精简。

```
;PROGRAM NAME:T7.ASM
;TEST TIMER0 MODE 2:8-BIT COUNT UP
;
COUNT       EQU     92              ;100μs FOR XTAL= 11.0592MHz
CHK_BIT     REG     P3.2
LED         REG     P3.7
;
            ORG     0
            MOV     R0, #00H
            DJN2    R0, $           ;WAIT
            CLR     CHK_BIT
            MOV     SP, #40H
;
            MOV     A, TMOD
            ANL     A, #11110000B
            CLR     ACC.3           ;GATE=0
            CLR     ACC.2           ;C/T=0
            SETB    ACC.1           ;TIMER0 M1=1
            CLR     ACC.0           ;TIMER0 M0=0
            MOV     TMOD, A
            MOV     TH0, #(256-COUNT)
            MOV     TL0, #00
            CLR     TF0             ;CLEAR TF0
            SETB    TR0
;
LOOP        JNB     TF0, LOOP       ;TF0=0
;TF0=1
OVER        CLR     TF0             ;CLEAR TF0
            CPL     CHK_BIT
$NEXT       SJMP    LOOP
;
```

10-8 Timer 模式 3 彻底研究

8051 的模式 3 和前面所谈的 3 种模式都不一样，首先，Timer0 被分割成两个 8 位定时计数器，并且占用原本是 Timer 1 所使用的 TR1 和 TF1 标志位，而 Timer 1 仍可在模式 0、1 和 2 下工作，但少了 TR1 和 TF1 溢位标志位，若将 Timer1 设成模式 3 时，其效果相当于 CLR TR1 指令一样，停止 Timer1 的 Count 功能。缺少了 TF1 亦可以程序检查的方式，随时读取 TH1 和 TL1 的值，以便确认是否有溢位的情形发生。

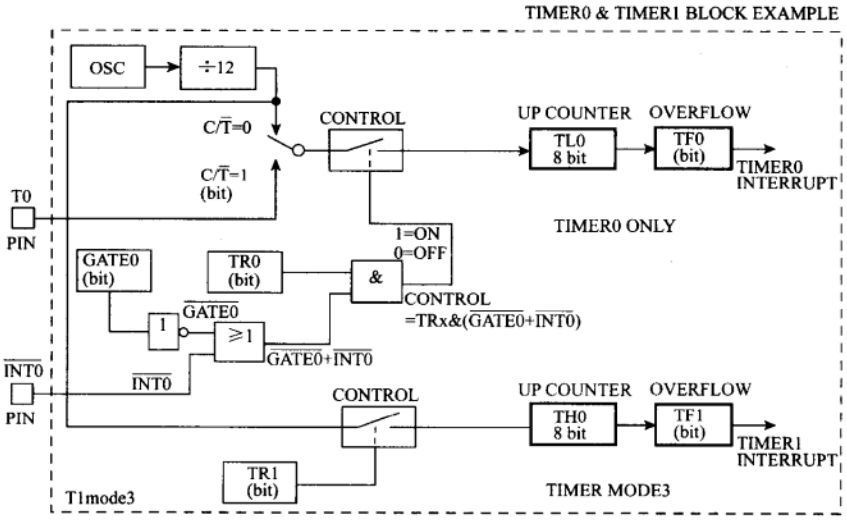


图 10-13　Timer0 在模式 3 下的工作模式图

特别说明：

（1）TH0 和 TL0 都没有计数值自动载入的功能。

（2）图中只表示出 Timer0 的功能，当 Timer1 被设成 MODE 3 时，相当于停止 Timer1 的计数功能。

（3）Timer1 仍可在非模式 3 下工作，但已失去 TR1 和 TF1 两个重要的位。

当 8051 单片机其 Timer0 被假设成模式 3 时，内部总共有 3 个计数/定时器在运行，但 Timer1 此时只有 3 种模式可供选择，这种方式的安排似乎是 Intel 当初设计 8051 单片机时刻意促成的，因为单片机另一个特点是强调其串行通信能力，若想做串行 RS232 方面的通信时，不可避免地要使用波特率发生器（Baud Rate Generator），当系统加入波特率发生器时会占用一组定时计数器，这使得系统只剩一组 Timer 可用，一组 Timer 对单片机系统而言是少了一点，由于波特率发生器必须由一组可重复载入的 8 位计数器做成，不需要中断和允许计数的控制位。所以 Intel 的手册中建议我们若想做串行传输通信时，将 Timer 1 设成模式 2 可自动载入的定时器，然后 Intel 工程师将 CPU 内部稍做修正，我们仍能分别使用两个 8 位定时/计数器做其他的用途，但是此时的两个 8 位 Timer 都没有重复载入的功能，并且 TH0 仅有定时的功能而已，请仔细对照文中的说明和图 10-13，以便对模式 3 有一个全盘性的理解。

10-9　8051 Timer 模式 3 的再探讨

8051 单片机若用到串行通信时，就一定占用了 Timer 1，并且设置成 Mode2 自动载入的模式，以达成其波特率发生器的各项功能要求，这时系统只剩下 Timer 0 可以使用。而 Timer0 可以做 4 种模式的选择，其中模式 0～2 都同时占用 TH0 和 TL0，只有在模式 3 下才可以把 Timer0 中的 TH0 和 TL0 分成两个独立的计数器来使用，又因为波特率发生器必须不断地计数也不需要对 CPU 中断，所以设计 8051 的 Intel 工程师才将 TR1 和 TF1 挪给 TH0 来使用，但是又因为 Timer1 仍可能使用 GATE 位或 T1、INT1 等硬件控制脚，所以此时的 TH0 仅能

供做定时的功能，而没办法有计数的功能要求了，我们最好把 TH0 应用为定时中断，通过固定时间的中断信号，要求 CPU 对某些状态或周边 IC 做检查。

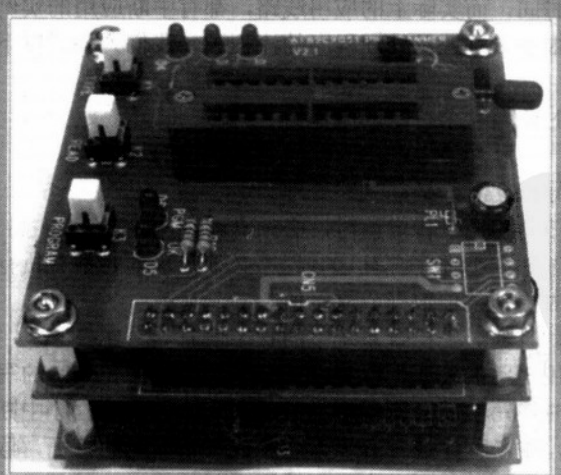
我们认为 Intel 对 MODE 3 的安排乃是权宜之计，串行传输时 Timer2 就会完全被占用，另外 TL0 能做 8 位的定时/计数器，TH0 仅能做 8 位的定时，可惜的是 TL0 和 TH0 都没有自动载入的功能，除非使用中断要求的方式，否则都会有定时误差存在。

由于 8051 在 Timer 上的各种缺点确实存在，造成不少设计工程师诸多困扰，才使得 Intel 公司在接着推出 8052 系列的单片机，其中加强最多的部分正是 Timer 这一部分，其中的 Timer2 包括了定时/计数器可自动载入及可捕获等功能的增加，关于这一部分的技巧及使用说明，将在本系列书籍的实习篇再详尽叙述。

习　题

1. 什么是定时/计数？
2. 8052 比 8051 多了一个定时/计数器，它是由哪个寄存器来控制的？
3. 8051 的定时/计数器是由哪些寄存器来控制？
4. 承上题，该寄存器可否用来位寻址？
5. TCON 的四大功能是什么？
6. 如何设置 8051 的定时/计数器？
7. 当 Timer 被设置为 MODE0 工作时，计数器的宽度有几位？
8. 当 Timer 被设置为 MODE2 工作时，计数器的最大计数值是多少？

11



USB 烧录器可以独立运行，只要从 USB 获得 +5V 电源，即可执行 Read 与 Program 的操作，一直到烧录完后

第 11 章　8051 中断彻底研究

8051 的中断研究也是颇具挑战性的，中断服务程序处理不当就会死机、中断的时间不能太久、程序不能进入死循环，这些是编写中断服务程序一定要建立的观念。学习 8051 的汇编语言而不懂得使用中断服务程序，那就不算精通了。

11-1　为何要有中断

有人说：学会写中断程序，才算是对程序及单片机有充分的了解，这话说得没错，因为在学习中断程序的处理过程中，我们才得以学到 CPU 的部分精髓。其实，我们操作的 PC 机就是大量使用中断（Interrupt）的一个好例子，只要我们开机后，每秒钟就有数十个中断信号源源不断地传送到 CPU 上，要求 CPU 做服务（Service），只有妥善且正确地处理这些要求（Request），才能使 PC 正常运行而不会死机。

11-2　8051 的中断

在学习 8051 单片机的过程中，对中断程序越了解，则越能有效地控制程序，并且可以减少程序开发及除错的时间，而实际上的应用线路，都有中断应用的实例，若我们对中断的步骤和原理不熟悉，就很难去掌握及处理程序的流程了。我们列举了以下几种可采用中断的时机：

（1）当计数值数完时要做一个特定的操作。
（2）当外部信号有一个脉冲信号产生时要做某种特定的运算。
（3）当外部某些信号成立时，必须立即处理。
（4）当收到通信接口上的一个特定值时，必须立即反应。
（5）当程序必须处理数个小程序，且这些程序必须几乎同时都在执行中。
（6）当程序必须随时更新某项数据或显示值。
（7）当程序必须自动且随时去检查系统中的状态值时。

上述的情形也可采用平常的程序，以持续或经常询问（Polling）的写法来应付，但是若系统有多个条件要询问时，写法就会变得相当的复杂。假如改用程序中断的写法，则程序的写法就会变得较简单些，中断程序通常有以下一个或一个以上的特征，假如我们的应用中发现有这些特征时，在程序规划前就要考虑是否采用中断的写法。

（1）每隔一个时间间隔就必须做某件特定的事情。
（2）当某个条件（不论软件或硬件线路）成立时，就让 CPU 立即处理。
（3）CPU 正以多任务（Multi-task）的模式，同时处理数个程序或信号。

（4）某项状态可能平时均不成立，可是一成立时，CPU 必须立即停止原先的操作，马上来处理这个状态，这种状态就类似“一病就要人命”的场合，若不即时处理系统恐将会不保。

MCS-5l 系列单片机中分别提供 5 个中断源，这包括了两个外部中断，两个定时/计数中断及一个串行中断，而 8052 则多了一个定时/计数中断。接下来的章节，我们将详细说明如何利用这 5 个中断源去触发中断，以及中断程序的设置及处理。

表 11-1　中断源和中断矢量地址的对照表

矢 量 地 址	中断源正式名称
0003H	外部 INT0 中断
000BH	内部定时/计数器 0 中断
0013H	外部 INTl 中断
001BH	内部定时/计数器 1 中断
0023H	串行传输中断
002BH	内部定时/计数器 2 中断（8052/8032 才有）

11-3　中断时软件的操作剖析

当 8051 单片机被允许中断后，当刚好有中断信号产生时，CPU 会依据中断信号源的种类，固定地跳到某几个中断矢量地址去执行所谓的中断服务例程 ISR（Interrupt Service Routine），这些中断服务例程的进入点分别是：

```
外部中断(IE0)              :进入点 0003H
定时/计数器 1(TF0)         :进入点 000BH
外部中断 2(IE1)            :进入点 001BH
定时/计数器 2(TE1)         :进入点 001BH
串行传输(RI+TI)            :进入点 0023H('+'表逻辑上的或 OR)
定时/计数器 2(TF2+EXF2)    :进入点 0023BH(仅 8052 提供此中断)
```

在 CPU 尚未跳进以上进入点前，会先将此时系统的程序计数值 PC（Program Counter）暂时存放在内部 Data Memory 的堆栈区中，然后才产生一个 LCALL 远程调用的方式，跳进上述的进入点地址，此时中断程序的写法和平常的程序写法类似，当中断服务程序执行完后，最后通过 RETI 指令回到原程序的中断离开点，继续执行原程序。

中断服务程序一定要以 RETI 作为结束，此时 CPU 会取回原先放在堆栈区中的 PC 值，同时亦告知 CPU 中断服务程序已结束，回原程序后可以再做下次中断，若只使用 RET 而结束 ISR 例程时，下个中断信号产生时，CPU 将不做任何反应，这也就是说系统仅中断一次而已，很显然程序有错误存在。

11-4 中断时的硬件操作剖析

依据 8051 手册上的写法，CPU 会在机器周期的 S5P2 阶段读入（原文为 Sampled，但翻译成“读取”似乎更为恰当）中断标志位，并在下一个机器周期中检查，如果中断条件成立时，系统会自行产生一个 LCALL 到相对应的中断服务程序中，可是如果有下面 3 种情况时，系统是不会对中断要求信号有反应的：

（1）有相等或更高权限的中断程序正在执行中，这跟我们处理突发事情的状况相同，既然已经在处理突发情况，当然就不再接受其他中断条件，除非接下来的中断情形的优先权比较高，否则免谈。

由此我们得到一个观点：所有的中断程序都应该尽量简捷，一处理完中断事项后立即回主程序，才不会占用过多时间，进而影响系统的性能（Performonce）。

（2）目前的机器周期不是该指令的最后一个周期，由于 8051 在指令执行时，分别有 1、2 和 4 个机器周期之分，这也就是说，必须完全执行完此指令后，系统对中断信号才会有所反应。比如说，当系统正在执行 MUL AB 指令（需花 4 个机器周期）时，中断信号必须出现在第四个机器周期上才算有效。这也就意味着，中断信号必须持续够长的时间，以便 8051 CPU 有时间去反应。

（3）若正在执行的指令为 RETI 或者是关于中断设置 IE 或 IP 的指令时，对正好出现的中断信号不反应，因为上述的情况刚好是某个中断服务程序的结束，或是允许/禁止某个中断的指令，当然是等到这些指令执行完毕后，才会对中断信号有所反应，这些指令最多占用两个机器周期的时间，所以这时的中断信号必须保持有两个机器周期以上的时间，才能被 8051 接受。

8051 接受中断信号后，会把现在的 PC 值传入（PUSH）堆栈中，然后由硬件产生一个 LCALL 远程调用，直接到该中断矢量的进入点上，对 CPU 的其他寄存器不做任何处置，甚至连程序标志位（PSW）也都不自动存入堆栈中，当中断程序中会改变某个寄存器或 PSW 值时，我们必须在中断程序正式执行前先将这些值暂时保存在堆栈中（或其他地址上），等到 ISR 执行完毕后，取回这些重要的寄存器值，然后再执行 RETI 回原程序的中断点。

以下是一个简单的中断程序示范，图 11-1 中所取到的数据完全来自 HP 的逻辑分析仪，每两行代表一个机器周期，示范程序将在稍后的章节中详述，此时我们仅列出 ISR 中断服务例程的内容。这个示范程序平常一直在地址 004 BH 与 004CH 内的无穷循环内，当定时中断发生时，CPU 会先跳到 000BH 的定时中断点，然后再跳到用户指定的地址，进行中断服务程序的处理，最后当然以 RETI 指令代表 ISR 的结束。

8051 使用外部程序存储器时，才能看到类似前面的数字状态值显示。如果你用的是内置 Flash 程序存储器的 AT89C51，就无法如此方便地看到这些时序变化了，对于 CPU 如何反应中断就只有凭空想象了。

```
004B    00      LOOP    NOP
004C    80FD            SJMP    LOOP
                ;
                ;TF0=1
```

```
004E            INT_TIMER0
004E    75 8C DB        MOV     TH0,#(65536-COUNT)/256

0051    75 8A FF        MOV     TL0,#(65536-COUNT).MOD.256
0054    05 20           INC     20H
0056    85 20 90        MOV     P1,20H
0059    32              RETI
```

Analyzer | Listing MCS_51 | Print | Run

Markers Off

Acquisition Time
15 Aug 2002 22:20:01

Label> Base>	ADDR Hex	DATA Hex	Intel MCS-51 Inverse Assembler mnemonics		STAT Symbol
-5	004C	80	SJMP 004BH	opcode fetch	OPCODE FETCH
-4	004D	FD		operand	OPERAND
-3	004D	FD		prefetch	PREFECH
-2	004E	75		prefetch	PREFECH
-1	004B	00	NOP	opcode fetch	OPCODE FETCH
0	004B	00		***interrupt***	INTERRUPT
1	004B	00		***interrupt***	INTERRUPT
2	004B	00		***interrupt***	INTERRUPT
3	000B	02	LJMP 004EH	opcode fetch	OPCODE FETCH
4	000C	00		operand	OPERAND
5	000D	4E		operand	OPERAND
6	000D	4E		prefetch	PREFECH
7	004E	75	MOV TH0,#DBH	opcode fetch	OPCODE FETCH
8	004F	8C		operand	OPERAND
9	0050	DB		operand	OPERAND
10	0050	DB		prefetch	PREFECH
11	0051	75	MOV TL0,#FFH	opcode fetch	OPCODE FETCH
12	0052	8A		operand	OPERAND
13	0053	FF		operand	OPERAND
14	0053	FF		prefetch	PREFECH
15	0054	05	INC 20H	opcode fetch	OPCODE FETCH
16	0055	20		operand	OPERAND
17	0056	85	MOV P1 ,20H	opcode fetch	OPCODE FETCH
18	0057	20		operand	OPERAND
19	0058	90		operand	OPERAND
20	0058	90		prefetch	PREFECH
21	0059	32	RETI	opcode fetch	OPCODE FETCH
22	005A	F0		prefetch	PREFECH
23	005A	F0		prefetch	PREFECH
24	005A	F0		prefetch	PREFECH
25	004B	00	NOP	opcode fetch	OPCODE FETCH
26	004C	80		prefetch	OPERAND

图 11-1　中断 ISR 执行时的状态分析

注：(1) 在 NOP 指令执行后，定时中断要求服务，于是 CPU 先将地址指向 000BH。

(2) 通常我们在 000BH 上会摆一个 LJMP 的指令，由图中看到是跳到 004EH。

(3) 位址 004EH 上对 TIMER0 填入 16 位的值，并且把 20H 内容加 1。

(4) 把 20H 内容送到 P1 端口上。

(5) RETI 后重新返回主程序 004BH 的 NOP 指令处。

Label> Base>	ADDR Hex	Intel MCS-51 Inverse Assembler mnemonics		Time Absolute	PORT 1 Hex
-5	004C	SJMP 004BH	opcode fetch	-2.712 µs	A8
-4	004D		operand	-2.168 µs	A8
-3	004D		prefetch	-1.624 µs	A8
-2	004E		prefetch	-1.088 µs	A8
-1	004B	NOP	opcode fetch	-544 ns	A8
0	004B		***interrupt***	0 s	A8
1	004B		***interrupt***	544 ns	A8
2	004B		***interrupt***	1.088 µs	A8
3	000B	LJMP 004EH	opcode fetch	1.632 µs	A8
4	000C		operand	2.168 µs	A8
5	000D		operand	2.712 µs	A8
6	000D		prefetch	3.256 µs	A8
7	004E	MOV TH0,#DBH	opcode fetch	3.800 µs	A8
8	004F		operand	4.344 µs	A8
9	0050		operand	4.888 µs	A8
10	0050		prefetch	5.424 µs	A8
11	0051	MOV TL0,#FFH	opcode fetch	5.968 µs	A8
12	0052		operand	6.512 µs	A8
13	0053		operand	7.056 µs	A8
14	0053		prefetch	7.600 µs	A8
15	0054	INC 20H	opcode fetch	8.136 µs	A8
16	0055		operand	8.680 µs	A8
17	0056	MOV P1 ,20H	opcode fetch	9.224 µs	A8
18	0057		operand	9.768 µs	A8
19	0058		operand	10.31 µs	A8
20	0058		prefetch	10.85 µs	A8
21	0059	RETI	opcode fetch	11.39 µs	A8
22	005A		prefetch	11.94 µs	A9
23	005A		prefetch	12.48 µs	A9
24	005A		prefetch	13.02 µs	A9
25	004B	NOP	opcode fetch	13.57 µs	A9
26	004C		prefetch	14.10 µs	A9

图 11-2　中断 ISR 状态分析加注时间

注：（1）本例子是设置定时/计数器 0 中断，所以中断的默认进入点在 000BH。

（2）TIMER0 的中断请求应该在触发后经过 1.088µs 被确认。

（3）CPU 开始执行中断 ISR 程序时已经过了 3.800µs，将近执行 3.5 个指令的时间。

（4）当中断信号被确认，所以 CPU 会自动做两个操作：一是把 PC 值存入 STACK 中，二是把 PC 值改成 000BH。

（5）CPU 在中断确认及 PC 值调整期间，硬件的 ALE 和 PSEN 信号还是持续不断，CPU 仍然传送出 4 次地址值到地址总线上，但读入的值未使用就舍弃掉。

11-5　中断的寄存器（IE 和 IP）的介绍

8051 单片机在加上电源后，默认是禁止任何中断信号的，这是为了防止中断信号随意产生而干扰了系统的运行和起始设置，当系统程序做好了各项设置程序后，才可以允许中断信号的触发，进而执行对应的 ISR 程序，想要允许（Enable）或禁止（Disable）8051 的中断，就必须涉及到两个关于中断设置的寄存器 IE 和 IP。

IE 寄存器（见图 11-3）占用 A8H 地址，而且可做位寻址，即可对该寄存器的任一个位做设置（Set）或清除（Reset 或 Clear）操作，IE 寄存器中除了 IE.6 原厂保留不用外，其他 7 个位都和中断的 Enable 和 Disable 有关。

(MSB)							(LSB)
EA	X	ET2	ES	ET1	EX1	ET0	ET0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

图 11-3　IE 寄存器

EA 或称 IE.7：这个位是 IE 寄存器中最重要的位，若 EA=0 则禁止系统中所有的中断要求，所以 8051 在系统重置后，一定要把此位清除为 0，假设 EA=1 时代表允许 CPU 接受中断，至于接受哪个中断要求，则视其他 6 个位的设置值而定。

ET2 或称 IE.5：决定是否允许定时/计数器 2 的中断，0 代表 Disable，1 代表 Enable。

ES 或称 IE.4：决定是否允许串行传输端口的中断请求，0 代表 Disable，1 代表 Enable。

ETl 或称 IE.3：决定是否允许定时/计数器 1 的中断，0 代表 Disable，1 代表 Enable。

EXl 或称 IE.2：决定是否允许外部的 INTl 中断要求，0 代表 Disable，1 代表 Enable。

ET0 或称 IE.1：决定是否允许定时/计数器 0 的中断要求，0 代表 Disable，1 代表 Enable。

EX0 或称 IE.0：决定是否允许外部的 INT0 中断要求，0 代表 Disable，1 代表 Enable。

通常我们设置 IE 寄存器的值时，都分成两次过程来处理，首先是设置 IE.0 到 IE.5 共 6 个位的值，然后等到所有中断的准备操作都就位后，才把 IE.7（即 EA）设成 1，从此开始接受各个源源不断的中断要求。

事有分大小轻重，8051 处理中断信号亦有轻重之分，而这些区分的操作就由 IP（Interrupt Priority）中断优先顺序寄存器来决定，8051 仅将优先顺序归成两类：高优先中断（High Priority Interrupt Request）和低优先中断（Low Priority Interrupt Request）。当系统复位 Reset 后，自行将各个中断的优先顺序都设成低优先顺序中断。若系统完成优先顺序的设置且允许中断后，将依所设置的中断优先顺序来处理中断要求，CPU 依照以下法则来决定优先权：

（1）若原先没有中断要求，则高或低优先顺序的中断信号皆可对 CPU 中断，而且 CPU 一定要遵照处理。

（2）如果已经在处理低优先顺序的中断 ISR 例程，假如又有高优先顺序的中断要求产生时，CPU 会暂时放下此例程，优先处理高优先顺序的中断。

（3）如果系统正在处理高优先顺序的中断 ISR 例程，此时若还有低优先顺序的中断要求出现时，系统并不理会该中断，必须等到较高优先顺序的 ISR 例程处理完毕后，再处理较低优先顺序的中断。

（4）如果两个优先权相等的中断信号同时发生，系统会另外以表 11-2 的顺序来处理这些中断要求。表中说明 IE0 占最高优先权，而串行传输中断的优先权就较低，8051 会做如此安排，可能是考虑到外部的中断信号可能稍纵即逝，必须先处理。

表 11-2　优先权相等时 CPU 处理中断的顺序

中　断　来　源		优先权设置
IE0	外部 INT0 中断	最高优先权
TF0	定时/计数器 0 中断	
IE1	外部 INT1 中断	
TF1	定时/计数器 1 中断	
RI+TI	串行传输中断	
TF2+EXF2	定时/计数器 2 中断	最低优先权

IP（Interrupt Priority）寄存器占用 B8H 地址，也可以对每一个位做位状态设置，由于 8051 仅有 5 个中断来源，所以 IP 寄存器中将有 3 个位未使用，而 8052 的 IP 寄存器中则有 2 个位未使用。

PT2 或称 IP.5：定义定时/计数器 2 的中断优先权层次，1 代表较高优先权，0 代表较低优先权。

(MSB)							(LSB)
X	X	PT2	PS	PT1	PX1	PT0	PX0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

图 11-4　IP 寄存器

PS 或称 IP.4：定义串行传输的中断优先权层次。
PT1 或称 IP.3：定义定时/计数器 1 的中断优先权层次。
PX1 或称 IP.2：定义外部中断 INT1 的中断优先权层次。
PT0 或称 IP.1：定义定时/计数器 0 的中断优先权层次。
PX0 或称 IP.0：定义外部中断 INT0 的中断优先权层次。

11-6　8051 的中断源彻底研究

在此之前我们已经谈到中断的目的和操作，还有关于中断设置的 IE 与 IP 两个寄存器，对中断的整个运行也有一些了解，但是了解以上知识，还无法有效且正确地控制 8051 的中断，必须彻底了解各个中断源（Interrupt Source）的设置，才算是对 8051 的中断过程入了门。

（1）外部中断源 0（即 INT0）：若系统外部想要中断 8051 单片机，可由 8051 的第 12 引脚送入中断信号，此时若 CPU 也同时允许该中断发生时，就会将 PC 值改成 0003H，开始执行 ISR 例程。由于外部信号可能会因其他状况而有所变化，所以 8051 可定义 INT0 的触发方式，一是低电平时触发中断，即只要 INT0 为低电平时就要求中断，另一种方式是负边沿触发，这代表 INT0 的状态由高变为低电位时要求中断，这个触发条件并不是在 IE 或 IP 寄存器上设定，而是在 TCON 计时计数控制寄存器的 IT0 上设置，如果 IT0=0 表示低电平触发，IT0=1 时为负边沿触发，当 INT0 的触发条件成立时，会将 TCON 寄存器中的 IE0 位设成 1，若 CPU 开始执行 ISR 程序时，IE0 会自动被清除为 0。

（2）定时/计数器 0 中断源：当 8051 内部的定时/计数器 0 开始往上计数，数到溢位时，就会对 CPU 产生中断要求，此时 TCON 寄存器中的 TF0 会被设成 1，若该中断被接受后则跳到 000BH 去执行 ISR 程序，并且 TF0 位会被自动清除为 0。

（3）外部中断源 1（即 INT1）：本中断源的定义和条件均与 INT0 类似，在 TCON 寄存器中的 IT1 定义此中断源是低电平中断或是负沿中断，当中断条件成立时，会将 TCON 中的 IE1 位设置成 1，若 CPU 有执行 0013H 开始的 ISR 例程时，IE1 会自动被清除成 0。

（4）定时/计数器 1 中断源：当定时/计数器 1 被允许计数，且递增至溢位时就会产生此中断且执行 001BH 开始的 ISR 例程时，TF1 会自动被清除成 0。

（5）串行传输中断源：本中断源其实含两个中断条件，只要串行传输寄存器 SBUF 已收妥一个值，或者 SBUF 已传送完一个值都可以产生中断要求，前一个条件成立时，串行控制寄存器 SCON 的 RI 位会设成 1，后一个条件成立时，SCON 寄存器的 TI 位会设成 1，若中断被承认且执行 ISR 后，RI 或 TI 都不会自动被清除成 0，必须用程序对该位另行清除。

（6）定时/计数器 2 中断源：这是 8052 系列 CPU 独有的中断源，当 T2CON 寄存器中的 TF2 或 EXF2 位被设成 1 时，都可让 CPU 跳到 002BH 执行 ISR 例程，上述两个位都不会因执行 ISR 例程而自行清除之，必须另外用程序将此位设成为 0。

若将这里所谈到的中断源，再加上先前所提到的 IE 和 IP 中断有关的寄存器，我们可以得到一个在 8051 有关中断的重要图示，如图 11-5 所示。

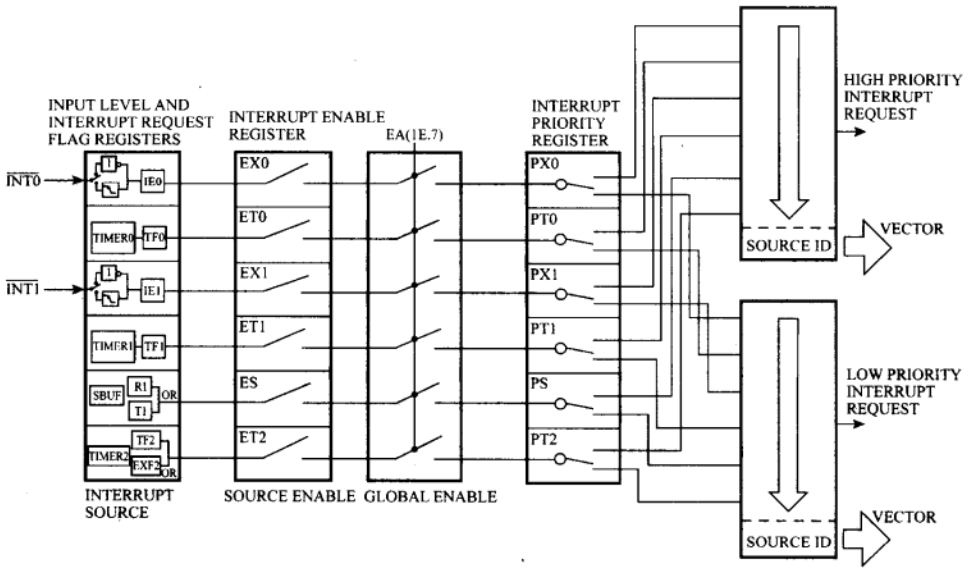


图 11-5　中断关系图

注：（1）8051 与 8051 所有中断的机制都在上图，所以请务必弄清楚。
（2）若中断源被确认中断，且执行对应的 ISR 例程时，IE0、TF0、IE1 和 TF1 会自动清除为 0，但 R1、TI、TF2 和 EXF2 位必须另外以程序清除之。
（3）上图明白地显示出 8051 有 5 个中断源，8051 则再多一个。
（4）外部中断源可以指定低电位触发或负边沿触发。
（5）每个中断都有其对应的使能位。
（6）是否允许系统中断的总管位是 EA（IE.7）。
（7）每个中断还可以指定高或低优先权，比较高的优先权可中断低优先权的 ISR。
（8）同一个时间还是只能有一个中断产生。
（9）串行中断中只要 RI 或 TI 为真就产生中断。

	(MSB)							(LSB)
IE	EA	X	ET2	ES	ET1	EX1	ET0	EX0
	(MSB)							(LSB)
IP	X	X	PT2	PS	PT1	PX1	PT0	PX0
	(MSB)							(LSB)
TCON	[TF1]	TR1	[TF0]	TR0	[IE1]	IT1	[IE0]	IT0
	(MSB)							(LSB)
T2CON	[TF2]	[EXF2]						
	(MSB)							(LSB)
SCON							[TI]	[RI]

图 11-6　与中断有关寄存器和位

注：（1）位字母代号：
T-TIMER
S-SERIAL
X-EXTERNAL
P-PRIORITY
O-TIMER0
1-TIMER1
2-TIMER2
TI-TRANSMIT INTERRUPT
RI-RECEIVE INTERRUPT
（2）有[ ]记号代表可中断 CPU。

11-7 8051 的中断设置步骤

当我们要用程序设置 8051 的中断，以及让 8051 执行 ISR 程序时，仍有一些技巧要学习，其中包括如何设置/解除中断，如何定义中断条件和正确的设置步骤等等，以下我们将说明一个中断的设置步骤是如何形成的，当然其中某些步骤是可有可无的，若我们对这些技巧愈熟悉，做实际应用时就不会有 bug 出现。中断的具体设置步骤如下：

（1）将中断服务程序中欲操作的寄存器或地址值清除或起始。若少了这个步骤时，真正中断发生时就会程序大乱。

（2）修改中断矢量地址，当中断发生时，CPU 会自动跳到各个默认的中断程序进入点上，有时候我们想暂时更改 ISR 程序的操作，可用几个跳转指令将操作暂时转向，这种作法是 PC 机对付各种接口卡中断最常用的手法，此步骤可依需要而加入。

（3）设置中断成立条件，若属于定时/计数方面的中断时，就必须对 TIMER 值做决定，若想做串行传输的中断时，就必须同时指定定时器操作及清除有关串行传输的位，这个步骤非得被执行一次不可，否则系统将不会有所中断。

（4）设置中断的优先权顺序以及指定系统可接受哪几个中断源，这个步骤实际上就是设置 SFR 特殊功能寄存器中的 IP 和 IE 值，此步骤也是不可省略的。

（5）清除中断发生的对应位，依实际情况我们必须对以下位做清除的操作：TF1、TF0、IE1、IE0、TF2、EXF2、TI 和 RI 共 8 个位，在某些情况下，这些位可能已被定为 1，会对中断程序造成错误操作，例如我们若指定定时器中断，则 TF1 和 TF0 必须在中断程序发生前被清除为 0，其他位则不必做额外的清除操作。

（6）当所有条件都就绪后，就可以开始接受中断信号源，所以最后一个步骤就将 IE.7 即 EA 位设成 1，至此之后当中断源提出中断要求时，CPU 就得去执行对应的 ISR 程序，然后才回原程序继续原先的操作。如果程序中有一段操作非常重要，必须避开任何中断源时，可以暂时将 EA 清除为 0，待操作完毕时才又把 EA 设成 1，继续接受中断源的信号。

以上 6 个重要步骤是执行中断前的设置操作，除了步骤 2 可省略外，其他 5 个步骤绝对是不可少的，而且设置的次序也不可颠倒，否则极易造成 8051 的操作错误。例如，先将 EA 设成 1 后才去设置中断源及优先顺序，您猜会有什么严重的后果呢？

我们再次把中断设置步骤归类整理如下：

（1）ISR 相关数据起始设置。

（2）修改中断矢量地址（如果可以修改的话）。

（3）设置中断成立条件。

（4）设置中断优先权及有中断的中断源。

（5）清除中断位。

（6）EA 位设成 l，系统开始允许中断。

虽然我们已经对中断程序完全理解了，但在实际应用中，仍有一些小技巧必须学习，这些 Know-How 将在稍后的实例章节中再做深入的探讨。

11-8 AT2051 控制板在中断上的安排

事实上，AT2051 的温度感测程序已经在使用定时中断与串行中断，前者对板上的七段显示器做定时的更新及扫描，后者是对外来的串行命令立即反映。当然，其他中断的使用场合也可以加入。用这块控制板来学习中断也是比较适合的。

当系统被允许 Timer0 中断而且中断源发出正确的定时中断要求后，单片机 8051 会跳到程序 ROM 的最前头，即地址为 000BH 处（Timer0 中断时），大多数的程序都在该地址上先做一个远程跳转 LJMP（Long Jump），直接跳进该 ISR 程序的进入点，这些程序操作都直接被烧录成程序 ROM，所以完全无法更改，当站在系统安全的角度上看时，这是个最稳定且正确的做法。

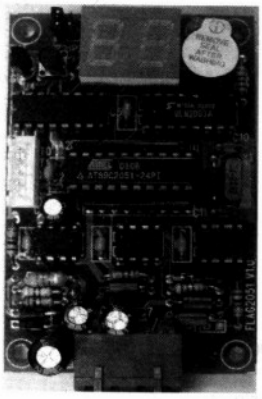


图 11-7 AT2051 控制板使用 timer 0 中断与串行中断实物图

11-9 内部计数器 0 中断程序范例

目的：了解 8051 的 Timer0 中断的程序写法。

程序名称：CH11_INTR1.ASM。

辅助硬件

数字式示波器。

```
;PROGRAM NAME:INTR1.ASM
;TEST TIMER0 INTERRUPT(MODE 1)
;
COUNT       EQU     9217        ;10ms FOR XTAL=11.0592MHz
CHK_BIT     REG     P3.2
;
            ORG     0000H
            LJMP    RESET
            ORG     0003H       ;INT0 INTERRUPT
            RETI
            ORG     000BH       ;TIMER0 INTERRUPT
            LJMP    INT_TIMER0
            ORG     0013H       ;INT1 INTERRUPT
            RETI
            ORG     001BH       ;TIMER1 INTERRUPT
            RETI
            ORG     0023H       ;SERIAL INTERRUPT
            RETI
;
RESET       MOV     R0,#00H
            DJNZ    R0,$        ;WAIT
            CLR     CHK_BIT
            MOV     SP,#40H
;
```

```
                MOV     A,TMOD
                ANL     A,#11110000B
                CLR     ACC.3           ;GATE=0
                CLR     ACC.2           ;C/T=0
                CLR     ACC.1           ;TIMER0 M1=0
                SETB    ACC.0           ;TIMER0 M0=1
                MOV     TMOD,A
                MOV     TH0,#(65536-COUNT)/256
                MOV     TL0,#(65536-COUNT).MOD.256
                CLR     TF0             ;CLEAR TF0
                SETB    TR0
;
                SETB    PT0
                SETB    ET0             ;ENABLE TIMER0 INTURRUPT
                SETB    EA              ;ENABLE SYSTEM INTERRUPT
;
LOOP            SJMP    LOOP
;
;TF0=1
INT_TIMER0
                CPL     CHK_BIT
                CLR     TF0             ;CLEAR TF0
                MOV     TH0,#(65536-COUNT)/256
                MOV     TL0,#(65536-COUNT).MOD.256
                RETI
;
;
```

程序操作说明

本程序范例是以计数器 0 为中断源，事先将计数器 0 设置 Mode1 的 16 位计数器，并对 TH0 及 TL0 均填入 00H 值，然后开始允许计数器计数及允许计数器 0 的溢位中断，每次中断发生时，代表该计数器已数了 9217 次了。由于 8051 做类似计数时，溢位产生中断后计数值无法自动载入（Autoload）重新计数值，所以在 ISR 程序里还需对 TH0 及 TL0 再做一次设置的操作。我们可以从图 11-8 看到 P3.2 输出的 10.00 ms 方波信号。

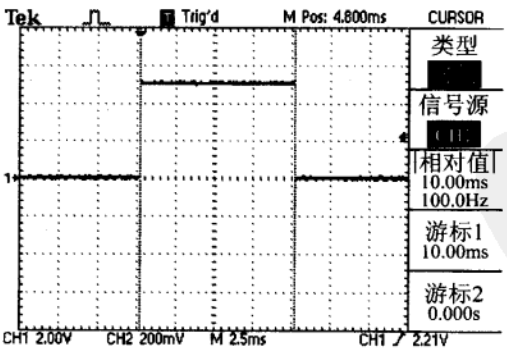


图 11-8　P3.2 的波形，时间正好是 10.00 ms 方波的信号

中断设置步骤仍以先前讲的 6 个步骤完成，欲做 TF0 中断时就必须：

（1）修改中断服务程序的进入点。

（2）设置定时中断的计数值及定时模式。

（3）允许定时。

（4）允许定时及允许定时中断。

（5）系统允许中断。

（6）ISR 程序的编写。

定时中断的写法是很重要的，几乎所有的微控器都是将此中断信号当成数据检查与更新的依据，请看下一个例子。

目的：利用 8051 的 Timer0 定时中断写七段显示器的显示程序。

程序名称：CH11_INTR2.ASM。

辅助硬件

数字式示波器。

程序操作说明

当程序执行时，我们可以看到 AT2051 上的显示器正显示着“34”的字样，3 的数字每次点亮的时间约是 300μs，4 的数字也是如此，系统每隔 10 ms 就分别点亮数字 4 与 3，由于人的眼睛有视觉暂留的特性，会让我们以为两个数字都同时显示一样。

```
;PROGRAM NAME:INTR2.ASM
;TEST TIMER0 INTERRUPT(MODE 1)
;
COUNT   EQU     9217          ;10mS FOR XTAL=11.0592MHz
;
CHK_BIT REG     P3.2
DIGIT1  REG     P1.4
DIGIT2  REG     P1.5
;
DISP    EQU     30H
;
        ORG     0000H
        LJMP    RESET
        ORG     0003H         ;INT0 INTERRUPT
        RETI
        ORG     000BH         ;TIMER0 INTERRUPT
        LJMP    INT_TIMER0
        ORG     0013H         ;INT1 INTERRUPT
        RETI
        ORG     001BH         ;TIMER1 INTERRUPT
        RETI
        ORG     0023H         ;SERIAL INTERRUPT
        RETI
;
RESET   MOV     R0,#00H
        DJNZ    R0,$          ;WAIT
        CLR     CHK_BIT
        MOV     DISP,#34H     ;DISPLAY '3''4'
```

```
        MOV     SP,#40H
;
        MOV     A,TMOD
        ANL     A,#11110000B
        CLR     ACC.3           ;GATE=0
        CLR     ACC.2           ;C/T=0
        CLR     ACC.1           ;TIMER0 M1=0
        SETB    ACC.0           ;TIMER0 M0=1
        MOV     TMOD,A
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
        CLR     TF0             ;CLEAR TF0
        SETB    TR0
;
        SETB    PT0
        SETB    ET0             ;ENABLE TIMER0 INTURRUPT
        SETB    EA              ;ENABLE SYSTEM INTERRUPT
;
LOOP    SJMP    LOOP            ;WAITTING
;
;TF0=1
INT_TIMER0
        CPL     CHK_BIT
        CLR     TF0             ;CLEAR TF0
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
        MOV     A,DISP
        ANL     A,#00001111B    ;GET LOW NIBBLES
        SETB    ACC.5           ;BIT5,BIT4=10
        MOV     P1,A
        ACALL   DELAY
        MOV     A,DISP
        SWAP    A
        ANL     A,#00001111B
        SETB    ACC.4           ;BIT5,BIT4=01
        MOV     P1,A
        ACALL   DELAY
        CLR     P1.4
        CLR     P1.5
        RETI
;
DELAY   MOV     R0,#00H
        DJNZ    R0,$
        RET
```

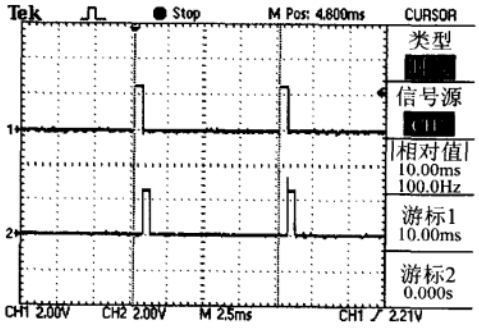


图 11-9　DIGIT1 与 DIGIT2 的波形

注：我们用示波器观察 DIGIT1（P1.4）与 DIGIT2（P1.5）的输出波形，其输出的时间间隔依然是 10 ms 在实际的七段显示控制上，每个数字是分别被点亮的。

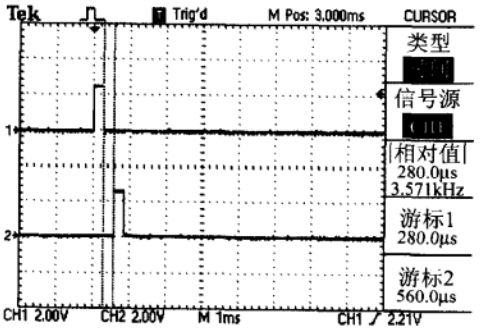


图 11-10　修正后的 DIGIT1 与 DIGIT2 的波形

注：数字与数字显示上要加上一小段空白的时间（图中显示约为 280µs），以免产生不必要的残影，本程序 INTR2.ASM 已经做了修正。

目的：利用 Timer0 定时中断写七段显示器与 LED 的显示程序。

程序名称：CH11_INTR3.ASM。

辅助硬件

数字式示波器。

程序操作说明

上面这个程序除了利用 10 ms 的定时中断去显示数字外，还在定时中断的服务程序内做一个 1 秒的定时器，并运用 SEC 的内容去控制 LED 的闪灭。到目前为止，我们尚未很认真地去分析定时中断服务程序 ISR 的正确写法。比较正确的方式是：进入前把预计变更的寄存器先放到 PUSH 堆栈里，最后离开前才从堆栈中 POP 取回原先的值。

```
;PROGRAM NAME:INTR3.ASM
;TEST TIMER0 INTERRUPT(MODE 1)
;
COUNT    EQU     9217          ;10mS FOR XTAL=11.0592MHz
;
CHK_BIT REG      P3.2
DIGIT1   REG     P1.4
```

```
DIGIT2  REG    P1.5
LED     REG    P3.7
;
DISP    EQU    30H
SEC     EQU    31H
;
        ORG    0000H
        LJMP   RESET
        ORG    0003H        ;INT0 INTERRUPT
        RETI
        ORG    000BH        ;TIMER0 INTERRUPT
        LJMP   INT_TIMER0
        ORG    0013H        ;INT1 INTERRUPT
        RETI
        ORG    001BH        ;TIMER1 INTERRUPT
        RETI
        ORG    0023H        ;SERIAL INTERRUPT
        RETI
;
RESET   MOV    R0,#00H
        DJNZ   R0,$         ;WAIT
        CLR    CHK_BIT
        MOV    DISP,#34H    ;DISPLAY '3''4'
        MOV    SEC,#00H
        MOV    SP,#40H
;
        MOV    A,TMOD
        ANL    A,#11110000B
        CLR    ACC.3        ;GATE=0
        CLR    ACC.2        ;C/T=0
        CLR    ACC.1        ;TIMER0 M1=0
        SETB   ACC.0        ;TIMER0 M0=1
        MOV    TMOD,A
        MOV    TH0,#(65536-COUNT)/256
        MOV    TL0,#(65536-COUNT).MOD.256
        CLR    TF0          ;CLEAR TF0
        SETB   TR0
;
        SETB   PT0
        SETB   ET0          ;ENABLE TIMER0 INTURRUPT
        SETB   EA           ;ENABLE SYSTEM INTERRUPT
;
LOOP    SJMP   LOOP         ;WAITTING
;
;TF0=1
INT_TIMER0
        CPL    CHK_BIT
        CLR    TF0          ;CLEAR TF0
        MOV    TH0,#(65536-COUNT)/256
        MOV    TL0,#(65536-COUNT).MOD.256
```

```
        INC     SEC             ;SEC=SEC+1
        MOV     A,SEC           ;CHECK SEC>=100?
        CJNE    A,#100,$CHK1    ;A-100
$CHK1   JC      $NEXT
        MOV     SEC,#00H        ;SEC=0
;
$NEXT   MOV     A,SEC
        CJNE    A,#40,$CHK2
$CHK2   JNC     OVER
        SETB    LED             ;LED ON
        SJMP    DISPLAY
OVER    CLR     LED
;
DISPLAY MOV     A,DISP
        ANL     A,#00001111B    ;GET LOW NIBBLES
        SETB    ACC.5           ;BIT5,BIT4=10
        MOV     P1,A
        ACALL   DELAY
;ADD    EXTRA   PART
        CLR     P1.4
        CLR     P1.5
        ACALL   DELAY
;
        MOV     A,DISP
        SWAP    A
        ANL     A,#00001111B
        SETB    ACC.4           ;BIT5,BIT4=01
        MOV     P1,A
        ACALL   DELAY
        CLR     P1.4
        CLR     P1.5
        RETI
;
DELAY   MOV     R0,#80H
        DJNZ    R0,$
        RET
;
```

11-10 外部负边沿中断 INT0 程序范例

目的：学习 8051 的外部中断用法。

程序名称：CH11_INTR4.ASM。

辅助硬件

一台可产生 1～10Hz 方波的信号产生器，信号加入 AT2051 控制板的外部中断 INT0 脚。

程序操作说明

当我们把方波信号加入 INT0 接脚时，可以看到 AT2051 控制板上的七段显示值一直往上加，当加超过十进位的 99 时，又从 00 开始往上加，也就是每次信号负边沿产生时，系统就执行一次 INT0 的中断，在 INT0 中断内就是调整 DISP 位置内的值。

```
;PROGRAM NAME:INTR4.ASM
;TEST TIMER0 INTERRUPT(MODE 1)
;
COUNT   EQU     9217        ;10mS FOR XTAL=11.0592MHz
;
DIGIT1  REG     P1.4
DIGIT2  REG     P1.5
LED     REG     P3.7
;
DISP    EQU     30H
SEC     EQU     31H
;
        ORG     0000H
        LJMP    RESET
        ORG     0003H       ;INT0 INTERRUPT
        LJMP    INT_ET0
        ORG     000BH       ;TIMER0 INTERRUPT
        LJMP    INT_TIMER0
        ORG     0013H       ;INT1 INTERRUPT
        RETI
        ORG     001BH       ;TIMER1 INTERRUPT
        RETI
        ORG     0023H       ;SERIAL INTERRUPT
        RETI
;
RESET   MOV     R0,#00H
        DJNZ    R0,$        ;WAIT
        MOV     DISP,#00H   ;DISPLAY '0''0'
        MOV     SEC,#00H
        MOV     SP,#40H
;
        MOV     A,TMOD
        ANL     A,#11110000B
        CLR     ACC.3       ;GATE=0
        CLR     ACC.2       ;C/T=0
        CLR     ACC.1       ;TIMER0 M1=0
        SETB    ACC.0       ;TIMER0 M0=1
        MOV     TMOD,A
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
        CLR     TF0         ;CLEAR TF0
        SETB    TR0
;
;TIMER0 SETTING
        SETB    PT0
        SETB    ET0         ;ENABLE TIMER0 INTURRUPT
;INT0   SETTING
        SETB    IT0         ;NEG FALLING TRIGGER ON INT0
        SETB    EX0         ;ENABLE INT0 INTERRUPT
;       SETB    PX0
```

```
;
        SETB    EA              ;ENABLE SYSTEM INTERRUPT
;
LOOP    SJMP    LOOP            ;WAITTING
;
INT_ET0 PUSH    A
        MOV     A,DISP
        ADD     A,#01H
        DA      A
        MOV     DISP,A
        POP     A
        RETI
;
;TF0=1
INT_TIMER0
        CLR     TF0             ;CLEAR TF0
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
        INC     SEC             ;SEC=SEC+1
        MOV     A,SEC           ;CHECK SEC>=100?
        CJNE    A,#100,$CHK1    ;A-100
$CHK1   JC      $NEXT
        MOV     SEC,#00H        ;SEC=0
;
$NEXT   MOV     A,SEC
        CJNE    A,#40,$CHK2
$CHK2   JNC     OVER
        SETB    LED             ;LED ON
        SJMP    DISPLAY
OVER    CLR     LED
;
DISPLAY MOV     A,DISP
        ANL     A,#00001111B    ;GET LOW NIBBLES
        SETB    ACC.5           ;BIT5,BIT4=10
        MOV     P1,A
        ACALL   DELAY
;ADD EXTRA PART
        CLR     P1.4
        CLR     P1.5
        ACALL   DELAY
;
        MOV     A,DISP
        SWAP    A
        ANL     A,#00001111B
        SETB    ACC.4           ;BIT5,BIT4=01
        MOV     P1,A
        ACALL   DELAY
        CLR     P1.4
        CLR     P1.5
        RETI
```

```
;
DELAY   MOV     R0,#80H
        DJNZ    R0,$
        RET
;
```

11-11 外部低电平中断程序范例

目的：学习 8051 的外部低电平中断的用法。

程序名称：CH11_INTR5.ASM。

辅助硬件

示波器外加一台可产生方波的信号产生器，若此信号产生器又可调整方波的 Duty Cycle 则对本范例更好。

程序操作说明

本程序接受由外部送来的低频数字信号为中断要求信号，只要该信号在低电位停留的时间够长，CPU 就会反应该中断信号，其中 ISR 程序依然是把 DISP 内的值加 1，然后定时中断 ISR 把此值传送到七段显示器上。开始试验时，可以把信号的 Duty Cycle 定在 50%，即该方波在高电位与低电位的时间都相等，频率保持不变，然后逐渐把低电位的时间缩小，从显示器上的变化及示波器上的信号高低比分别观察 8051 是如何反应该中断信号的。你也可以拿一条短路线，直接把 INT0 点与 GND 短路，这时就能看到显示的数值一直在变化着，直到短路线被移开为止。

我们发觉低电平触发和负边沿触发有极大的不同，前者是只要外部输入的状态是低电位时，就执行 ISR 程序然后返回原中断点，若此时 CPU 又察觉外部输入的状态仍是低电位时，仍会再做一次 ISR 程序，所以，假设信号产生器输入方波的频率是 1Hz 时，单片机将有几乎 0.5 秒的时间都在执行 ISR 程序，剩余 0.5 秒的时间做原来的程序，因此我们也可以看到七段显示器的变化是一半时间一直闪烁不停，另外一半时间为静止不动的状态。

```
;PROGRAM NAME:INTR5.ASM
;TEST TIMER0 INTERRUPT(MODE 1)
;
COUNT   EQU     9217        ;10mS FOR XTAL=11.0592MHz
;
DIGIT1  REG     P1.4
DIGIT2  REG     P1.5
LED     REG     P3.7
;
DISP    EQU     30H
SEC     EQU     31H
;
        ORG     0000H
        LJMP    RESET
        ORG     0003H       ;INT0 INTERRUPT
        LJMP    INT_ET0
        ORG     000BH       ;TIMER0 INTERRUPT
        LJMP    INT_TIMER0
```

```
        ORG     0013H           ;INT1 INTERRUPT
        RETI
        ORG     001BH           ;TIMER1 INTERRUPT
        RETI
        ORG     0023H           ;SERIAL INTERRUPT
        RETI
;
RESET   MOV     R0,#00H
        DJNZ    R0,$            ;WAIT
        MOV     DISP,#00H       ;DISPLAY '0''0'
        MOV     SEC,#00H
        MOV     SP,#40H
;
        MOV     A,TMOD
        ANL     A,#11110000B
        CLR     ACC.3           ;GATE=0
        CLR     ACC.2           ;C/T=0
        CLR     ACC.1           ;TIMER0 M1=0
        SETB    ACC.0           ;TIMER0 M0=1
        MOV     TMOD,A
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
        CLR     TF0             ;CLEAR TF0
        SETB    TR0
;
;TIMER0 SETTING
        SETB    PT0
        SETB    ET0             ;ENABLE TIMER0 INTURRUPT
;INT0   SETTING
        CLR     IT0             ;LOW LEVEL TRIGGER ON INT0
        SETB    EX0             ;ENABLE INT0 INTERRUPT
;       SETB    PX0
;
        SETB    EA              ;ENABLE SYSTEM INTERRUPT
;
LOOP    SJMP    LOOP            ;WAITTING
;
INT_ET0 PUSH    A
        MOV     A,DISP
        ADD     A,#01H
        DA      A
        MOV     DISP,A
        POP     A
        RETI
;
;TF0=1
INT_TIMER0
        CLR     TF0             ;CLEAR TF0
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
```

```
        INC     SEC             ;SEC=SEC+1
        MOV     A,SEC           ;CHECK SEC>=100?
        CJNE    A,#100,$CHK1 ;A-100
$CHK1   JC      $NEXT
        MOV     SEC,#00H        ;SEC=0
;
$NEXT   MOV     A,SEC
        CJNE    A,#40,$CHK2
$CHK2   JNC     OVER
        SETB    LED             ;LED ON
        SJMP    DISPLAY
OVER    CLR     LED
;
DISPLAY MOV     A,DISP
        ANL     A,#00001111B ;GET LOW NIBBLES
        SETB    ACC.5           ;BIT5,BIT4=10
        MOV     P1,A
        ACALL   DELAY
;ADD EXTRA PART
        CLR     P1.4
        CLR     P1.5
        ACALL   DELAY
;
        MOV     A,DISP
        SWAP    A
        ANL     A,#00001111B
        SETB    ACC.4           ;BIT5,BIT4=01
        MOV     P1,A
        ACALL   DELAY
        CLR     P1.4
        CLR     P1.5
        RETI
;
DELAY   MOV     R0,#80H
        DJNZ    R0,$
        RET
;
```

这个程序给了我们几个重要的启示：

（1）处理中断程序要特别小心。

（2）处理低电平式的中断程序更要特别小心。

（3）禁止中断时，不仅是下达 CLR EA 指令，同时还要将对应的中断使能位清除，本例中应该多加一个 CLR EX1 指令，停止由 INT1 端接受中断要求信号。

（4）停止不必要的中断，即使是 RET1 指令，也会影响整个单片机系统的执行性能（Performance）。

11-12 串行传输中断程序范例

目的：学习正确使用 8051 的 TI 位中断。

程序名称：CH11_INTR6.ASM。

辅助硬件

PC 机利用 RS485 端口接收 AT2051 送来的显示数据。

程序操作说明

程序每隔一段时间就传送回“HELLO”字符串，并且七段显示幕上的值加 1，这个示范程序已经有一点复杂了，它有 10 ms 定时中断在更新显示屏，同时 LED 也在闪烁，在此一同时间也有串行中断发生，请仔细研究一下程序的流程与写法。

```
;PROGRAM NAME:INTR6.ASM
;TEST SERIAL INTERRUPT(MODE 1)
;
COUNT   EQU     9217            ;10mS FOR XTAL=11.0592MHz
;
DIGIT1  REG     P1.4
DIGIT2  REG     P1.5
LED     REG     P3.7
;
DISP    EQU     30H
SEC     EQU     31H
O_CNT   EQU     32H
;
O_TRUE  REG     20H.0
;
        ORG     0000H
        LJMP    RESET
        ORG     0003H           ;INT0 INTERRUPT
        RETI
        ORG     000BH           ;TIMER0 INTERRUPT
        LJMP    INT_TIMER0
        ORG     0013H           ;INT1 INTERRUPT
        RETI
        ORG     001BH           ;TIMER1 INTERRUPT
        RETI
        ORG     0023H           ;SERIAL INTERRUPT
        LJMP    SERIAL
;
RESET   MOV     R0,#00H
        DJNZ    R0,$            ;WAIT
        MOV     DISP,#00H       ;DISPLAY '0''0'
        MOV     SEC,#00H
        MOV     SP,#40H
;
        MOV     A,#00100001B
        MOV     TMOD,A
;TIMER0 IN MODE 1:16-BIT COUNT
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
        CLR     TF0             ;CLEAR TF0
        SETB    TR0
;
```

```
;TIMER1 IN MODE 2:8-BIT AUTO RELOAD MODE
        MOV     TH1,#FDH
        SETB    TR1             ;TIMER1 START
;
        MOV     SCON,#01000000B
        SETB    REN             ;RECEIVE ENABLE
;
;TIMER0 SETTING
        SETB    PT0
        SETB    ET0             ;ENABLE TIMER0 INTURRUPT
;
        CLR     RI
        CLR     TI
        SETB    ES              ;ENABLE SERIAL INTERRUPT
;
        SETB    EA              ;ENABLE SYSTEM INTERRUPT
;
LOOP    LCALL   LDELAY
        LCALL   LDELAY
;
        MOV     A,DISP
        ADD     A,#01H
        DA      A
        MOV     DISP,A          ;DISP=DISP+1 (PLUS DAA)
;
        MOV     O_CNT,#00H
        SETB    O_TRUE
        SETB    TI
        SJMP    LOOP            ;WAITTING
;
TABLE   DB      ' ','H','E','L','L','O',0DH,0AH
;
SERIAL  JNB     TI,NO_DATA
        JNB     O_TRUE,NO_DATA
;TI=1 AND O_TRUE=1
        CLR     TI
        MOV     A,O_CNT
        INC     O_CNT           ;OUT_CNT=O_CNT+1
        MOV     DPTR,#TABLE
        MOVC    A,@A+DPTR
        MOV     SBUF,A
        CJNE    A,#0AH,NO_DATA
        MOV     O_CNT,#00H
        CLR     O_TRUE
NO_DATA
        RETI
;
;TF0=1
INT_TIMER0
        CLR     TF0             ;CLEAR TF0
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
```

```
        INC     SEC             ;SEC=SEC+1
        MOV     A,SEC           ;CHECK SEC>=100?
        CJNE    A,#100,$CHK1 ;A-100
$CHK1   JC      $NEXT
        MOV     SEC,#00H        ;SEC=0
;
$NEXT   MOV     A,SEC
        CJNE    A,#40,$CHK2
$CHK2   JNC     OVER
        SETB    LED             ;LED ON
        SJMP    DISPLAY
OVER    CLR     LED
;
DISPLAY MOV     A,DISP
        ANL     A,#00001111B ;GET LOW NIBBLES
        SETB    ACC.5           ;BIT5,BIT4=10
        MOV     P1,A
        ACALL   DELAY
;ADD EXTRA PART
        CLR     P1.4
        CLR     P1.5
        ACALL   DELAY
;
        MOV     A,DISP
        SWAP    A
        ANL     A,#00001111B
        SETB    ACC.4           ;BIT5,BIT4=01
        MOV     P1,A
        ACALL   DELAY
        CLR     P1.4
        CLR     P1.5
        RETI
;
DELAY   MOV     R7,#80H
        DJNZ    R7,$
        RET
;
LDELAY  MOV     R1,#00H
$R0     MOV     R0,#00H
        DJNZ    R0,$
        DJNZ    R1,$R0
        RET
```

目的：学习正确使用 8051 的 RI 位中断。

程序名称：CH11_INTR7.ASM。

辅助硬件

PC 机利用 RS485 端口对 AT2051 传送显示数据。

程序操作说明

本程序设置系统可接受串行传输中断，只要有数据（00H～64H）进入到串行传输寄存器 SBUF 时，就将此值传送到七段显示器上。当串行中断发生时，程序必须先判断究竟该中断

要求是由 RI（已收妥）位或 TI（已送妥）位传送出的，在此只检查 RI 位，若 RI=1 代表串行数据已收妥放在 SBUF 中，所以将 SBUF 上的数据转存到累加器中。关于串行通信的更进一步应用将在稍后章节中再进行说明。

```
;PROGRAM NAME:INTR7.ASM
;TEST SERIAL INTERRUPT
;
COUNT   EQU     9217            ;10mS FOR XTAL=11.0592MHz
;
DIGIT1  REG     P1.4
DIGIT2  REG     P1.5
LED     REG     P3.7
;
DISP    EQU     30H
SEC     EQU     31H
I_CNT   EQU     32H
;
O_TRUE  REG     20H.0
;
        ORG     0000H
        LJMP    RESET
        ORG     0003H           ;INT0 INTERRUPT
        RETI
        ORG     000BH           ;TIMER0 INTERRUPT
        LJMP    INT_TIMER0
        ORG     0013H           ;INT1 INTERRUPT
        RETI
        ORG     001BH           ;TIMER1 INTERRUPT
        RETI
        ORG     0023H           ;SERIAL INTERRUPT
        LJMP    SERIAL
;
RESET   MOV     R0,#00H
        DJNZ    R0,$            ;WAIT
        MOV     DISP,#00H       ;DISPLAY '0''0'
        MOV     SEC,#00H
        MOV     SP,#40H
;
        MOV     A,#00100001B
        MOV     TMOD,A
;TIMER0 IN MODE 1:16-BIT COUNT
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
        CLR     TF0             ;CLEAR TF0
        SETB    TR0
;.
;TIMER1 IN MODE 2:8-BIT AUTO RELOAD MODE
        MOV     TH1,#FDH
        SETB    TR1             ;TIMER1 START
;
        MOV     SCON,#01000000B
        SETB    REN             ;RECEIVE ENABLE
```

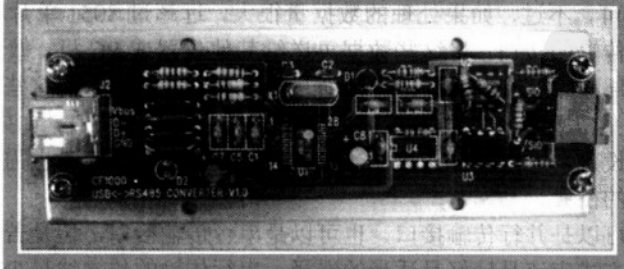
```
;
;TIMER0 SETTING
        SETB    PT0
        SETB    ET0             ;ENABLE TIMER0 INTURRUPT
;
        CLR     RI
        CLR     TI
        SETB    ES              ;ENABLE SERIAL INTERRUPT
;
        SETB    EA              ;ENABLE SYSTEM INTERRUPT
;
LOOP    LCALL   LDELAY
        LCALL   LDELAY
        SJMP    LOOP            ;WAITTING
;
;
SERIAL  JNB     RI,NO_DATA
        CLR     RI
        MOV     A,SBUF
        CJNE    A,#100,$NEXT
$NEXT   JNC     NO_DATA
        MOV     DISP,A          ;DISPLAY A NEW VALUE
NO_DATA RETI
;
;TF0=1
INT_TIMER0
        CLR     TF0             ;CLEAR TF0
        MOV     TH0,#(65536-COUNT)/256
        MOV     TL0,#(65536-COUNT).MOD.256
        INC     SEC             ;SEC=SEC+1
        MOV     A,SEC           ;CHECK SEC>=100?
        CJNE    A,#100,$CHK1;A-100
$CHK1   JC      $NEXT
        MOV     SEC,#00H        ;SEC=0
;
$NEXT   MOV     A,SEC
        CJNE    A,#40,$CHK2
$CHK2   JNC     OVER
        SETB    LED             ;LED ON
        SJMP    DISPLAY
OVER    CLR     LED
;
DISPLAY MOV     A,DISP
        ANL     A,#00001111B ;GET LOW NIBBLES
        SETB    ACC.5           ;BIT5,BIT4=10
        MOV     P1,A
        ACALL   DELAY
;ADD EXTRA PART
        CLR     P1.4
        CLR     P1.5
        ACALL   DELAY
;
```

```
        MOV     A,DISP
        SWAP    A
        ANL     A,#00001111B
        SETB    ACC.4          ;BIT5,BIT4=01
        MOV     P1,A
        ACALL   DELAY
        CLR     P1.4
        CLR     P1.5
        RETI
;
DELAY   MOV     R7,#80H
        DJNZ    R7,$
        RET
;
LDELAY  MOV     R1,#00H
$R0     MOV     R0,#00H
        DJNZ    R0,$
        DJNZ    R1,$R0
        RET
```

习　题

1. 8051 的中断时机有哪些？请列举 5 项。
2. 应用 8051 程序时，当程序具有哪些特征时可考虑使用中断写法？
3. MCS-51 系列单片机提供了几个中断源？它们的地址及名称是什么？
4. 在哪些情况下，8051 系统接收到中断要求信号时不会产生反应？
5. 8051 所提供的中断寄存器有哪些？试绘出其结构图。
6. 中断源如何设置？（提示：6 个步骤。）
7. PC 机用于各种接口卡中断的常用方式有哪些？
8. 请依自己的想法，对本章所附的 CH11_INTR1.ASM 程序码进行注释。

12



USB 转 RS485 接口板可以连接多达 32 个 RS485 的设备，用此模式通信时每个设备都要有其独立的 ID 值

第 12 章　8051 串行通信彻底研究（一）

8051 的串行通信接口赋予了单片机新的面貌，所有的数据可以通过串行接口与其他仪器共享。许多设备的设置值要在面板上进行多次的调试，现在只要保留串行通信端口就行了，甚至于程序的更新也可以通过串行端口来下载，也就是说，8051 添加了串行通信后，许多事情都可以进行远端遥控。串行通信真的很重要，因此我们特地分成两章彻底讲解 8051 的串行接口与相关程序。

12-1　为何要通信

当 8051 单片机执行 MOVX A, @DPTR 指令时，代表向外部存储器读取一个 8 位的数据，此时 CPU 会在适当的时机内送出地址值及 RD 信号，通知外部存储器将该地址的内容值送回 CPU，整个过程只用了两个机器周期的时间（约 2μs）就完成，传送速度相当快，当所有操作执行完后，CPU 的累加器 ACC 内就有一个 8 位的内容值，这种一次 8 位数据传输的方式称为并行传输，它的特点是传输速度较快，但是传输线路较多且传输距离较短，通常并行传输的距离都限定在数米以内，这是因为并行数据进行高速传输时会相互干扰，距离过长时，内部信息会相互干扰，再加上外部的信息干扰会变得相当严重，所以 PC 机上超过 10m 以上的数据传输，都改用同步或异步串行传输的方式，而 PC 机常用的串行传输接口 RS232C 正是非同步传输的接口标准。

并行传输时每次传输的单位是字节（8 位），但是串行传输时，其传输的单位却是位，在传输线上变化 8 次后才能组合成 1 个字节的数据。串行传输的特点是，传输线数少，传输距离远（可达数公里），但其传输速度相对地较并行传输慢了许多，这刚好印证了鱼与熊掌不可兼得的道理，我们很难找到一种传输方式既能大量传输且传送距离又很远的。

如果单片机 8051 只要求自己工作及控制时，是不需要做串行传输的，只需依照程序的流程做好各项事件即可，不过，如果处理的数据量很大，远超过 8051 单片机本身所能负荷时，就需要一个数据传输的接口，将这些数据再送给其他仪器或 PC 机，做进一步的处理或存储。如果 8051 内部程序控制的因素相当多而且随时在进行修正，而这些修正值是其他的 PC 机或设备产生的，也需要有一个传输接口来接收（Receive）这些值。如果有多个 8051 单片机同时在控制一套设备，一定有一些数据或资料必须大家共享，此时也需要一个传输接口来做彼此间数据交换的桥梁。

上述的传输接口可以是并行传输接口，也可以是串行传输接口，但是若考虑到传输距离时，后者的串行传输方式应该是较好且适当的选择。串行传输的传输线最少只需要 3 条线：传送端（Transmit）、接收端（Receive）和地端（Ground）。相同的情况下，并行传输至少需要 10 条以上传输线：数据线 8 条、双方交握控制线（Handshake）和地端。

12-2 如何进行串行通信

PC 机内部最基本的处理单位是字节，每个字节又分成 8 个位，如果要做串行传输时，必须有一个接口（Interface）将原先的并行 8 位数据转换成串行 1 位数据，这就是所谓的并行转串行转换器（Parallel to Serial Device），将数据逐一送到外部，如果外部的 PC 机或设备要接收这些数据时，必须自备一个串行转并行转换器（Serial to Parallel Device），先把数据恢复成原来并行（Byte）的格式，然后才由 CPU 读入，做进一步的数据处理。串行通信的简单示意如下图所示。

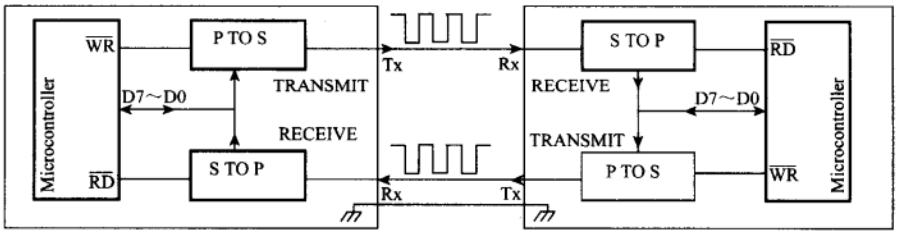


图 12-1 串行通信的示意图

串行传输方式的确可以减少传输线的数量，但是另外衍生出一大堆技术上的问题必须克服，若这些问题无法有效解决，串行通信根本就不可行！以下就是这些问题点以及解决之道。

1. 传输速度

当我们利用并行到串行转换器将数据送出时，每个位之间一定有一个传输速度存在，也就是每隔多少时间内送出一个位的数据，由于发送和接收端的两个微机系统可能完全不同，若不能定义出可行的传输速度来，收与送端的数据可能完全走样，所以在串行传输接口上才有所谓的波特率（Baud Rate）存在，波特率直接定义出数据串行间的速度，其单位是 b/s（bit per Second），现在微机间传输较常用的波特率有 1200、2400、4800 和 9600 b/s，以 1200 b/s 为例，这代表每秒可以传送 1200 位的数据串，若把 1200 除以 8 得到 150，理论上每秒最多可以送出 150 字节的数据，传输速度愈快时愈能减少传输所花的时间，但这却是双方的事，当两部 PC 机设备要做串行传输时，首先要定好传输速度。

2. 传输格式

当一大串 0 与 1 的数据出现在接收端时，我们可以想一想硬件线路是如何对待这些数据的，若处理不当，这些数据将变成垃圾，反之，则成为有用的资源。我们也知道 CPU 处理以下的各项操作都是要花时间的：

（1）从串行接口中写入或读取 1 个字节的数据。

（2）从存储器取出一个数据。

（3）从存储器中存入一笔数据。

（4）串行接口送出数据串行。

（5）主程序判断哪些数据该送出。

（6）主程序接收数据后的处理过程。

我们遇到的问题是所有的操作都有时间上的延迟，如何去保持正确的收与送。这项操作必须由传输格式定义清楚，接收双方才得以顺利地交换信息，为了“正确”的缘故，我们必

须在原有数据串行中加入所谓的开始位（Start bit）和停止位（Stop bit），以便对方的硬件得以与收到的数据串同步。请看图 12-2 加入 Start bit 和 Stop bit 之后的数据串格式。

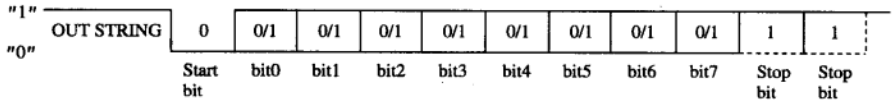


图 12-2　包括了 Start bit 和 Stop bit 的串行数据格式

注：（1）每个送出去的数据（8 位）都包括 1 个 Start bit 开始位和 1 个 Stop bit 停止位。

（2）Start bit 开始位的持续时间与每个单一位的传送时间相同。

（3）Stop bit 结束位的时间可以选择 1、1.5 或 2 个位的传送时间。

（4）加入开始及结束位的主要目的是让接收方的硬件线路或程序得以同步接收。

例如，图 12-3 所示的数据串代表 ASCII 码的'Z'=01011010B。

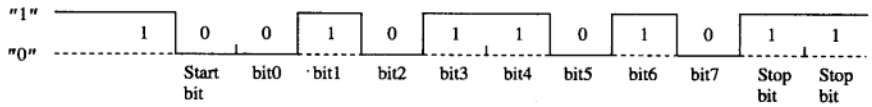


图 12-3　ASCII 码'Z'的串行格式

当接收端的电位一直维持在高电位时，代表传输线处于闲置（Idle）的状态，如果接收端检测到低电位 0 的状态后，再等待 1/2 的位时间，若传输线仍为低电位状态时，代表已有开始位被检测到，则接收端每隔一个位的时间就读入一个位的数据，刚好组合成 1 个字节的数据，最后再隔 1 个位的时间后读入另一个位的值，若此位为 1 时代表已收妥停止位，此时传输线回到高电位的闲置状态，接收端再度进入检测开始位的状态，准备接收下一个数据串，看图 12-4 示范如何接收一个 ASCII 的“Z”字符，上述的检测及位组合的操作看似复杂，但已被设计成专用的硬件串行收送接口，只要定义好波特率，就可展开串行的收发操作。

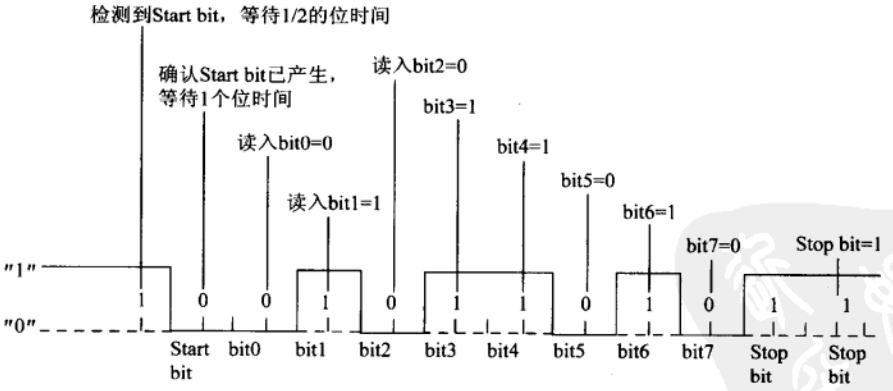


图 12-4　接收端如何读入数据串

说明：（1）当传输线由高变低的瞬间才启动接收端的电路。

（2）接收端必须确认 Start bit 产生后，才继续接收数据串，每次的间隔为收送 1 个位的时间。

（3）传输线最后一定要回高电位，代表数据已结束。

（4）接收端在数据位的中央被取样并读入内部的串行转并行的电路中，若波特率稍有误差时，仍能传送无误。

1. 传输电平

当传输的距离在一两米内时，使用+5V 的数字电平来传送数据是没什么问题的，可是距离一旦拉长后，5V 的电平可能无法有效地防止外界各种杂信的干扰，于是有人将 0～5V 的标准数字传输信号提升到±12V 的信号位准，大大地提高了传输的距离和抗干扰的能力。

2. 传输过程的正确或错误判断

当数据通过传输线传送时，谁都无法保证数据会正确无误地传到接收端，有可能是传输接口故障或遭受突然的干扰，导致部分数据的错误，为了保证每个字节的数据的正确性，有人在原先数据串的停止位前再加入一个位做检查，这个检查位又称为奇偶校验位（Parity Bit）。这类的检查方式又分成两种：奇位法（Odd Parity）和偶位法（Even Parity），前者是使数据串中高电平“1”的个数保持偶数值。以 ASCII 码的“Z”字符为例，其二进制为 01011010B，1 的个数有 4 个，若欲加入奇位的检查码时，这个新加入的检查位应该是 1，以便使全部 1 的个数保持奇数值。若想加入偶位的检查码时，新加入检查位应该是 0，以便使全部 1 的个数保持偶数值。这种加入同位检查的方法对于有偶数字出错的情况无法辨识，比较好的方法是在一段数据串最后加入一个检查字节，若此字节与在接收端经过某种计算程序后的值相符合时，代表这整段数据串都是正确无误的，若有错误时可再要求发送端重送一次，这样就能确保接收数据的正确性了。

12-3 RS232C 的规格

在 PC 机与 PC 机间的通信协议（Protocol），其实就是详细定义数据交换时的传输速度、传输格式和传输电平，在串行通信中最常用的通信协议就是 RS232C 接口，该接口是由美国的 EIA 协会（Electronic Industries Association）所制定的串行传输规格，主要的应用是在 PC 机与通信设备间的串行数据传输，图 12-5 是 RS232C 的应用示范，此时的 DTE 实际上就是一部 PC 机设备，而 DCE 则是调制解调器（MODEM），由图中我们可以看出 DTE 和 DCE 间的 RS232C 线路长度还是有所限制的，若想做数十甚至数百公里的数据传递时，还是要靠电信局的网络做转接才行，此时就要靠图中的 MODEM 将数据调变成音频信号再转到电话线上，接收端的 MODEM 则把音频信号解调回原来的数字信号，再传回给接收端的 PC 机设备。

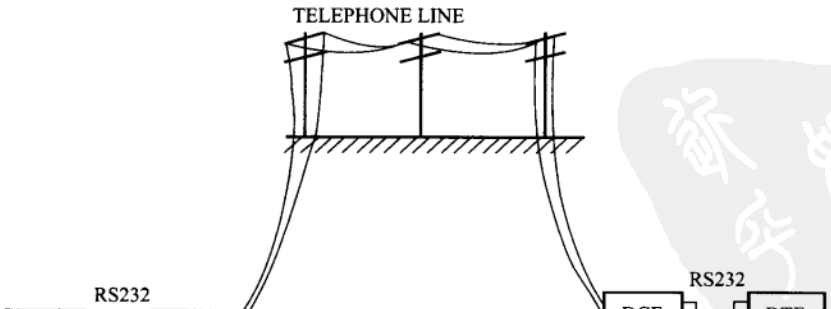


图 12-5 RS232C 的应用示范图

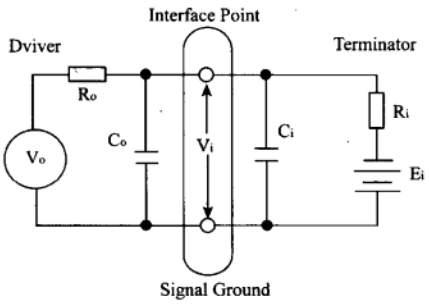


图 12-6　RS232C 的接口等效电路图

注：V_o：开路时电压。
R_o：回路上的直流阻抗。
C_o：发送端 Cable 的电容量。
V_i：连接点的电压。
C_i：接收端的 Cable 电容量。
R_i：接收端的直流阻抗。
E_i：接收端的偏压。

表 12-1　RS232C 的电气规格表

项　目		规　格
发送端	最大输出电压（无负载时）	25V（绝对值）
	最小输出电压（有负载时）	当负载为 3～7kΩ 时输出为 5～15V(绝对值)
	电源开路时的最小输出阻抗	300Ω
	短路时的最大输出电流	500mA（绝对值）
	Slew Rate	最大为 30V/μs
接收端	输入阻抗	3～7kΩ
	输入临界电压	3V（绝对值）
	输入电压	最大 25V（绝对值）

	"SPACE"	"MARK"
二进制表示	0/ON	1/OFF
输出条件	+5～+15V	-5～-15V
输入条件	≥+3V	≤-3V

表 12-2　标准 RS232 接口信号名称

引脚	信号名	信号方向 DTE DCE	信号名称说明	PC 机上的 25P 接头	PC 机上的 9P 接头
1	AA	←→	Cable Shield	1（FG）	
2	BA	→	Transmit Data	2（TR）	2
3	BB	←	Receive Data	3（RD）	3
4	CA	→	Request to Send	4（RTS）	7

续表

引脚	信号名	信号方向 DTE DCE	信号名称说明	PC 机上的 25P 接头	PC 机上的 9P 接头
5	CB	←	Clear to Send	5（CTS）	8
6	CC	←	Data Set Ready	6（DSR）	6
7	AB	←→	Signal Ground	7（GND）	5
8	CF	←	Carrier Detector	8（DCD）	1
9			未定义		
10			未定义		
11			未定义		
12	SCF	←	Backward CH Carrier Detector		
13	SCB	←	Backward CH Clear to Send		
14	SBA	→	Backward CH Transmit Data		
15	DB	←	Transmit Element Timing（DCE）		
16	SBB	←	Backward CH Receive Data		
17	DD	←	ReceiveElementTiming（DCE）		
18			未定义		
19	SCA	→	Backward CH Request to Send		
20	CD	→	Data Terminal Rendy	20（DTR）	4
21	CG	←	未定义		
22	CE	←	Ring Indicator	22（RI）	9
23	CI,C	←→	Data Speed Select		
24	DA	→	Transmit Element Timing（DTE）		
25			未定义		

注：PC 机上的 RS232C 接头是设计给 MODEM 使用的，所以实际的引脚数可以减少到只有 9 个。

12-4 8051 的串行接口概述

8051 的设计中大概属串行接口是最令软硬件工程师欣赏的，在这个串行接口中，它不仅提供了全双工（Full Duplex）传输的功能，并且以缓冲式的接收模式来处理所收到的串行数据，所谓的全双工功能就是指 8051 的串行接口能够同时接收及发送串行数据，这种安排可以使串行传输的效率提到最高，但是程序的写法却是最复杂的，另外，缓冲式的接收模式则是指 8051 的串行接收接口在接收到 1 个字节的数据后，还能够继续接收下一个字节的数据，若前一个字节没有被读走时，接下来的这笔数据将会覆盖掉上一次数据，而造成一个字节数据的漏失，所以如果想好好且无误地接收串行数据时，最好是将串行传输的程序以中断的写法处理，这样才不会造成过多的失误。

8051 串行数据的收发都是通过特殊寄存器 SBUF 来处理的，只要设置好串行传输的模式，之后的 MOV SBUF，A 指令就是把值送到串行传输寄存器上，并立即将该笔数据以串行方式送出，而 MOV A，SBUF 指令则是由串行传输寄存器上取回外界送来的串行数据，两指令中的 SBUF 并不是指到一个相同的寄存器，而是分别属于两个不同的寄存器，这些观念将在串行传输的模式设置的章节中再做详尽的说明。

8051 串行传输共有 4 种模式可供选择，其功能几乎可以涵盖所有串行传输的各项应用，我们将先谈这 4 个模式的操作原理，然后在接下来的各节中再对各个模式做程序示范及串行数据波形观察。

（1）串行传输模式 0。此模式似乎不属于我们先前所谈到的串行传输方式，纯粹是做系统 I/O 的扩展，其串行数据是由 RxD 脚送出或读入，而 TxD 产生串行数据在移位时所必须的脉冲，并且此脉冲的频率固定是系统石英晶体频率的 1/12，刚好是 8051 一个机器周期的时间。

（2）串行传输模式 1。此模式下每次总共送出或接收 10 个位的数据，包括了 1 个 Start bit、8 个位数据本体和 1 个 Stop bit，其传输速率可通过定时/计数器来设置。若考虑到与 PC 的连线，此模式是唯一的选择，我们所有的范例将尽量用此模式做例子。

（3）串行传输模式 2。此模式较模式 1 多送出 1 个位，每次可送出或接数 11 个位的串行数据，其中包括了 1 个 Start bit、8 个数据位，1 个可程序设置的第 9 个数据位和最后的 Stop bit，其传输速率仅有两种选择，分别是系统石英晶体频率的 1/32 或是 1/64。

（4）串行传输模式 3。每次传送或接收 11 个位的数据，包括了数据开头的 1 个 Start bit、8 个数据位、1 个可程序规划设置的数据位和数据最后的 Stop bit，和模式 2 不同的是此模式的串行传输速率是可变的。

由这里的串行模式概述中我们可以约略看出，模式 0 是供做系统硬件扩展用的，反而不是串行数据的互传，而模式 2 的传输波特率只有两种选择，除非是同一系统中多个 CPU 连线，否则其实用价值不算太高，模式 1 和模式 3 的波特率都是可变动的，可以轻易地与其他 PC 机系统或设备连线，所以这两个模式将是我们稍后程序示范的重点所在，值得一提的是，串行传输模式 3 又可支持多微处理机的通信用，特别适用在多台 8051 的控制设备连线上，如果有多台 AT2051 也可以做这方面的连线实验。表 12-3 是 8051 的 4 种串行传输的各项特性比较。

表 12-3　串行传输模式比较表

模式	每笔串行数据	发 送 端	接 收 端	传输速率	应用范围
模式 0	8 个数据位	RxD TxD 产生脉冲	RxD TxD 产生脉冲	$f_{osc}/12$	硬件 I/O 扩展用
模式 1	1 Start bit+ 8 数据位+ 1 Stop bit	TxD	RxD	可变	不同系统连线用
模式 2	1 Start bit+ 8 数据位+ 1 可设置数据位+ 1 Stop bit	TxD	RxD	$f_{osc}/32$ $f_{osc}/64$	相同系统连线用
模式 3	1 Start bit+ 8 数据位+ 1 可设置数据位+ 1 Stop bit	TxD	RxD	可变	不同系统连线用

注：f_{osc} 是指 8051 系统所使用的石英振荡晶体频率。

12-5 串行传输控制有关的寄存器：SCON

除了波特率设置必须动到 Timer 外，所有串行模式的设置及状态值都由 SCON 寄存器掌握，SCON 寄存器占用 98H 地址，而且可做位寻址，所以我们可以用 MOV SCON，#DATA 指令设置串行传输的模式，亦可以用 SETB 或 CLR 指令单独设置 SCON 中某个位的状态值。8051 系统开机后，若定义好 SCON 寄存器的值后，8051 内部的串行接口就进入预备接收或发送的状态下，只要程序中有值填入 SBUF 寄存器时，串行接口会随即将并行数据化成串行值，并依程序所指定的模式，一一将数据送出。

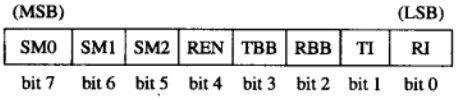


图 12-7 SCON 寄存器可做位寻址

SM0 和 SM1：组合成 4 种串行传输模式，其组合方式如下：

（1）SM0=0，SM1=0：模式 0 的移位寄存器，其波特率为 fosc/12 且不可更改。

（2）SM0=0，SM1=1：模式 1 的 8 位标准串行数据传输格式，波特率只有两种选择，分别是 fosc/32 和 fosc/64。

（3）SM0=1，SM1=1：模式 3 的 9 位标准串行数据传输格式，波特率可用程序设置。

（4）SM2 或称 SCON.5：在串行模式为 2 或 3 时，决定是否做多微处理器的通信。

（5）REN 或称 SCON.4：决定是否让串行接口接收外部传来的串行数据，当 REN=1 时代表串行接收接口已经启动准备接受串行数据。

（6）TB8 或称 SCON.3：在串行传模式为 2 或 3 时，欲加入的第 9 个位数据，若下达 SETB SCON.3 时，送出去的串行数据中，第 9 个位值就是 1，反之是 CLR SCON.3 时，则第 9 个位值就是 0。

（7）TB8 或称 SCON.2：在串行传输模式为 2 或 3 时，所收到的第 9 个位的状态值，当串行接口被设置成模式 0 时，此位没有被使用到，若是模式 1 时，TB8 的内容是最近一次串行数据其 Stop bit 的状态。

（8）TI 或称 SCON.1：发送中断标志位，可对 CPU 要求中断，当串行接口处于模式 0 时，送完第 8 个位数据后，TI 就会变为 1，表示数据已送完可再传送下一组数据。在其他 3 种模式下，在送出 Stop bit 时，TI 就由 8051 内部的硬件设成 1。本位不会自动清除成 0，所以程序在载入 SBUF 一个传送数据的同时，也要下一个 CLR TI 的指令，以免 TI 一直处于 1 的状态而一直产生中断要求信号。

（9）RI 或称 SCON.0：接收中断标志位，也可对 CPU 要求中断，当串行接口被设置成模式 0 时，8051 内部的硬件会在 TxD 脚上自动送出 8 个脉冲，以接收串行数据，当收到第 8 个脉冲后，RI 会被设成 1，表示数据已收妥，CPU 应尽快取走 SBUF 上的串行数据。在其他模式时，在收到 Stop bit 的一半时间时，也会把 RI 设成 1，代表该串行接口已收妥一笔数据，程序应该在最短的时间内来取走 SBUF 内的数据，否则在下组数据收妥后，会自动覆盖上一组数据，我们的程序在读取 SBUF 值的同时，要再下一个 CLR RI 的指令，代表 RI 状态已被更新一次了。

12-6 8051 串行传输的波特率设置

8051 的 4 个串行传输模式中，有两个模式是无法变动其波特率的，这包括了模式 0 和模式 2，而模式 1 和模式 3 可依我们的需要设置各种高低速的波特率。

（1）模式 0 的波特率。固定为石英振荡器频率的 1/12。

（2）模式 2 的波特率。有两种速率可选择，而其选择位在 PCON 功率控制寄存器的 SMOD 位上，这种安排方法恐怕又是 Intel 设计时的权宜之计，因为 SCON 中 8 个位都用上了，只好找其他寄存器中空余的位置利用。波特率=振荡器频率/64（SMOD=0 时），波特率=振荡器频率/32（SMOD=1 时），当 8051 系统刚开机时，若 PCON 寄存器程序都没碰到时，其 SMOD=0，所以若指定模式 2 的串行传输时，波特率正是振荡器频率除 64 后的值。

（3）模式 1 和模式 3 的波特率。由 8051 Timer1 的定时溢位率和 SMOD 位决定，如果用公式表示时为：

$$\text{波特率}=\frac{2^{\text{SMOD}}}{32}\times(\text{Timer1 的溢位率})$$

通常在这种情况下，Timer1 都会被定义成模式 2 的定时自动载入功能，实际有计数功能的计数器为 TL1，而 TH1 则当成自动载入时的设置值，所以波特率值又可以写成：

$$\text{波特率}=\frac{2^{\text{SMOD}}}{32}\times\frac{\text{振荡器频率}}{12\times[256-\text{TH1值}]}$$

$$=\frac{2^{\text{SMOD}}}{32}\times\frac{\text{机器周期的频率}}{[256-\text{TH1值}]}$$

表 12-4 列出几种常用的波特率下，Timer1 中 TH1 所要的设置值请留意，此时的 Timer1 一定工作在模式 2：自动载入的定时模式。

表 12-4 8051 在模式 1 和模式 3 下，常用波特率的设置值

Baud Rate（BitPer Second）	f_{osc}	SMOD	Timer1（8 位）	Error（%）	Remark
62500	12MHz	1	FFH	0.0	只在同一系统
19200	11.0592MHz	1	FDH	<0.1	
9600	11.0592MHz	0	FDH	<0.1	
4800	11.0592MHz	0	FAH	<0.1	
2400	11.0592MHz	0	F4H	<0.1	
1200	11.0592MHz	0	F8H	<0.1	

注：（1）Timer1 在模式 2 自动载入的定时模式。

（2）从 1200~19200 b/s 都是 RS232C 数据传输常使用的波特率。

（3）Timer1 的载入值也可以是其他值，但如此就无法与其他不同系统的 PC 机设备连接了。

（4）AT2051 控制板使用的石英振荡晶体频率为 11.059MHz，并且以 9600 b/s 的波特率与 PC 连接。

编写 8051 串行传输的程序时，程序设计人员将有绝对的现场感，似乎整个系统都在我们的掌握中，但是若稍有不慎，则会造成程序不知去向如何，除非关机否则怎么处置都挽救不回，其中有部分是串行传输程序处理不当所引起的，8051 的串行传输接口设置步骤应该以下面提供的方式来设计，才不至于造成系统连线的失败，若连线本身不成功，接下来的数据传输就不需要提了。具体的设置步骤如下：

（1）决定波特率，并通过表 12-5 找到 Timer1 的设置值。波特率越高传输的时间越短，但是传与收双方的程序必须反应够快，否则会有数据遗漏。以 9600 b/s 为例，传送 8 位数据共花了 8×1/9600=0.000833s=833μs 的时间，8051 每个指令处理的时间约是 1μs 左右，看起来似乎程序有足够的时间去获取外界送来的串行数据，但是程序如果刚好卡在一个循环中时，结果就很难想象了。

（2）设置 TMOD 值，让 Timer1 工作在模式 2 下的自动载入状态，另外的 Timer0 可做其他用途。

（3）对 TH1 及 TL1 填入计数值，由于 Timer1 工作在自动载入模式，当 TL1 计数到上限值并溢位后，8051 内部会自动把 TH1 的值送到 TL1 上，自动继续定时的操作。

（4）启动 Timer1，也就是下达 SETB TR1 指令，让 Timer1 开始操作。

（5）决定 SCON 寄存器的设置值，决定传输模式、是否接收等操作。

（6）决定是否产生串行中断，有需要时另依中断设置步骤做好各项程序中断前的准备操作。

（7）清除 RI 和 TI 标志位，并设置 REN=1，让串行的接收及发送两接口进入待命状态。完成以上的设置步骤后，串行接口已经开始运行，若有串行数据进入时，串行接口会逐一接收，等到所有数据都到齐后，RI 标志位会设成 1，此时程序可用询问（Polling）或中断（Interrupt）两方式得到 RI 的状态，若 RI=1 时再做一个 SBUF 读取的操作，得到最新的串行数据值，然后把数据再转存到系统串行数据缓冲区内。最后的收尾操作是 CLR RI 将该标志位清除，以免下个检查 RI 的程序误操作。

当我们欲通过串行接口送出数据时，要先确认 TI 是否为 1，是的时候代表上一个数据已完全送出，我们只要将数据填入 SBUF 中，就可以做此次数据的传输，然后把 TI 设成 0，代表此数据准备送出，等到 TI 又等于 1 时，表示此数据已完全送到其他 PC 机系统或设备中，如果我们不检查 TI 标志位，随意送出一段数据字符串时，很可能会有大部分数据漏失掉，造成传输上的严重错误。

12-7 串行传输模式 0 彻底研究

8051 串行传输模式 0 提供了同步的数据串行发送和接收的功能，在此模式中，所有串行数据的进出都由 RxD 脚接收或发送，而 TxD 脚则提供串行数据移入（Shift in）或移出（Shift out）所需要的脉冲（Clock），其送出串行数据的操作就好像 TTL 系列的 74164 一样，而接收串行数据的操作如同 74165 一样，接下来我们分两个方面，分别解释如何利用此模式发送数据，Intel 手册中又称本模式为 I/O 扩展模式（I/O Expansion Mode），利用此模式可以非常容易地扩展 I/O 的数量。图 12-8 是串行模式 0 下的框图。

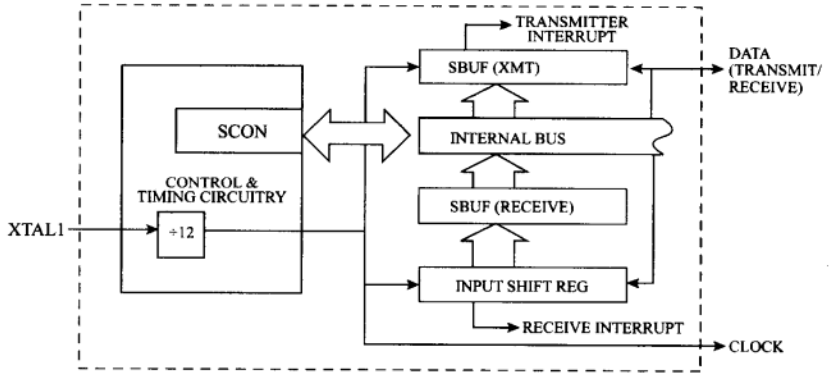


图 12-8　串行模式 0 下的框图

注：SBUF 分别有接收及发送两个，虽然名称相同但功能完全不同。

当模式 0 被设置好时，只要有任何指令对 SBUF 做数据载入的操作时，SBUF 将以一个机器周期的速率逐一将 8 个位数据移出，最低位（LSB）先移出，最后才是最高位（MSB），当数据的 MSB 位移出时，TI 标志位也成为 1，代表数据已传送完毕，这也就是说，8051 只花约 8μs（8 个机器周期）的时间就可把 1 个字节数据以串行的方式送出。由于串行数据只由 RxD 脚进出，所以同一时间内，只能做发送或接收的其中一项操作。

若想在模式 0 中接收串行数据时，只要将 REN 位设成 1 开始接收（此时负责传送的 SBUF 停止操作），并且在清除 RI 位后，8051 的 TxD 端会立刻送出 8 个移位脉冲，脉冲速率刚好是 8051 一个机器周期的速率，大约 8μs 左右的时间内就读回这 8 个串行位数据，通常做这类应用时，外部都有类似 74165 的移位寄存器，方便将并行数据转成串行数据，然后才传回 8051 单片机系统中。

目的：观察串行通信模式 0 的串行数据和脉冲的输出情形。

程序名称：.S1.ASM。

辅助硬件

数字式示波器观察 TxD 和 RxD 的信号。

程序操作说明

本测试程序将串行接口定义成模式 0，然后不断地将 81H 和 AAH 两个值送到 SBUF 中，依照本节的讲法，我们应该可以在 RxD 和 TxD 分别看到数据和脉冲的信号变化，在 AT2051 控制板的 P3.0 上有 RxD 和 TxD 的信号输出，示波器的测试点可直接加在这里，除此之外，我们还碰到一个难题，那就是 AT2051 控制板的 P3.0 是接到串行接收 IC 75176 的输出端，如果依此模式送出数据串，刚好会造成两个数字输出接在一起的冲突，不过，还是有办法克服的，那就是把 P3.0 脚拨开，不要让它与 IC 座接触，那就不会有状态不同的冲突发生了。

只要调整示波器的水平时基，应该可以在示波器的屏幕上看到数据 81H 和 AAH 的串行数据，伴随着 16 个脉冲信号（由 TxD 脚输出），除了对照串行数据是否正确外，另外要计算 TxD 操作变化时，所花费的时间，由示波器上我们也可以看出 TxD 和 RxD 平常都保持在高电位，当 TxD 由高变为低电位的同时，RxD 的状态正是输出串行数据有效值的时候。程序中的 MUL AB 和 DIV AB 指令只作延迟时间用，让串行接口有足够的时间（8.12μs+1.8μs）送出串行数据串，这项操作是串行传输时一定要做的操作。

```
;PROGRAM NAME S1.ASM
;TEST SERIAL PORT MODE 0,SHIFT OUT
;
BUZZER      REG         P3.4
LED         REG         P3.7
RESET       MOV         R0,#00H
            DJNZ        R0,$
            CLR         BUZZER
            CLR         BUZZER
;
            MOV         SP,#60H
            MOV         SCON,#00000000B     ;MODE 0 DEFINE
LOOP        MOV         A,#81H
            MOV         SBUF,A
            MUL         AB
            MUL         AB
            MOV         A,#AAH
            MOV         SBUF,A
            DIV         AB
            DIV         AB
            SJMP        LOOP
;
```

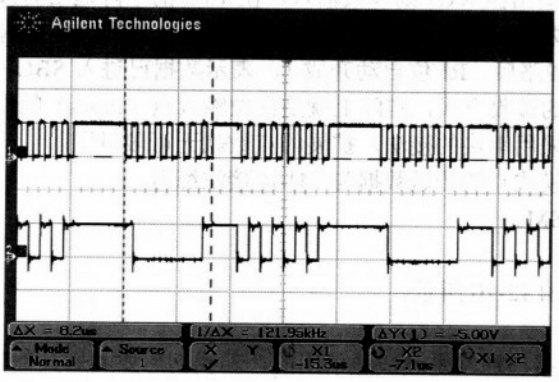


图 12-9　模式 0 所送出的串行波形

注：（1）第一个串行数据送出的值分别是 10000001，由于传送时是 LSB 先开始，所以真实的传送值是 81H。
（2）第 2 个串行数据送出的值分别是 01010101，所以真实的传送值是 AAH。
（3）AT2051 使用的频率为 11.059MHz，所以其一个机器周期的时间约是 1.085μs。
（4）TxD 由高电位降为低电位的时候，RxD 的值正好是传送出的真实值。
（5）图中标示 X 和 O 间的时间，刚好是 7 个半脉冲，由图中看到这段时间为 8.12μs，换算后刚好是 7 个半机器周期的时间。

8051 串行模式 0 下所送出的数据和脉冲可以直接接到 74164 IC 上，74164 会把串行的数据又转换成并行数据的输出，而且可以依我们的需要扩展多个 74164，利用串接的方式把多个 74164 连在一起，图 12-10 是两个 74164 串接在一起的线路范例，我们特地把 74164 的输出直接驱动 LED，可以用目视的方法观察 8051 送出的值。

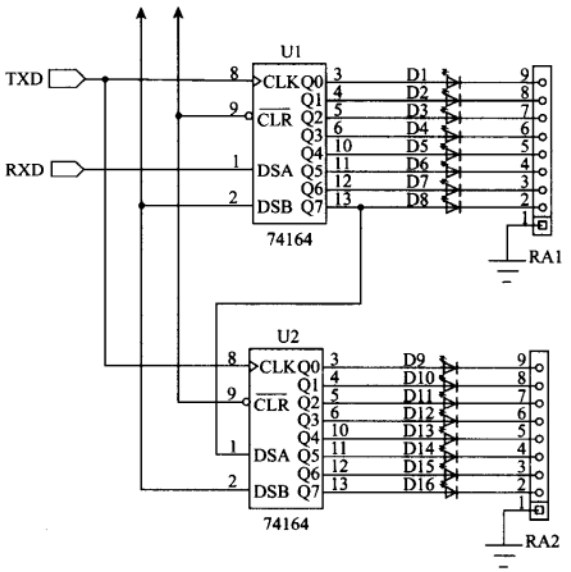


图 12-10　串行输出范例的线路

8051 串行传输模式 0 下，除了发送数据外，还可以接收外部传回来的串行数据，此时，TxD 仍提供 8 个脉冲输出但 RxD 则改成输入，在 TxD 的下降边沿时读入数据，最后组成一个 8 位的数据值。只要把 REN 位设成 1，代表允许接收，并把 RI 位清除成 0，待 TxD 送完 8 个脉冲信号给外部线路后，RI 位自动升成 1，表示数据已进入 SBUF 中，程序应该尽快取走该值。传输模式 0 的接收方式，实际上就是在做输入点（Input）的扩展，每次可扩充 8 个输入点，若外部线路采用串接的接法，扩展的点数就无可限量了。

目的：学习串行模式 0 的传送数据与 74164 接法练习。

程序名称：S2.ASM。

辅助硬件

串行输出线路如图 12-10 所示。

程序操作说明

8051 的串行传输接口通常是连线通信用的，若我们想要利用其他位来做输出入扩展时，就可采用下面程序写法，虽然速度慢了 20 倍，但对一般应用应该足够。

```
;PROGRAM NAME S2.ASM
;TEST SERIAL OUTPUT PORT SIMULATION
;
BUZZER  REG     P3.4
LED     REG     P3.7
;
RESET   MOV     R0,#00H
        DJNZ    R0,$
        CLR     BUZZER
        CLR     BUZZER
;
```

```
        SETB    P3.0            ;RXD=1
        SETB    P3.1            ;TXD=1
AGAIN   MOV     R7,#81H
        MOV     R6,#AAH
        LCALL   SERIAL_OUT
        LCALL   DELAY
        SJMP    AGAIN
;
SERIAL_ OUT
        CLR     C
        MOV     A,R7
        LCALL   SENDBYT
        NOP
        NOP
        NOP
        CLR     C
        MOV     A,R6
        LCALL   SENDBYT
        RET
;
SENDBYT MOV     R0,#08H
SBIT    RRC     A
        JC      OUT1
OUT0    LCALL   OUTLO
        SJMP    SENDNXT
OUT1    LCALL   OUTHI
SENDNXT DJNZ    R0,SBIT
        RET
;
OUTLO   CLR     P3.0
        LCALL   PULSE
        RET
;
OUTHI   SETB    P3.0
        LCALL   PULSE
        RET
;
PULSE   SETB    P3.1
        CLR     P3.1
        NOP
        NOP
        SETB    P3.1
        RET
;
DELAY   MOV     R0,#0
$1      MOV     R1,#0
        DJNZ    R1,$
        DJNZ    R0,$1
        RET
```

12-8 串行传输模式 1 彻底研究

8051 的串行传输模式 1 和模式 0 是截然不同的，模式 1 属于标准的通信模式，只要波特率定义相同时，可以轻易地与其他 PC 系统或设备连接，在此模式下 TxD 专用于串行数据的发送，RxD 则处理外部传过来的串行数据，每次发送和接收都把 10 位的数据当成一个数据帧（Frame），其中包括了 1 个 Start bit 起始位、8 个数据位和 1 个 Stop bit 结束位。只要定义完成 Timer1 的波特率后，就可展开数据收送的操作，当 RI 位成为 1 时，表示已有一个字节被填进 SBUF 中，而由 8051 发送数据时，若 TI 位成为 1 时，表示 10 个位已全数送出，可以再输出下一个字节数据。如果要 CPU 正确且迅速地收送数据时，RI 或 TI 位为真时都可以产生中断要求信号。

AT2051 控制板也是定义串行接口在此模式，并以 9600 b/s 的波特率和 RS485 模式与外部周边连线，所以接下来的范例程序都是以此波特率为准的，至于其他波特率的设置值请参考表 12-5，而 PC 端的连线程序会经常让其 RS485 接口处于接收的状态，所以，我们可以靠这种方便性，随时传送一些值回 PC，以便观察传输上是否有问题。

目的：学习串行模式 1 的发送格式。

程序名称：S3.ASM。

辅助硬件

数字式示波器观察程序产生的同步信号和 TxD 信号。

```
;PROGRAM NAME S3.ASM
;TEST SERIAL PORT(MODE 1)
;
BUZZER      REG     P3.4
LED         REG     P3.7
;
RESET       MOV     R0,#00H
            DJNZ    R0,$
            CLR     LED
            CLR     BUZZER
;
            MOV     TMOD,#00100000B         ;TIMER 1 MODE2
            MOV     TH1, #FDH               ;BAUD RATE=9600B/S
            SETB    TR1                     ;START TIMER 1
            MOV     SCON, #01010000B        ;T/R,1START BIT
                                            ;8BITDATA,1STOP BIT
START       CPL     P1.0                    ;SYNC SIGNAL FOR SCOPE
            CLR     TI                      ;MONITOR TxD FROM SCOPE
            MOV     A,#31H
            MOV     SBUE,A
WAIT        JNB     TI,WAIT
            SJMP    START                   ;IF ANY KEY HIT THEN
;
DELAY       MOV     R0,#00H
            DJNZ    R0,$
            RET
```

程序分别设置 Timer1 为模式 2 自动载入的功能，并且将 TH1 存入 FDH 值，这个值是波特率 9600 b/s 的设置值，接着立即启动 Timer1，串行传输接口则指定工作在模式 1，并且允许接收接口操作，程序只送出 31H 值（即“1”数字），直到在 PC 上按下任一键后才停止，循环中每次都改变 P1.0 的状态一次，这个信号可充当示波器的同步及参考信号，以方便我们看清 TxD 的变化，也比较容易计算时间。

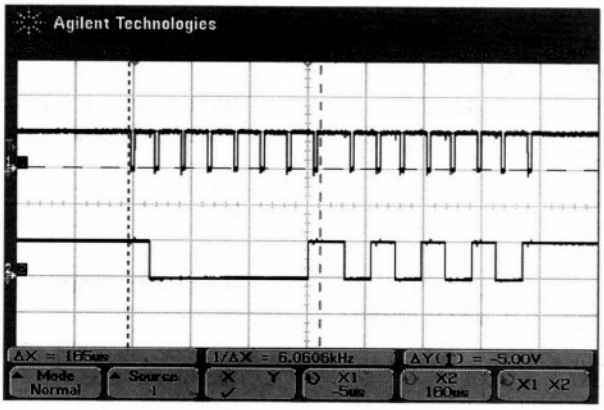


图 12-11　程序每次执行 MOV F5H，A（MOV SBUF，A）的时间间隔为 1.042ms

注：（1）CH1 为 TxD 的串行信号，CH2 为 P1.0 的变化情形。

（2）Δ=104μs，代表每个位所花的时间总共送出 10 个位的串行数据，所以共花了 1.04 ms 的时间。

（3）扣除 START 和 Stop bit 送出的串行正好是 3lH。

图 12-11 列出了每次送值给 SBUF 的时间间隔为 1.040 ms，在 1.040 ms 内共送出了 10 个位的串行数据，所以每个位送出所花的时间=1.040 ms/10=0.104 ms。

换算后的波特率为 $\frac{1}{0.1040\ \text{ms}}$ =9615 b/s（与 9600 b/s 相当接近）

由以上的时间看来，8051 执行的速度较串行传输速度快很多，若程序处理得当，CPU 应该会有足够的时间去送或获取 SBUF 上的值。

目的：学习串行传输模式 1 的接收写法。

程序名称：S4.ASM。

辅助硬件

由 PC 机的键盘输入值。

程序操作说明

程序设置好传输速率后，随即进入接收数据的状态，若 RI 位为 1 时，表示已有一个字符进入 SBUF 中，程序除了读取 SBUF 值外并将此值又回送到 SBUF 端口上，所以可以马上观察传输值是否正确，当 RI=1 时，SCON 寄存器的另一个位 RB8 中是 Stop bit 的状态。当由 PC 的键盘上输入“!”惊叹号时，范例程序才结束接收的操作程序。

```
;PROGRAM NAME S4.ASM
;TEST SERIAL PORT MODE 1 INPUT TEST
;
BUZZER      REG     P3.4
```

```
LED         REG     P3.7
;
RESET       MOV     R0,#00H
            DJNZ    R0,$
            CLR     LED
            CLR     BUZZER
;
            LCALL   INIT_SIO
START       CLR     RI
WAIT        JNB     RI,WAIT             ;WAIT UNTIL RI=1
            MOV     A,SBUF              ;READ A CHARACTER
            NOP
            NOP
            NOP
            NOP
            MOV     SBUF,A              ;OUTPUT TO SBUF
            CJNE    A,#'!',START
;
STOP        SJMP    STOP                ;ENDLESS  LOOP
;
INIT_SIO
            MOV     TMOD,#00100000B     ;TIMER 1 MODE 2
            MOV     TH1,#FDH            ;BAUD RATE=9600 b/s
            SETB    TR1                 ;START TIMER 1
            MOV     SCON,#01010000B     ;ENABLERECEIVE
            RET                         ;1 START,8 BITDATA,1 STOP BIT
```

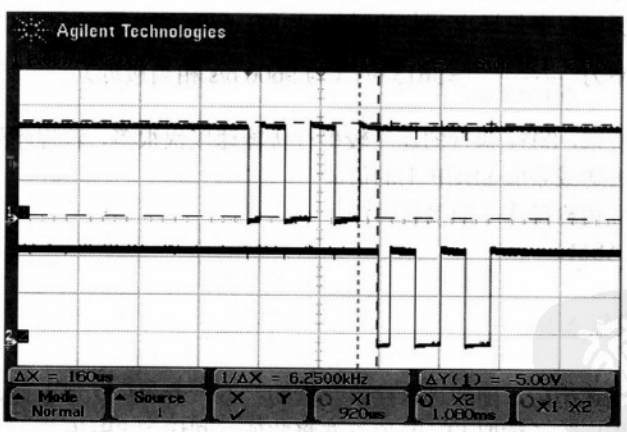


图 12-12　串行模式 1 时，8051 收妥值后立即回应相同值

程序先检查 RI 位，若 RI 位是真时，立即从 SBUF 读回一个值并摆在 ACC 累加器上，然后再度送到 SBUF 寄存器上，随即通过 8051 的串行接口把值回送 PC 端，因此，我们可以在 PC 的屏幕上看到另一个值，这个值正是我们刚刚由键盘上敲入的值，例如我们由键盘上输入“1”，则 AT2051 回应“1”，当输入“2”时则回应“2”值，其余操作以此类推。

在此之前我们曾经提过，8051 执行一个指令的时间较其串行接口传送 1 个字节要快 500

倍以上，前者是 μs 的等级，后者则是 ms 的等级，依此观点我们来看看本程序 S4.ASM，程序大部分的时间都花在 RI 位的检查上，在实际的应用场合里，这种写法是行不通的，因为花了太多时间在 RI 位的检查上，其他事情反而没空去做，所以串行程序在实际应用中，都是利用程序中断的方式来做串行数据的收与发，这样 CPU 才有其他时间去做主程序的控制，由于 RI 和 TI 都可产生中断要求，所以中断程序在执行前需要做这两个位的判断，然后才真正开始执行中断服务程序（ISR）。

目的：学习串行传输模式 1 时，中断服务程序的写法。

程序名称：S5.ASM。

辅助硬件

PC 机的屏幕与键盘。

```
;FILENAME S5.ASM
;
BUZZER  REG     P3.4
LED     REG     P3.7
;
        ORG     0000H
        LJMP    RESET
        ORG     0023H
        LJMP    SIOISR
;
RESET   MOV     R0,#00H
        DJNZ    R0,$
        CLR     LED
        CLR     BUZZER
;
START   MOV     SP,#3FH
        LCALL   INIT_SIO
STEP1   MOV     20H,#00H
        CLR     F0
STEP2   SETB    REN             ;SET RECEIVE ENABLE
STEP3   SETB    PS              ;SERIAL INTERRUPT HIGHER PRIORITY
        SETB    ES              ;ENABLE SERIAL INTERRUPT
STEP4   CLR     RI              ;CLEAR RI
        CLR     TI              ;CLEAR TI
STEP5   SETB    EA              ;ENABLE SYSTEM INTERRUPT
;
MAIN                            ;MAIN ROUTINE START
        INC     20H
        LCALL   DELAY
        JNB     F0,MAIN         ;LOOP AGAIN
;
STOP    SETB    BUZZER
        SJMP    STOP            ;DISABLE SERIAL INTERRUPT
;
SIOISR  JNB     RI,ISREND
        MOV     A,SBUF          ;READ CHARACTER FROM SBUF
        CLR     RI
```

```
        CLR     TI
        MOV     SBUF,A          ;ECHO BACK
        CJNE    A,#'!',ISREND
        SETB    F0
ISREND RETI
;
INIT_SIO
        MOV     TMOD,#00100000B ;TIMER 1 MODE 2
        MOV     TH1,#FDH        ;BAUD RATE=9600 BPS
        SETB    TR1             ;START TIMER 1
        MOV     SCON,#01010000B ;ENABLE RECEIVE
        RET                     ;1 START,8 BITDATA,1 STOP BIT
;
DELAY   MOV     R0,#04H
        MOV     R1,#00H
DLY1    DJNZ    R0,$
        DJNZ    R1,DLY1
        RET
```

本程序有关于中断设置的程序部分请参考相关章节中关于中断彻底研究内的说明，当 RI=l 时 CPU 会先确认串行传输的中断要求，然后跳到 SIOISR 处开始执行中断服务程序，这里的写法就和原先用询问 RI 的方式相同，取回 SBUF 上的值并立即清除 RI 和 TI 位，以便下次再度产生中断。当收到的字符是“!”时，程序会启动蜂鸣器代表这个示范程序已经终止。

12-9 串行传输模式 2 彻底研究

当 8051 的串行接口被设置成模式 2 时，每次传送或接收 11 个位的串行数据，其中包括了 1 个 Start bit、8 个数据位、1 个可程序设置的位（TB8 或 RB8 位）和 1 个 Stop bit，在此模式下传输的速率只有两种选择，这是由 PCON 寄存器上的 SMOD 位控制，当 SMOD=0 时，波特率等于系统的石英晶体频率除以 64，而 SMOD=1 时波特率为系统频率除以 32，这两个频率都不是 RS232C 所适用的传输频率，当数个系统都是以 8051 为 CPU 时，就可以采用此模式以减少传输时间。以 AT2051 使用 11.059MHz 的石英晶体为例，SMOD=0 时的传输率约为 173 kb/s，而 SMOD=1 时的传输速率为 345 kb/s，速度已经相当快了，我们用的 75176 串行差动 IC 还是可以承受如此高速的传输频率的。

目的：观察串行传输模式 2 的发送波形（SMOD=0）。

程序名称：S6.ASM。

辅助硬件

数字式示波器分别观察 P1.0 和 TxD 脚。

```
;PROGRAM NAME S6.ASM
;TEST SERIAL PORT (MODE 2)
;
BUZZER      REG     P3.4
LED         REG     P3.7
;
```

```
            ORG     0000H
RESET       MOV     R0,#00H
            DJNZ    R0,$
            CLR     LED
            CLR     BUZZER
;
            MOV     SCON,#10010000B     ;T/R,1START BIT
                                        ;8 BITDATA,1 TB8,1STOP BIT
            MOV     PCON,#00H           ;SMOD=0
START       SETB    P1.0                ;SYNC SIGNAL FOR SCOPE
            CLR     TI                  ;MONITOR TxD FROM SCOPE
            SETB    TB8                 ;BIT7=1
            MOV     A,#31H              ;ASCII'1'
            MOV     SBUF,A
WAIT1       JNB     TI,WAIT1
            CLR     P1.0                ;SYNC SIGNAL FOR SCOPE
            CLR     TI
            CLR     TB8                 ;BIT7=0
            MOV     A,#31H
            MOV     SBUF,A
WAIT2       JNB     TI,WAIT2
            SJMP    START               ;RE-START
;
```

由于此模式不需要设置波特率，所以只要设置好 SCON 寄存器的值后，就可以开始发送与接收，在本程序中我们分别示范了 TB8（第 10 个传送位）的清除与设置方法，较正确的程序是先设置 TB8 的值，然后才对 SBUF 填入发送值，当允许接收且收妥 11 笔串行数据时，8 个数据位存放在 SBUF 中，而 SCON 寄存器中的 RB8 位内存放第 10 个接收到的位。

当本程序执行时，可从示波器的屏幕上观察到 TxD 的变化情形：而 P1.0 信号是供参考及同步用，图 12-13 分别显示 P1.0、8051 的 TxD 端，每个位送出的时间约为 6μs，取倒数等于 166 kb/s，和前面计算值 173 kb/s 相差不远。

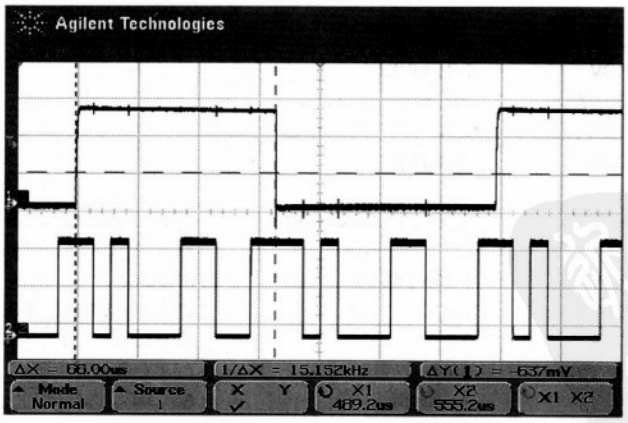


图 12-13　P1.0 与 TxD 的信号(SMOD=0)

注：CH1 P1.0 供示波器同步及参考用，CH2 8051 的 TxD 输出端。

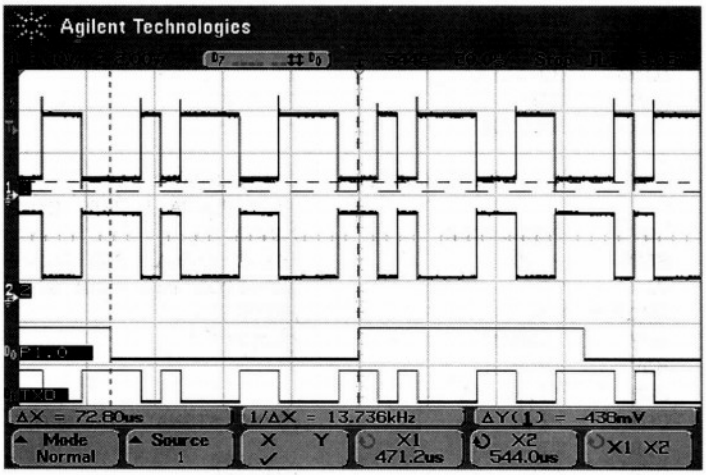


图 12-14　观察 75176 的输出 SIO 与 $\overline{SIO}$

注：170kb/s 下的输出方波仍然很漂亮。

目的：观察串行模式 2 的发送波形（SMOD=1）。

程序名称：S7.ASM。

辅助硬件

数字式示波器观察 TxD 输出。

```
;PROGRAM NAME S7.ASM
;TEST SERIAL PORT(MODE 2)
;
BUZZER      REG         P3.4
LED         REG         P3.7
;
            ORG         0000H
RESET       MOV         R0,#00H
            DJNZ        R0,$
            CLR         LED
            CLR         BUZZER
;
            MOV         SCON,#10010000B     ;T/R,1START BIT
                                            ;8 BIT DATA,1TB8,1STOP BIT
            MOV         PCON,#80H           ;SMOD=1
START       SETB        P1.0                ;SYNC SIGNAL FOR SCOPE
            CLR         TI                  ;MONITOR TxD FROM SCOPE
            SETB        TB8                 ;BIT7=1
            MOV         A,#31H              ;ASCII'1'
            MOV         SBUF,A
WAIT1       JNB         TI,WAIT1
            CLR         P1.0                ;SYNC SIGNAL FOR SCOPE
            CLR         TI
            CLR         TB8                 ;BIT7=0
```

```
        MOV     A,#31H
        MOV     SBUF,A
WAIT2   JNB     TI,WAIT2
        SJMP    START           ;RE-START
;
```

程序操作说明

本程序和上一个程序完全相同，只有 PCON 寄存器中的 SMOD 位改设成 1，此时的传输速率约为 345 kb/s，已远远超过 RS232C 的传输速率极限，8051 的 TxD 信号仍准确地送出，到了 75176 后仍然是生龙活虎似的。详细信号变化情形，如图 12-15 所示。

模式 2 下的接收方式完全与模式 1 相同，当 RI=1 时 CPU 就可以读取 SBUF 上的内容，另一个传输位则存在 SCON 寄存器的 RB8，这个位可供做奇偶校验用（Parity Check），或者是做多处理器通信用，这方面的技巧我们将在串行传输模式讲完后再探讨。

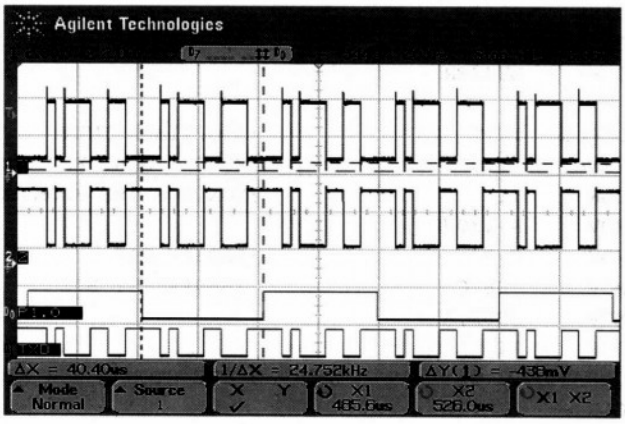


图 12-15　观察 75176 的输出 SIO 与 $\overline{\text{SIO}}$

注：340kb/s 下的输出方波还很漂亮。

12-10　串行传输模式 3 彻底研究

串行传输模式 3 和模式 2 几乎相同，但其波特率是靠 Timer1 来决定，当 CPU 是 8052 或 8032 时，也可靠另一个新增的 Timer 2 做波特率的调整，模式 3 的发送和接收也含有 11 个位，包括 1 个 Start bit、8 个数据位、1 个程序可设置或清除的位（TB8 或 RB8）和 1 个 Stop bit。发送前先清除 TI，然后将发送值填入 SBUF 中，当串行接口传送完 11 个位后，会自动将 TI 位设成 1，程序只要检查 TI 位就可判定是否能送下一个数据给 SBUF。发送时的 TB8 可作为同位检查用或者是做多处理机连线通信用，TB8 必须在 SBUF 被填入值前设置。若把 SCON 寄存器的 REN 设成 1 时，串行接口还可同时做接收的操作，当 SBUF 收妥 11 个位值后，会将 RI 位设成 1，第 10 个接收位摆在 RB8 位上，CPU 应全部将接收数据取走，以免另一组数据进来后，覆盖掉原有的值。

8051 规划此模式主要是做 RS232C 通信及多处理机通信用，前者可加入同位检查以保证

每次接收都是正确值，后者则可指定传送的数据是数据或是地址值。

目的：观察串行模式 3 的发送波形（9600 b/s）。

程序名称：S8.ASM。

辅助硬件

双轨示波器观察 P1.0 和 TxD 的输出波形。

```
;PROGRAM NAME S8.ASM
;TEST SERIAL PORT(MODE 3)
;
BUZZER      REG     P3.4
LED         REG     P3.7
;
            ORG     0000H
RESET       MOV     R0,#00H
            DJNZ    R0,$
            CLR     LED
            CLR     BUZZER
;
            LCALL   INIT_SIO            ;SIO,TIMER INIT
START       SETB    P1.0                ;SYNC SIGNAL FOR SCOPE
            CLR     TI                  ;MONITOR TxD FROM SCOPE
            MOV     A,#30H
            MOV     C,P                 ;PARITY BIT
            MOV     TB8,C               ;TB8=PARITY BIT
            MOV     SBUF,A
WAIT1       JNB     TI,WAIT1
            CLR     P1.0                ;SYNC SIGNAL FOR SCOPE
            CLR     TI
            MOV     A,#32H
            MOV     C,P                 ;PARITY BIT
            MOV     TB8,C               ;TB8=PARITY BIT
            MOV     SBUF,A
WAIT2       JNB     TI,WAIT2
            SJMP    START               ;RE-START
;
INIT_SIO
            MOV     TMOD,#00100000B     ;TIMER 1 MODE 2
            MOV     TH1, #FDH           ;BAUD RATE=9600 b/s
            SETB    TR1                 ;START TIMER 1
            MOV     SCON, #11010000B    ;T/R,1START BIT
                                        ;8BITDATA,1TB8,1 STOP BIT
            RET
```

程序操作说明

本程序设置的波特率是 9600 b/s，但是每笔串行数据有 11 个位，开始执行程序后，您猜在 PC 的屏幕上会有什么结果呢？

当程序执行时，我们可以在示波器上看到稳定且清晰的波形，图 12-15 是由逻辑分析仪

上看到 30H 值被送出去的时序图，8051 每次指令执行完毕后，会将累加器 ACC 的同位值填入 PSW.0（即 Parity 标志位）上，当 ACC 中有奇数个 1 时，Parity 标志位就等于 1，反之 Parity 标志位等于 0，由于 30H 转换成二进制表示时为 00110000B，所以其 Parity 标志位应该是 0，正如图 12-16 所示，传送 11 个位要花 1.150 ms，平均每个位为 104μs，换算后的波特率是 9615 b/s（程序定义为 9600 b/s）。

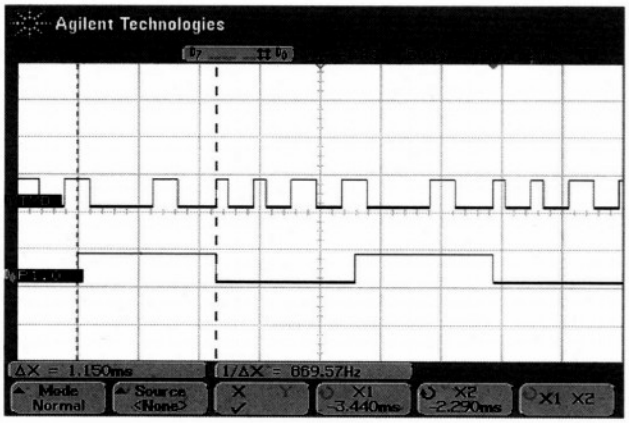


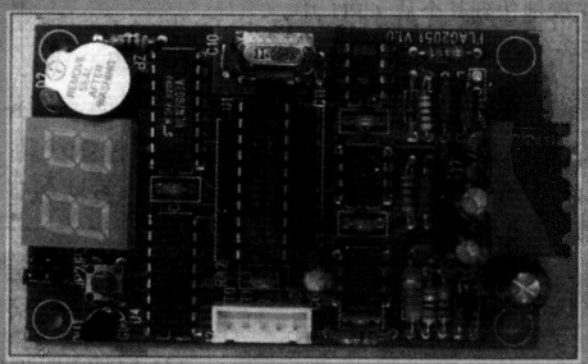
图 12-16　P1.0 和 TxD 的波形

当程序执行时我们在 PC 的屏幕上看到“0”和“2”两个数字交替显示出来，看起来似乎 PC 端并没有做 Parity Check，它只判断串行数据的前 10 个位，然后取出中间的 8 个位数据做显示，所以才看到我们送出的“0”和“2”两个数字。

习　　题

1. 并行传输与串行传输的差异是什么？试比较其优缺点。
2. 在做串行通信时，有哪些技术上的问题是必须克服的？
3. 传输过程的正确与否，通常可用哪些方法来判断？
4. 什么是 RS232C？
5. 串行传输的模式共有几种？试简述各模式的原理及应用。
6. 串行传输控制所使用的寄存器是什么？其可否作为位寻址？
7. 什么是波特率？
8. 串行传输的波特率如何设置？

13



我们可以把多块 AT2051 控制板并联起来，通过 RS485 接口传送出各自的温度测量值

第 13 章　8051 串行通信彻底研究（二）

8051 的串行通信里还可以支持多处理机通信，本章就是讨论这方面的技巧。不过，若连接的对象是 PC 时，9600 N81（9600 b/s，No parity，8 bits data，1 stop bit）反而是最常用的通信协议，所以，在此我们会尽量举 PC 的连接实例，因为 PC 还是我们身边最常用且最方便取得的外部设备，当然你也可以用两块 AT 2051 控制板做串行连接实验。

13-1　8051 的多处理器通信彻底研究

8051 串行传输模式 2 和模式 3 宣称可以做多处理器通信（Multiprocessor Communication），其中的重点正在 SCON 寄存器的 SM2 位和 TB8 及 RB8 位上，这几个位就决定了多处理器通信的核心，至于波特率的问题，只要收送双方都能接受即可，图 13-1 是一个典型多处理器通信的连接示意图。

多处理器通信和 RS232C 的 1 对 1 通信最大的差异是多了一个地址传送码，1 对 1 通信时，我送数据就轮到您接收，而您送数据时就换成我接收，完全没有商量的余地，可是多处理器通信时，主 CPU 发出的信息如何正确地传到某个从 CPU 中呢？针对这个问题，在多处理器系统中每一个从 CPU 都有其特定的编号（也有人称为 ID），在数据传输前每个从 CPU 都处于待命的状态，当主 CPU 指定到某个特定的从 CPU 时，该从 CPU 才开始接收或发送数据，而这项指定特定从 CPU 的操作，实际上就是送出地址 ID 值，每个从 CPU 上的程序收到地址后，会先判断一下是否叫到要启动，若属实时则开始执行启动的所有程序。接下来我们用步骤说明的方式，详细介绍多处理器通信的程序。

在多处理器通信中我们一直提到“地址传送”，在串行传输时，我们如何去辨别地址值

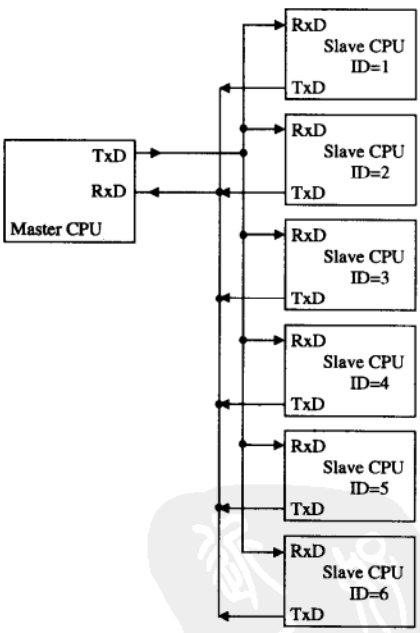


图 13-1　多处理机通信的方式

注：（1）主 CPU 的 RxD 与各从 CPU 的 TxD 相接。
（2）主 CPU 的 TxD 与各从 CPU 的 RxD 相接。
（3）主 CPU 发出的串行数据各从 CPU 都收得到。
（4）各从 CPU 送出的信息都可在主 CPU 上收到。
（5）此处的硬件连接方式已非 RS232C 的架构。

（Address）和数据（Data）呢？这正是 8051 串行传输模式 2 和模式 3 的魅力所在，当 8051 工作在以上两个模式时，除了数据占 8 位外，传输时多了一个 TB8 位，接收时则多了一个 RB8 位，之前的范例程序中都以此位当成同位检查用，但在多处理器通信时，当 TB8=1 代表正传送一个地址值，该地址值有 8 位宽，所以理论上可连接 256 个 CPU 在同一个通信系统上，TB8=0 时代表正在传送一个数据值，在此串行传输线上，每笔串行数据共有 11 个位，但其中以第 10 个位的状态来区分地址或数据，接收端可依此格式立刻判断出其中的差异来。

主 CPU（Master）端		数据流向	从 CPU（Slave）端
传送步骤 1	执行主程序		执行主程序，但进入接收地址模式
传送步骤 2	送出地址值，准备再送出数据	→	接收地址值，并判断是否被指定到要收数据
传送步骤 3	数据处理		若是则再进入数据接收模式，若不是则重新回接收地址模式
传送步骤 4	送出所有数据	→	接收所有数据
传送步骤 5	回步骤 1		回步骤 1

主 CPU（Master）端		数据流向	从 CPU（Slave）端
传送步骤 1	执行主程序		执行主程序，但进入接收地址模式
传送步骤 2	送出地址值，准备开始接收数据	→	接收地址值，并判断是否被指定到要传送数据
传送步骤 3	接收数据操作开始		若是则开始送出数据若不是则回步骤 1
传送步骤 4	接受所有数据	→	送出所有数据
传送步骤 5	回步骤 1	←	回步骤 1

图 13-2　多处理器的通信步骤说明

在从 CPU 上通常以串行中断的方式来对待进来的数据，若把 SCON 寄存器的 SM2 位设置成 1 时，即允许 8051 的串行接口做多处理器的地址判断，在这个模式下：

CPU 只在收到地址值（其 RB8=1）时才产生中断要求。

在串行中断的服务程序上，只要由 SBUF 内读回地址值，再做一个判断是否被调用，若是时则跳去执行程序设置的操作，不是时则不做任何事，随即结束此中断服务程序回主程序，被指定到的从 CPU 先设成 SM2，然后开始发送或接收数据，而其他的从 CPU 并未启动传输，所以此一瞬间仍然是 1 对 1 的通信，当主 CPU 最初在发送地址阶段，却是 1 对多的通信。图 13-1 的连线规格可否沿用 RS232C 呢？答案是否定的。

当主 CPU 传送数据给各从 CPU 时，只有一个发送其他接收的情况下是可使用类似 RS232C 的架构，可是当主 CPU 欲收数据而其中只有一个从 CPU 发送时，在图中标示 A 处会造成数据电平冲突，使得主 CPU 无法由 RxD 脚上得到正确值，因此，在实际应用上该采用另一种接口传输规格 RS422A，这种方式只有传输的硬件上稍有不同而已，程序上的串行程序都无需做任何修改，RS422A 传输线数只有 4 条（+Tx、-Tx、+Rx 和-Rx），其连接方式如图 13-3 所示。

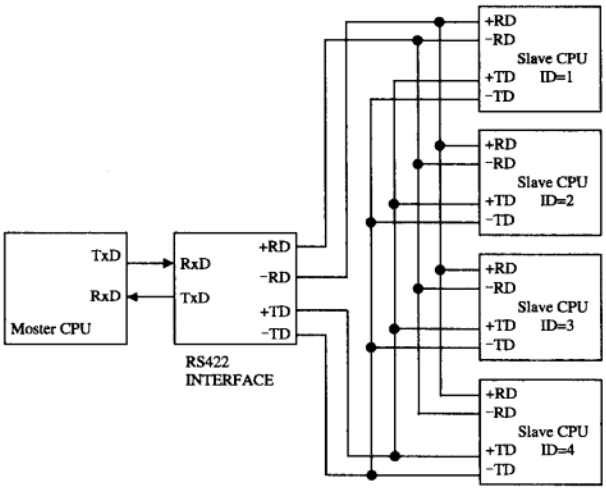


图 13-3　以 RS422A 接口连接的多处理器通信

注：（1）传输线只有 4 条。
（2）主 CPU 与各从 CPU 的地端不需连接。
（3）RS422A 利用差动的方式收发数据，传输距离较远，速度也可较 RS232C 快。
（4）一个发送点最多可接 10 个接收端。

另外一种常用的多处理器传输的接口是 RS485 接口，只要两线式的双绞线（Twisted Pair Line）就可将多个 CPU 连线在一起，这种做法使得数据共享的可能性大为提高，请看图 13-4 的 RS485 连接图，当多处理器以 RS422A 接口连接时，一定有一个主 CPU（Master CPU）和多个从 CPU（Slave CPU），若想作从 CPU 间的数据传输时，所传输的路径是从 CPU 到主 CPU，然后由主 CPU 再送给另一个从 CPU，若采用图 13-4 的 RS485 架构时，只要传输线空闲，任一两个 CPU 都可相互传收数据，完全没有主从（Master/Slave）之分，若以此架构再做发展时，就形成现在我们常用的 ARCNET 或 ETHERNET 的 LAN 网络了。

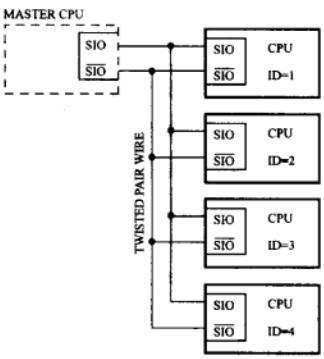
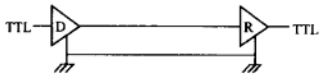


图 13-4　以 RS485 接口所连接的多处理机通信

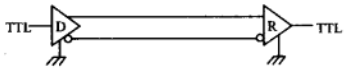
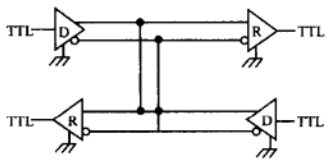
注：（1）传输线只剩两条。
（2）发送/接收端以此双绞线传输。
（3）传输时性较 RS422A 更佳。
（4）最多可以有 32 个发送/接收点。

表 13-1 列出我们这章所谈到的 3 种串行传输接口的接法与比较表，8051 的串行接口完全可以配合这些接口，很方便地与其他设备电脑成 CPU 连线。

表 13-1　3 种数据传输（Data Transmission）的标准及比较

	RS232C（1 个发送对 1 个接收）	使用接口 IC
非平衡式	 Logic 1=−5～−15V 传输线长（MAX）50ft Logic 0=+5～+15V 传输率 50kb/s	SN75188 SN75189 MAX232 ICL232

续表

平衡式	RS422A（1 个发送对 10 个接收）  Logic 电平+2V 差动输出入　传输线长　传输率 12.192m（40ft）　10Mb/s 121.92m（400ft）　1Mb/s 1219.2m（4000ft）　100kb/s	SN75157 SN75158 AM26LS31 AM26LS32
平衡式	RS485（32 个发送对 32 个接收）  Logic 电平+1.5V 差动输出入　传输线长　传输率 12.192m（40ft）　10Mb/s 121.92m（400ft）　1Mb/s 1219.2m（4000ft）　100kb/s	SN75176 SN75172 SN75173 SN75174 SN75175

13-2　AT2051 的串行硬件线路分析

知道了 3 种最常用的串行传输方式之后，请再来看看本书主角 AT2051 控制板的 RS485 通信接口（见图 13-5），我们只用了两颗 IC（75176 与 NE555）就完成 RS485 的接口电路，实际的串行传输重任是 75176 来执行的，而 555 则是做单击的任务，图右侧的电阻与二极管

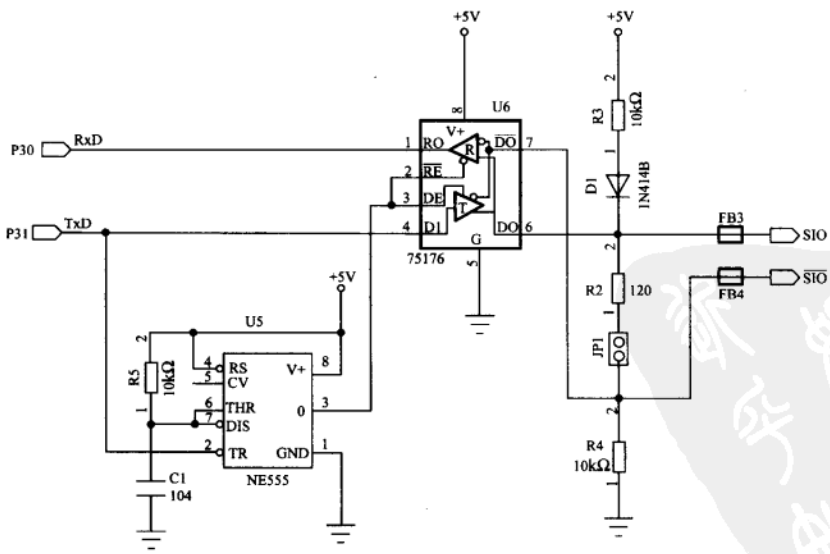


图 13-5　AT2051 控制板的 RS485 接口

的作用是在维持 RS485 的传输信号电平，即 SIO 的电平一定要比/SIO 要高，以免接收串行数据时内部差动电路误判。

RS485 只用两条线就完成串行数据的传送与接收，所以在传输电路设计上有一个重点，那就是随时要把线路空出来给别人使用，而这个任务就是由 555 来负责的，当 8051 有一个字节要从 P3.1（TxD）送出时，记得吗？8051 串行数据第一个位一定是 0 的 Start bit，这个数字状态 0 的信号也刚好触发 555 的单击，我们特地把 555 内单击的时间定为 1 ms，而 555 的输出端直接掌控 75176 的驱动使能脚（Drive Enable），也就是 555 输出 1 时正是 75176 切换成输出驱动的模式，而且此模式最多只维持 1 ms，之后又会自动回到接收的状态，把传输线的控制权交回。1 ms 的时间恰好是传输速度 9600 b/s 下传递 1 字节所需要的时间。

有些 RS485 的电路中，其 75176 的第 2 与第 3 脚是另行用硬件配合软件来控制的，也就是说直接用程序来定义 75176 的收与送的状态，采用这个方法并非不行，不过一定要避免电路板死机或故障，进而使得整个 RS485 传输线停滞的情形发生。

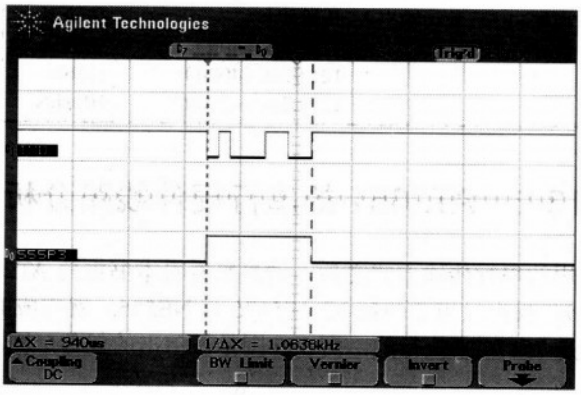


图 13-6　AT2051 控制板从串行端口送出一个字节的波形例

注：CH1 TxD (P3.1)　CH2 NE555 one-shot 输出。

13-3　AT2051 控制板如何与 PC 连接

为什么 AT2051 控制板的串行接口要是 RS485？这样不是反而增加困扰!这是一个非常具有思考性的问题。1992 年旗威科技公司推出了 FLAG51 控制板，使用 25P 的串行传输接口，可以随时下载程序进行试验，FLAG51 与 PC 做一对一的沟通绝对是很方便的，可是要多块 FLAG5l 对 PC 时就有困难了，除非我们去买 4 端口或 8 端口的串行通信卡。另一方面许多新的笔记型电脑已经没有 COM 串行通信端口，且操作系统都是微软的 Windows，许多实验很难在原有的 DOS 环境下运作。我们了解：在 8051 本身是支持多微处理器串行通信的，许多应用是要多个 8051 同时工作并且互相交换信息。为了示范相关的实验与应用，我们才把 AT2051 控制板的串行通信接口改为 RS485，不论一台或多台都可以很方便地与 PC 连线通信，在 PC 上只要外加一块串行通信转换板，把信号改为差动的 RS485 电平即可，而对 PC 而言其面对的都仍是串行的 COM 端口。

在这里我们要示范两种与 AT2051 控制板连接的方法。

方法 1：RS232 转 RS485 转换板。

在 PC 上通过 COM1 或 COM2 端口与 AT2051 控制板，图 13-7 是这块转换板的外观，如果你的电脑只有 COM 通信端口，可以用这块转换板与 AT2051 控制板相连。控制软件程序就属 DOS 版最为方便，在 Windows 环境下可以用 VB 调用 MSCOMM 对象也可以顺利控制 COM 端口。

方法 2：USB 转 RS485 转换板。

如果电脑只有 USB 的扩充端口，那么你可以用 USB 转 RS485 转换板去叫到 AT2051 控制板。图 13-8 为其实际照片。由于 USB 的驱动程序几乎都是 Windows 版的，所以你仅能在视窗的环境内与 AT2051 控制板沟通，在本书的范例程序中，我们会提供适当的连接程序，让你可以观看连接的内容与结果。

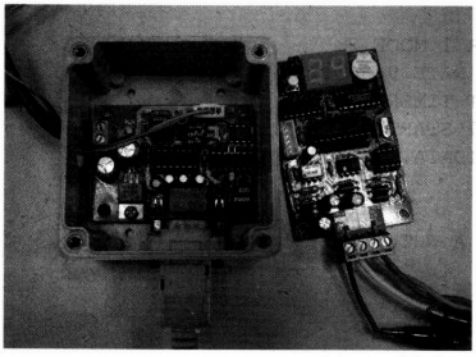


图 13-7　RS232 转 RS485 转换板

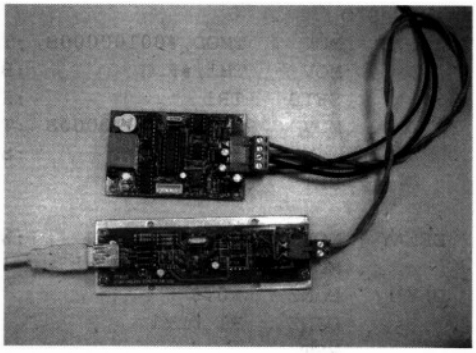


图 13-8　USB 转 RS485 转换板

13-4　多处理器通信的写法分析

目的：学习多处理器通信的接收程序。

程序名称：S9.ASM。

程序操作说明

```
;PROGRAM NAME S9.ASM
;TEST SERIAL PORT (MODE 3)
ADDR    EQU     27H
STOP    EQU     00H
BUZZER  REG     P3.4
LED     REG     P3.7
;
        ORG     0000H
RESET   MOV     R0,#00H
        DJNZ    R0,$
        CLR     LED
        CLR     BUZZER
;
START   LCALL   INIT_SIO        ;SIO,TIMER INIT
```

```
        CLR     RI
        MOV     P1,#00H          ;CLEAR P1 PORT
        SETB    SM2              ;MULTIPROCESSOR COMMUNICATION
WAIT    JNB     RI,WAIT          ;WAIT UNTIL RI=1
        MOV     A,SBUF           ;READ ADDRESS
        CLR     RI
        CJNE    A,#ADDR,WAIT     ;ADDRESS CHECK
        CLR     SM2              ;SM2=0,READ DATA MODE
DWAIT   JNB     RI,DWAIT
        MOV     A,SBUF           ;READ DATA
        CLR     RI
        SETB    SM2              ;MULTIPROCESSOR COMMUNICATION
        CJNE    A,#STOP,WAIT     ;WAIT ANOTHER CODE
        SJMP    START            ;RE-START
;
INIT_SIO
        MOV     TMOD,#00100000B  ;TIMER 1 MODE 2
        MOV     TH1,#FDH         ;BAUD RATE=9600 BPS
        SETB    TR1              ;START TIMER 1
        MOV     SCON,#11010000B  ;T/R,1 START BIT
                                 ;8 BITDATA,1 TB8,1 STOP BIT
        RET
;
DELAY   MOV     R0,#00H          ;WAIT A WHILE
        MOV     R1,#00H
DLY1    DJNZ    R0,$
        DJNZ    R1,DLY1
        RET
;
```

这个程序范例在说明并示范 8051 接收端程序的写法，AT205l 的程序先设置 SM2=1，处于地址接收的状态，并随时将收到地址值（8 位），假设地址值正好是 27H 时（27H 值为该控制器的地址值），程序就设置 SM2=0，准备接收下一个串行信息，但此信息将被视为数据，然后程序又回到接收地址的状态，当接收到的数据是默认的 STOP 码（00H）时，程序才算是结束。类似的串行接收及判断程序可以放进串行中断的服务程序中，这样的安排会使程序有足够的时间去处理其他子程序和其他无关串行接口的操作。

13-5 8051 串行接口发送硬件分析

8051 的串行接口的工作模式共有 4 种，除了模式 0 是做 I/O 扩展用途外，其余 3 种模式都在执行串行传输的操作，当 Intel 第一次公布 8051 的使用手册之时，并未对串行接口的硬件做详细的介绍，仅仅用程序及文字说明，简单地交待了这些模式的操作。Intel 的工程师们认为开发 8051 的软件或硬件系统，不一定要完完全全理解内部串行接收的情形，可是，有许多设计工程师无法理解这种观点，希望 Intel 能进一步说明串行接收的实际运行情况，所以在下一版 8051/8052 共享的使用手册中，就加入了串行接收的内部框图，还有数页密密麻麻的操作叙述，这下可好了，看了这些说明后，反而对串行接口的操作更模糊不清，除了写手册的人搞不清楚外，照着原文翻成中文的 8051 著作更是语义不详，俗话说，越描越黑，大概就

是这个道理。

在本书中我们想尽量避开这些症结，只针对串行传输模式 1 详细说明，其他模式的操作原理大致类似，也可参考 Intel 的英文版使用手册，以获得进一步的说明。模式 1 下每次共传输 10 个位的数据，前后各包括 1 个 Start bit 和 Stop bit，中间才是 8 个数据位（Data bit）。当我们设置波特率是 9600 b/s 时，必须将 Timer 1 设置成可自动载入的计时模式 2，TH1 上要填入的值是 FDH，所以 Timer 1 的实际溢位率为 11.059MHz÷12÷(256−253)=307.2 kHz。此溢位率的脉冲信号并未有直接供应给串行接口的波特率发生器使用，当 PCON 寄存器上的 SMOD 位是 0 时，上述的脉冲先做除 2 的操作，然后再除 16 后的脉冲才是其传输通信的波特率。

```
307.2kHz÷2÷16=9600Hz(SMOD=0 时)
307.2kHz÷1÷16=19200Hz(SMOD=1 时)
```

最后的脉冲信号分别送到发送单元和接收单元上，请看图 13-9 所示发送线路框图。对 8051 的串行传输发送硬件而言，只要定义好波特率和传输模式，任何对 SBUF 写入值的操作，都视同启动发送，内部的硬件将逐一把串行数据送到 TxD 脚上，其传送详细步骤如下：

（1）类似 MOV SBUF，A 的指令执行完后，发送的 SBUF 寄存器上已备妥欲传输的值了，这里的 SBUF 实际上是一个移位寄存器。

（2）图 13-9 的 Tx 控制单元送出 SEND 信号，开始把 Start bit 送出。SEND 信号持续 1 个位的时间即 1/9600s，约 10 μs 左右。

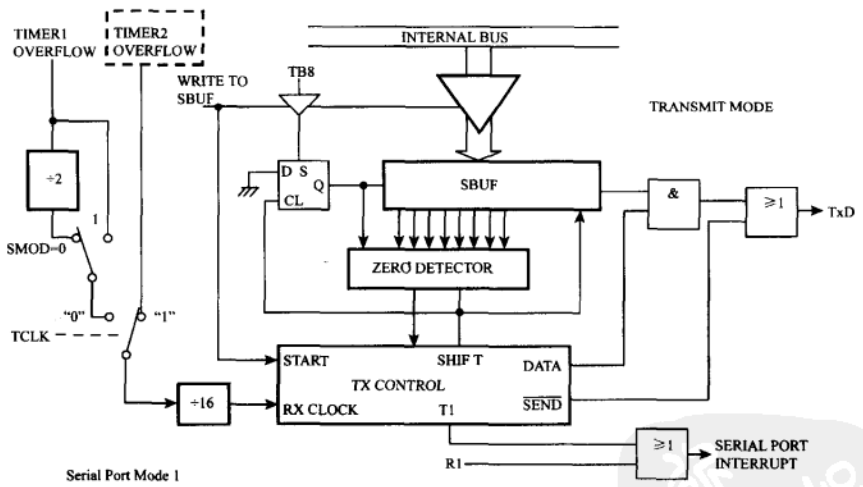


图 13-9　串行发送线路单元图（模式 1）

注：（1）输出 SBUF 本身是一个可一次载入 8 位，且能做右移的移位寄存器。

（2）Tx 发送控制单元控制发送串行的操作，SEND=0 开始发送，SEND=1 时停止发送，DATA=1 时代表正在传送数据。

（3）图中的 SHIFT 总共送出 9 个脉冲给输出 SBUF。

（4）程序在发送数据前一定要检查 TI 位，否则会使 Tx 发送控制单元错误操作。串行接口的接收操作就比发送复杂多了，接收电路共采用了两个脉冲输入，其频率分别是 9600Hz 和 153600Hz，前者是波特率使用，后者供给 RxD 取样用，其取样率始终是波特率的 16 倍，当我们把 SCON 寄存器中的 REN 位设成 1 时，串行接收电路被启动，准备开始接收 RxD 上的数据，正常情况下，RxD 应始终保持在 1 的状态。

（3）数据位中的 LSB 由 SBUF 移出到 TxD 上，又过了 104μs 时间后，Tx 控制单元送出第一个 SHIFT 信号，让 SBUF 右移一次，每次右移时，SBUF 的 MSB 被填入 0。

（4）每隔 104μs 时间，SBUF 右移一次共移出 8 个数据位，此时 SBUF 上已全部是 0 了，此时会触发图中的零值检侦电路，此电路会通知传输快结束了。

（5）Tx 控制单元做完最后一次移出信号后，SEND 信号回到 1 结束输出，此时的 TxD 变成 1，刚好是传输的最后一个 Stop bit。

（6）又过了 104μs 后，Tx 控制单元将 TI 位设成 1，完成 10 个位数据的传送，整个过程共耗了 104μs×10=1.04 ms 的时间。

接下来是接收步骤的详细说明，请参考图 13-10 的接收线路框图。

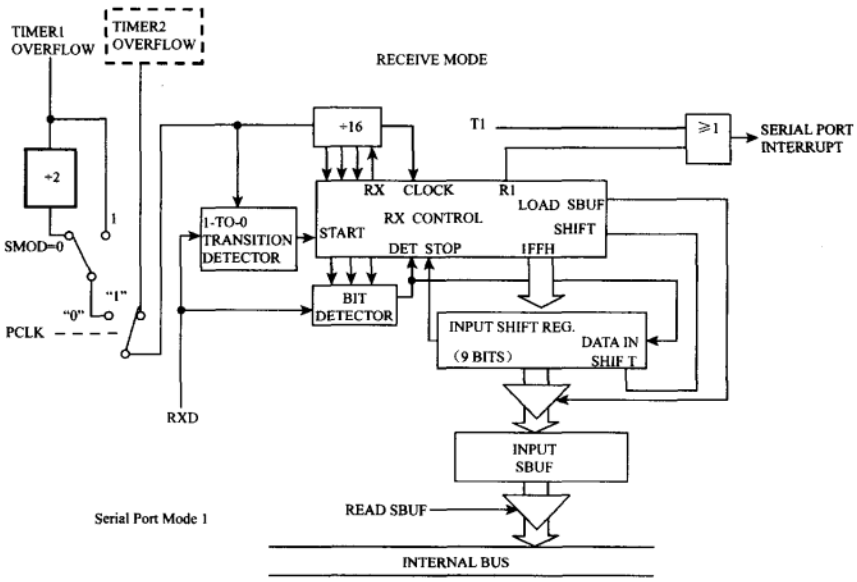


图 13-10　串行接收线路单元图（模式 1）

注：（1）接收 Rx 控制单元必须 RI=0，才会将接收数据由输入移位寄存器上载入 SBUF 中。

（2）RxD 输入端开始由 1 变成 0 时，会启动侦测线路，并通知接收 Rx 单元开始操作，并事先对输入移位寄存器载入一个 9 位值（1FFH）。

（3）Rx 接收控制单元中的 SHIFT 共送出 10 个脉冲，最后的一个脉冲使得输入移位寄存器中的 Start bit 移出，刚好停止接收的操作。

（1）当 RxD 的状态由 1 变成 0 时，图中的状态侦测电路会立即操作，马上把图中的 4 位（除 16）计数器 Reset 归零，并且通过 Rx 控制单元把 1FFH（共 9 位）值填入输入移位寄存器中。

（2）4 位计数器共有 16 个状态值，接收电路只在第 7、8 和 9 状态中对 RxD 端取样，共得到 3 个状态值，并以取样的多数值当成该位的值。

（3）若在第一个位的时间内取到的值不是 0 时，代表转态侦测电路因杂信而误操作，自动回到准备接收 Start bit 的状态，反之，则代表已收到一个 Start bit，此位会被移入输入移位寄存器中。

（4）4 位计数器继续下一次 16 个状态的计数，照例在第 7、8 和 9 个状态下对 RxD 取样，并以取样的多数值为串行数据的 LSB 状态，此位也移入输入移位寄存器中。

（5）重复步骤 4 共 8 次，收到 8 个数据位，此时的移位寄存器上的 9 个位都已更新，最左边的位为 Start bit，此位一定是 0。

（6）接收最后一个 Stop bit，此时 Stop bit 已被移出输入移位寄存器，并通知 Rx 控制单元将数据位载入输入 SBUF 内，而且把 TI 设成 1，RB8 也在这时被填入 Stop bit 的状态值。

假使 RI 标志位在接收前未被清除，或者是 SCON 中的 SM2 被设成 1 时，纵使接收电路有操作，但最后的数据位和停止位并没有确实载入 SBUF 和 RB8 中，所以在做接收前一定要先清除 RI 位，同时模式 1 下我们也不可以将 SM2 设成 1，否则会导致无法得到外界传来的串行数据。

13-6 串行传输实用程序范例

关于串行传输各种相关软硬件的知识与技巧在本章前几节都讲得相当多了，当所有硬件都就绪后，剩下就是程序的实战经验了，在这节中我们将分别示范两个应用程序例，来说明如何通过串行传输接口，完成各种数据的传输与控制。应用例中的程序分成 PC 端和 AT2051 端两部分，PC 端的程序是架构于 VB 语言上，本书所附的光盘中皆包含原始程序和执行文件，可供进一步研究用，AT2051 端则是以组合语言写成的程序，虽然两者的语言架构完全不同，但是在传输线上的数据与命令是相同的，所以能够相互传输也不会有问题出现，本节的范例中内定 8051 的串行传输模式为模式 1，并且以 9600 b/s 的波特率和 Windows 下的 PC 作双向通信。

范例程序一

目的：学习 PC 与 AT2051 以串行接口互传信息的各种技巧。

程序名称：S10.ASM

S10.ASM 程序，请查看书附光盘中的 CH13_S10.ASM 文件。

程序操作说明

当两台电脑要互通信息时，最重要的是通信协议（Protocol）的制订，只有两者的通信协议一致时才可相互交换数据或信息，本程序主要在示范如何在收到信息后，先做判断然后才开始做某个指定的操作，这些技巧看似简单，实际上隐藏着非常多学问，如果没看过这方面程序的人，很难想象其中的处理过程是如何安排的。

先假设 PC 端要送一段 ASCII 字符串给 AT2051 控制板，当 AT2051 依序收到这些字符串后，应当如何操作呢？仅是这点就能考倒许多人，通常串行传输通信时会在串行数据最后加上结束码（Terminator），表示该段数据已结束，对方应该尽快处理该命令或数据，若有需要时，再将处理后的结果传回，若该命令有错误时，则立即传回一个错误信息指示，当然此时的错误信息也应该加入结束码，以让接收对方知道该信息的段落。

当以 ASCII 码互传时，较常用的结束码是 ODH（Carriage Return）或 OAH（Line Feed）。只要接收到该结束码时，接收端的程序接下来要做语法分析及判断，辨别该笔命令或数据是否有效。以 AT2051 为例，要完成这些操作需要以下的子程序配合：

（1）串行初始化（SIO_INIT）子程序。

（2）串行接收（SIO_R）子程序。

（3）语法分析（SYNTAX）子程序。

（4）命令回应（MESSAGE）子程序。

AT2051 控制板用中断的方式接收 PC 端传来的信息，由于 PC 端仅做配合试验用，所以只将键盘上的数据送达 AT2051，然后随时显示 AT2051 送回的信息或数据，该段测试程序已并入 DEMO2051.EXE 连线程序中，请看稍后的说明，不过在真实应用上，PC 端程序的复杂度可能远高于 8051 的程序，为了方便起见，PC 端的控制程序是以 VB 语言写成的，对这方面不了解的读者请参考相关书籍，以便对程序有进一步的认识。

通信协议：波特率 9600 b/s，1 Start bit，8 Data bits，1 Stop bit，不做奇偶校验（No Parity）

参考程序：中断各相关程序

当我们在 PC 端开始测试后，若随便敲入几个键再按 SEND 时，AT2051 会回应"INPUTERROR!"等字样，若：

```
输入"HI"则回应"HELLO!!"
输入"HOW ARE YOU"则回应"I AM FINE"
输入"END"则回应"DEMO TEST END"等等信息。
```

以上回应的信息都是程序经过语言判断后的结果，除此之外，主程序部分仍一直让 LED 闪烁，直到我们输入"END"字符串为止。程序看起来似乎很复杂，但若以一个 Label 为一个单位逐一判读，应该不难理解其中的操作，以下就是各个标志位的大致说明：

START：系统起始设置，主要完成中断及串行接口的起始设置。

MAIN：主程序部分，控制 8 个 LED 灯不停地闪烁。

STOP：程序结束回监督程序。

SYNTAX：分析串行接口收到的字符串，不论结果如何，最后都把缓冲区内的值清除掉以免下次分析时出错。

CMDX：各个命令值的内容，由 PC 下的命令必须完全相符才可以回应正确的信息，程序中并未做大小写的判断，只有大写值才能接受，您知道如何修改吗?

MSGX：AT2051 回应的信息，每个信息后所加的 CR（0DH）和 LF（0AH）可让 PC 上的屏幕上滚一行，防止数据被后到的串行数据覆盖掉，最后的'$'符号是字符串结束的识别码，可依各别的需要改成其他符号。

CHK CMD 1：检查接收的字符串是否为"HI"，若是则回应 MSl 信息。

CHK_CMD 2：检查接收的字符串是否为"HOW ARE YOU"，若是则回应 MS2 信息。

CHK_CMD 3：检查接收的字符串是否为"END"，若则回应 MS3 信息。

MESSAGE：送出回应的字符串值，请留意传送每个字节间都加入时间延迟，以便 PC 有足够的时间接收并显示。

GETCODE：通过 DPTR 和 R5 值取到一个值供命令分析程序比较，由于本程序已烧录成 ROM，所以用 MOVC 指令取得内定的字符串值。

SIOISR：串行中断服务程序，程序执行前先禁止各种中断，然后把常用的寄存器值寄存在堆栈中，数据由 SBUF 读入后先放入缓冲区，待收到 ODH（结束码）时，才展开语法分析，请留意其中的 SUF_CNT 位置，每收到个字节值后就加 1。

CLR BUF：清除部分 RAM 区，供串行数据储存用，另外亦清除 SUF_CNT 值（串行数据计数值）。

INIT_SIO：串行接口起始。

DELAY：时间延迟用，对串行接口是有需要的。

13-7 串行传输的应用与影响

若说串行传输可以赋予单片机新的生命及应用是一点也不为过的，如果仔细地观察各种仪器、设备和电脑系统，几乎都配备有标准的串行传输接口，所以电脑间可相互通信，电脑与仪器、仪器与电脑也可以相互交换信息和数据，今天若我们开发了一项高价位的产品（电脑应用机械、化工甚至环保的监测系统），若不加入可供连线的接口，是很难在市场上被认同的。

图 13-11 HP34970 数据收集器背后有串行通信端口

AT2051 上使用的串行传输速度严格地说，不算很快（但可以很慢地传送数据），若以经济效益来衡量也是划不来的，以 AT2051 与 PC 的连接程序为例，虽然号称 9600 b/s 的传输速度，但是扣除一前一后的 Start bit 与 Stop bit 后有用的传输值只占七成多一点点，虽然有这些小毛病存在，但它仍是现在使用最广泛而且价格最公道的接口。学习 8051 单片机若没做过串行传输的各项实验，那就可真的辜负了 Intel 开发 8051 单片机工程师们的良苦用心了。

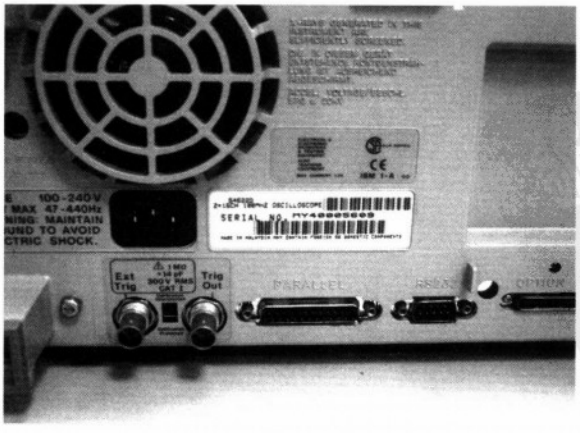


图 13-12 AGILENT 54622D 示波器的串行通信接口

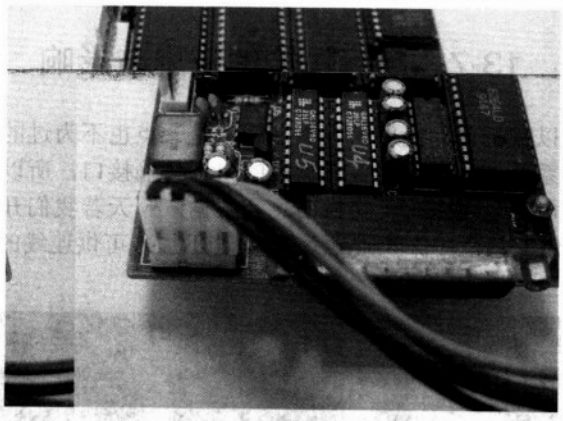


图 13-13　FLAG51 上的串行通信端口

习　题

1．试说明多处理器的通信步骤。

2．RS422A 的传输线数是多少？其传输原理是什么？

3．RS485 的传输线数是多少？其传输原理是什么？

4．列举使用 RS232C、RS422A、RS485 接口的 IC 各 3 种。

5．当 8051 的 Timer1 设置为计时模式 2，TH1=FDH，石英晶体的振荡频率为 12MHz，试问其溢位率为多少？

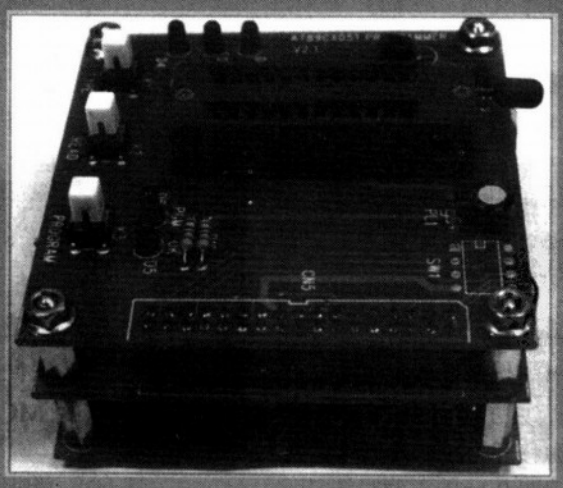
6．承上题，当 SMOD=1 时，传输通信的波特率应设置为什么？

7．当 ASCII 码进行传送时，较常用的结束码为什么？

8．简单说明串行传输的应用及影响。

进阶实习篇

14



旗威科技开发的 USB 烧录器可以让烧录的时间减至最少，怎么做？请到旗威科技公司的网站获取相关资料

第 14 章　AT2051 进阶练习（一）

本章所举的例子都属初级的应用范例，虽然这些是比较简单的程序，但是每个程序都是结构完整的程序，开机后一定会延迟一小段时间，然后进行堆栈值的设置、默认值的设置、系统中断的定义等等，完成开机所有的起始设置后，才会进入真正的控制程序，这些步骤都是不能省略的。

本书的操作练习单元都将采取以下的编排方式，以加深读者的学习广度。

范例目的：阐明学习与操作的目的。

准备工具：做本练习时所必须的工具或器材，如果有需要的话，我们会再列出辅助工具，有这些工具一定会使学习的速度加快。

操作说明：当程序执行时，AT2051 控制板会有什么反应及操作，都会在这里做描述。

程序范例：汇编语言的列表及说明。

程序说明：针对特定的函数或子程序做进一步的说明。

检查步骤：电源打开前后，你必须对软硬件做的检查。

讨论：这个练习给你的启示。

相关学习主题：如果你想要学习相关的程序或硬件，可以研究或探讨的方向。

14-1　练习：蜂鸣器的控制

目的

掌握蜂鸣器控制程序。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

AT2051 控制板上的蜂鸣器是内置振荡器型的，这种蜂鸣器叫的频率固定而且价格较贵，但是它能减少 CPU 的负担，只要接上 5V 的电压就会大声鸣叫，直到把电源移开为止。我们这里要讲解的是每秒让蜂鸣器鸣叫 0.1s 的正确写法。

图 14-1　蜂鸣器的实物图

程序范例：

```
;PROGRAM NAME B11-1.ASM
MS10 EQU 9217     ;APPROXIMATELY 10MS INTERRUPR
```

```
INTR_CNT   EQU   30H ;CNT STORIAGE
;
SCOPE      REG   P3.3    ;CHECK BY SCOPE
BUZZER     REG   P3.4
LED        REG   P3.7
;
           ORG   0000H
           JMP   START
           ORG   0003H        ;[DISABLE]  EXTERNAL INT0 INTERRUPT
           RETI
           ORG   000BH        ;[ENABLE]  TIMER0 OVERFLOW INTERRUPT
           JMP   T0_ISR
;
           ORG   0013H
           RETI               ;[DISABLE] EXTERNAL INT1 INTERRUPT
           ORG   001BH
           RETI               ;[DISABLE] TIMER1 OVERFLOW INTERRUPT
           ORG   0023H
           RETI               ;[DISABLE] SERIAL INTERRUPT
;
           ORG   30H
START      MOV   R0,#00H
           DJNZ  R0,$         ;DELAY A WHILE
           MOV   SP,#60H
           MOV   P1,#FFH
           MOV   P3,#FFH
           MOV   INTR_CNT,#00H
           CLR   BUZZER       ;P3.4=0,BUZZER OFF
           CLR   LED ;P3.7=0,LED OFF
;
           CALL  SET_TMR0
$LOOP      SJMP  $LOOP
;
T0_ISR
           MOV   TH0,#(65536-MS10)/256   ;RELOAD TH0
           MOV   TL0,#(65536-MS10).MOD. 256
           PUSH  A
           MOV   A,INTR_CNT
           CJNE  A,#10,$1
$1         JNC   $2
           SETB  BUZZER
           SJMP  $3
$2         CLR   BUZZER
$3         MOV   A,INTR_CNT
           INC   A
           MOV   INTR_CNT,A
           CJNE  A,#100,$T0_END
           MOV   INTR_CNT,#00H
           CPL   SCOPE
$T0_END    POP A
```

```
        RETI
;
SET_TMR0
        MOV     A,#01H                  ;TIMER MODE 1
        MOV     TMOD,A
        MOV     TH0,#(65536-MS10)/256   ;RELOAD TH0
        MOV     TL0,#(65536-MS10) .MOD. 256
        CLR     TF0                     ;CLEAR TIMER0 FLAG
        SETB    TR0                     ;TIMER0 START COUNT
        SETB    ET0                     ;ENABLE TIMER0 INTERRUPT
        SETB    EA                      ;ENABLE SYSTEM INTERRUPT
        RET
;
```

程序操作说明

AT89C2051 在 RESET 后，首先延迟一小段时间，然后设置 I/O 端口的状态（P1 与 P3 都要），接着启动 Timerl 做 10ms 的定时中断。每次定时中断时把 INTR_CNT 内的值加 1，加到 100 为止。当 INTR_CNT<10 时，就把 Buzzer 起动，反之则关闭。接上电源后可以看到 LED 灯马上被熄灭掉，接下来每一秒钟内，Buzzer 会叫 0.1s 后停止，持续这种叫法直到电源被中断为止。

检查步骤

用示波器分别观察 SCOPE（P3.3）与 Buzzer（P3.4）输出的时间，P3.3 每秒会取反一次，在 10ms 定时中断内，会把记数值加 1，当记数值小于 10 时，P3.4 都会是 1，所以 Buzzer 叫的时间就刚好是 100ms 即 0.1s。

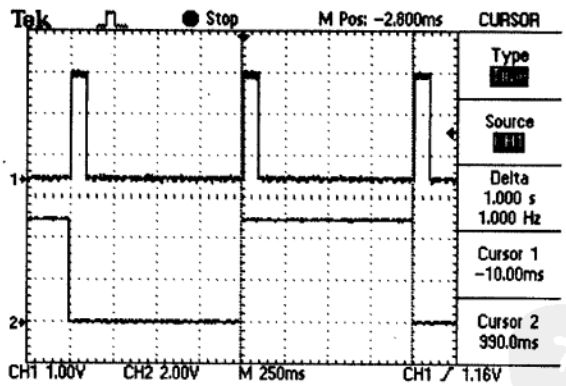


图 14-2　蜂鸣器的控制波形

讨论

一个蜂鸣器叫的程序竟然会有如此规模，很难令人相信。但是，一个严谨的小程序就是这个样子。RESET 后定义所有中断进入地址，一定先做延迟等到系统稳定后，设置所有的 IO 与内部变量值的清除。接下来才是中断的使能。

我们在先前的章节讲过，中断服务程序（Interrupt Service Routine 简称 ISR）一定要越短越好，并且要把会受影响的寄存器 PUSH 到堆栈中，当 ISR 结束时才一一 POP 取回，而且取回的

顺序不得有误。这里示范的 ISR 只有用到 ACC 寄存器，所以只有一个 PUSH A 与 POP A 的操作。

相关学习主题

LED 灯的闪烁控制也可以用类似的写法来处理。

14-2 练习：中断服务程序所占用的时间

目的

观察中断服务程序所占用的时间。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- T89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

用示波器查看 ISR 进入到离开的时间。

程序范例：

```
;PROGRAM NAME B11-2.ASM
MS10     EQU     9217          ;APPROXIMATELY 10MS INTERRUPR
INTR_CNT EQU     30H           ;CNT STORIAGE
;
SCOPE    REG     P3.3          ;CHECK BY SCOPE
BUZZER   REG     P3.4
LED      REG     P3.7
;
         ORG     0000H
         JMP     START
         ORG     0003H         ;[DISABLE]  EXTERNAL INT0 INTERRUPT
         RETI
         ORG     000BH         ;[ENABLE]  TIMER0 OVERFLOW INTERRUPT
         JMP     T0_ISR
;
         ORG     0013H
         RETI                  ;[DISABLE] EXTERNAL INT1 INTERRUPT
         ORG     001BH
         RETI                  ;[DISABLE] TIMER1 OVERFLOW INTERRUPT
         ORG     0023H
         RETI                  ;[DISABLE] SERIAL INTERRUPT
;
         ORG     30H
START    MOV     R0,#00H
         DJNZ    R0,$          ;DELAY A WHILE
         MOV     SP,#60H
         MOV     P1,#FFH
         MOV     P3,#FFH
         MOV     INTR_CNT,#00H
         CLR     BUZZER        ;P3.4=0,BUZZER OFF
         CLR     LED           ;P3.7=0,LED OFF
;
```

```
        CALL    SET_TMR0
$LOOP   SJMP    $LOOP
;
T0_ISR  SETB    SCOPE
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD. 256
        PUSH    A
        MOV     A,INTR_CNT
        CJNE    A,#10,$1
$1      JNC     $2
        SETB    BUZZER
        SJMP    $3
$2      CLR     BUZZER
$3      INC     INTR_CNT
        MOV     A,INTR_CNT
        CJNE    A,#100,$T0_END
        MOV     INTR_CNT,#00H
$T0_END POP     A
        CLR     SCOPE
        RETI
;
SET_TMR0
        MOV     A,#01H          ;TIMER MODE 1
        MOV     TMOD,A
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD. 256
        CLR     TF0             ;CLEAR TIMER0 FLAG
        SETB    TR0             ;TIMER0 START COUNT
        SETB    ET0             ;ENABLE TIMER0 INTERRUPT
        SETB    EA              ;ENABLE SYSTEM INTERRUPT
        RET
;
```

程序操作说明

这个程序跟上个程序类似，但是它在进入 ISR 时把 SCOPE 引脚设成 1，在 RETI 之前又把 SCOPE 设成 0，我们就可以在示波器上直接看到整个 ISR 的处理时间，观察中断执行的总时间如下图所示。

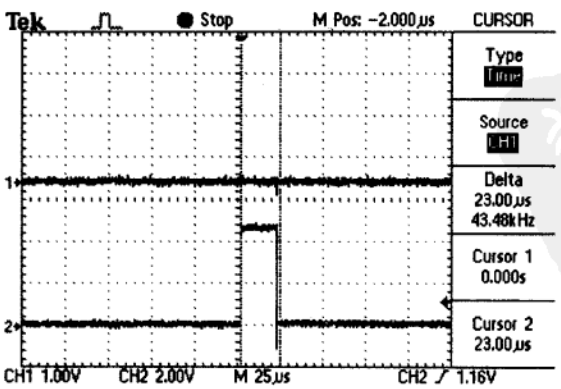


图 14-3　观察中断执行的总时间

检查步骤

示波器的 CH1 接 Buzzer，CH2 接 SCOPE（P3.3），触发源定为 CH2，我们应该可以看到 SCOPE 出现的间隔始终是 10ms，而我们关心的 ISR 执行所耗掉的时间不到 25μs，25μs/10ms=1/400，这意味着 ISR 时间只占用 1/400 的时间，系统还有 399/400 够多的时间去处理主程序或其他中断等等。请特别记住一点：当 ISR 所占用的时间过多时，系统的整体执行性能会变低，外在的表现就是整个反应速度突然变慢许多。

讨论

只要我们用到系统的任何中断，我们一定要把握一个原则：ISR 越短越好，而且 ISR 内部不得有跑不出来的循环，否则系统死机的可能性会增高，同时也不容易除错。越早懂得这个原则你就不会被汇编语言程序耍得团团转。

相关学习主题

七段显示器的定时扫描、串行中断的处理。

14-3 练习：七段显示器的初步使用

目的

七段显示器的初步使用。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

在显示幕上显示出“12”的字样。

程序范例：

```
;PROGRAM  NAME B11-3.ASM
MS10      EQU   9217        ;APPROXIMATELY 10MS INTERRUPR
INTR_CNT  EQU   30H         ;CNT STORIAGE
BUFFER    EQU   31H         ;DISPLAY BUFFER
;
SCOPE     REG   P3.3        ;CHECK BY SCOPE
BUZZER    REG   P3.4
LED       REG   P3.7
;
          ORG   0000H
          JMP   START
          ORG   0003H       ;[DISABLE]  EXTERNAL INT0 INTERRUPT
          RETI
          ORG   000BH       ;[ENABLE]  TIMER0 OVERFLOW INTERRUPT
          JMP   T0_ISR
;
          ORG   0013H
          RETI              ;[DISABLE] EXTERNAL INT1 INTERRUPT
```

```
        ORG     001BH
        RETI                ;[DISABLE] TIMER1 OVERFLOW INTERRUPT
        ORG     0023H
        RETI                ;[DISABLE] SERIAL INTERRUPT
;
        ORG     30H
START   MOV     R0,#00H
        DJNZ    R0,$        ;DELAY A WHILE
        MOV     SP,#60H
        MOV     P1,#FFH
        MOV     P3,#FFH
        MOV     INTR_CNT,#00H
        CLR     BUZZER      ;P3.4=0,BUZZER OFF
        CLR     LED         ;P3.7=0,LED OFF
        MOV     BUFFER,#12H ;DISPLAY DATA
;
        CALL    SET_TMR0
$LOOP   SJMP    $LOOP
;
T0_ISR  SETB    SCOPE
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD. 256
        PUSH    A
;
        MOV     A,BUFFER
        ANL     A,#0FH      ;GET LOW NIBBLE
        SETB    ACC.5       ;BIT5=1
        MOV     P1,A        ;SHOW 1 DIGIT
        MOV     R0,#30H
        DJNZ    R0,$        ;DELAY
;inter-blanking
        CLR     ACC.4
        CLR     ACC.5
        MOV     P1,A        ;ALL DIGIT BLANK
        MOV     R0,#04H
        DJNZ    R0,$        ;DELAY
;
        MOV     A,BUFFER
        ANL     A,#F0H      ;GET HIGH NIBBLE
        SWAP    A           ;SWAP A
        SETB    ACC.4       ;BIT4=1
        MOV     P1,A        ;SHOW ANOTHER DIGIT
        MOV     R0,#30H
        DJNZ    R0,$        ;DELAY
;
        CLR     ACC.4
        CLR     ACC.5
```

```
        MOV     P1,A            ;ALL DIGIT BLANK
;
;INTR_CNT+1
        INC     INTR_CNT
        MOV     A,INTR_CNT
        CJNE    A,#100,$T0_END
        MOV     INTR_CNT,#00H
$T0_END POP     A
        CLR     SCOPE
        RETI
;
SET_TMR0
        MOV     A,#01H          ;TIMER MODE 1
        MOV     TMOD,A
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD. 256
        CLR     TF0             ;CLEAR TIMER0 FLAG
        SETB    TR0             ;TIMER0 START COUNT
        SETB    ET0             ;ENABLE TIMER0 INTERRUPT
        SETB    EA              ;ENABLE SYSTEM INTERRUPT
        RET
;
```

程序操作说明

要显示的字样已经设为 12 且存在 BUFFER 位置上。本程序也是利用定时中断的机会把七段显示值从 P1 端口送出。由于硬件安排的缘故，12 两个数字是分批送出的，第一次送出“2”并指定 DIGIT2 即个位数，并维持该数字亮的时间约 110μs 左右，第二次送出“1”并指定 DIGIT1 即十位数，也维持该数字亮的时间约 110μs 左右，最后同时把两个位数都熄灭。平均起来每位数字每秒亮的时间约是 11ms 上下。在 AT2051 控制板上的七段显示器是属高亮度型的，所以只要 11ms 短短的时间就可以得到足够的亮度。

检查步骤

用示波器查看 DIGIT1 与 DIGIT2 的输出，即可测出每个位数被点亮的时间。

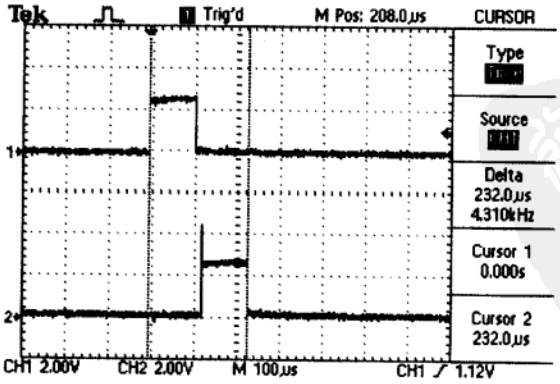


图 14-4　DIGIT1 与 DIGIT2 有 10μs 左右的间隔

讨论

在七段显示器的点亮程序中，我们在两位数字点亮的过程间，特地加上一小段熄灭（Inter-Blanking）的延迟时间，这会使数字显示上不会有残影出现。所谓的残影就是显示 1 的数字上还能依稀看到 2 的字样，而显示 2 的数字上还是有 1 的残影在。加上这小段全部都不显示的延迟就可以有效地消除显示的残影。

相关学习主题

定时中断的写法、BCD 转换程序。

14-4 练习：ACC 值的转换与显示

目的

ACC 值的转换与显示。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

将一连串十进制数据显示在七段显示器上（0～99）。

程序范例：

```
;PROGRAM NAME B11-4.ASM
MS10      EQU   9217        ;APPROXIMATELY 10MS INTERRUPR
INTR_CNT  EQU   30H         ;CNT STORIAGE
BUFFER    EQU   31H         ;DISPLAY BUFFER
;
SCOPE     REG   P3.3        ;CHECK BY SCOPE
BUZZER    REG   P3.4
LED       REG   P3.7
;
          ORG   0000H
          JMP   START
          ORG   0003H       ;[DISABLE]  EXTERNAL INT0 INTERRUPT
          RETI
          ORG   000BH       ;[ENABLE]  TIMER0 OVERFLOW INTERRUPT
          JMP   T0_ISR
;
          ORG   0013H
          RETI              ;[DISABLE] EXTERNAL INT1 INTERRUPT
          ORG   001BH
          RETI              ;[DISABLE] TIMER1 OVERFLOW INTERRUPT
          ORG   0023H
          RETI              ;[DISABLE] SERIAL INTERRUPT
;
          ORG   30H
```

```
START   MOV     R0,#00H
        DJNZ    R0,$            ;DELAY A WHILE
        MOV     SP,#60H
        MOV     P1,#FFH
        MOV     P3,#FFH
        MOV     INTR_CNT,#00H
        CLR     BUZZER          ;P3.4=0,BUZZER OFF
        CLR     LED             ;P3.7=0,LED OFF
        CALL    SET_TMR0
        MOV     BUFFER,#00H     ;DISPLAY DATA
;
$LOOP   MOV     A,BUFFER
        ADD     A,#01H
        DA      A
        MOV     BUFFER,A
        ACALL   DELAY
        ACALL   DELAY
        ACALL   DELAY
        SJMP    $LOOP
;
DELAY   MOV     R7,#00H
$1      MOV     R6,#00H
        DJNZ    R6,$
        DJNZ    R7,$1
        RET
;
T0_ISR  SETB    SCOPE
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD. 256
        PUSH    A
        PUSH    PSW
;
        MOV     A,BUFFER
        ANL     A,#0FH          ;GET LOW NIBBLE
        SETB    ACC.5           ;BIT5=1
        MOV     P1,A            ;SHOW 1 DIGIT
        MOV     R0,#30H
        DJNZ    R0,$            ;DELAY
;inter-blanking
        CLR     ACC.4
        CLR     ACC.5
        MOV     P1,A            ;ALL DIGIT BLANK
        MOV     R0,#04H
        DJNZ    R0,$            ;DELAY
;
        MOV     A,BUFFER
        ANL     A,#F0H          ;GET HIGH NIBBLE
        SWAP    A               ;SWAP A
        SETB    ACC.4           ;BIT4=1
        MOV     P1,A            ;SHOW ANOTHER DIGIT
```

```
        MOV     R0,#30H
        DJNZ    R0,$          ;DELAY
;
        CLR     ACC.4
        CLR     ACC.5
        MOV     P1,A          ;ALL DIGIT BLANK
;
;INTR_CNT+1
        INC     INTR_CNT
        MOV     A,INTR_CNT
        CJNE    A,#100,$T0_END
        MOV     INTR_CNT,#00H
$T0_END POP     PSW
        POP     A
        CLR     SCOPE
        RETI
;
SET_TMR0
        MOV     A,#01H        ;TIMER MODE 1
        MOV     TMOD,A
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD. 256
        CLR     TF0           ;CLEAR TIMER0 FLAG
        SETB    TR0           ;TIMER0 START COUNT
        SETB    ET0           ;ENABLE TIMER0 INTERRUPT
        SETB    EA            ;ENABLE SYSTEM INTERRUPT
        RET
;
```

程序操作说明

这个程序的主循环里，每隔一段时间就把 BUFFER 的内容加 1，并且做 DAA 调整，这会使 BUFFER 的结果都是十进制的格式。由于定时中断会取出 BUFFER 的内容做显示，所以，我们就可以看到数字由 00 一直变化到 99。这个程序与上个程序在定时中断服务程序上有一个重要的差异：多了 PUSH PSW 与 POP PSW。这是因为主程序正在执行 DAA 指令时，不可避免地会遇到 10 ms 的定时中断，若没有将 PSW 上的进位 Carry 存起来，当 ISR 执行完时，Carry 的真或假就不得而知。你可以试着把 PUSH 与 POP PSW 这两行指令消去，重新烧录一颗 AT89C2051，看看显示上是否有错误出现。

检查步骤

我们可以用示波器观察 DIGIT1 与 DIGIT2 的输出波形，其输出的时间还是保持在 232μs 上下。

讨论

ISR 程序要越短越好，但是如果使用到特定的寄存器时，使用前须要先将该寄存器放入堆栈中。若不这样做时会干扰到原来主程序的运行，本示范程序就是一个典型的例子。

相关学习主题

8051 中断的机制研究、对重复中断的处理。

14-5 练习：BCD 值的转换与显示

目的

BCD 值的转换与显示。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

将一个二进制码数据显示在七段显示器上（0～99）。

程序范例：

```
;PROGRAM NAME B11-5.ASM
MS10      EQU    9217          ;APPROXIMATELY 10MS INTERRUPR
INTR_CNT  EQU    30H           ;CNT STORIAGE
BUFFER    EQU    31H           ;DISPLAY BUFFER
BINARY    EQU    32H           ;BINARY DATA STORIAGE
;
SCOPE     REG    P3.3          ;CHECK BY SCOPE
BUZZER    REG    P3.4
LED       REG    P3.7
;
          ORG    0000H
          JMP    START
          ORG    0003H         ;[DISABLE]  EXTERNAL INT0 INTERRUPT
          RETI
          ORG    000BH         ;[ENABLE]  TIMER0 OVERFLOW INTERRUPT
          JMP    T0_ISR
;
          ORG    0013H
          RETI                 ;[DISABLE] EXTERNAL INT1 INTERRUPT
          ORG    001BH
          RETI                 ;[DISABLE] TIMER1 OVERFLOW INTERRUPT
          ORG    0023H
          RETI                 ;[DISABLE] SERIAL INTERRUPT
;
          ORG    30H
START     MOV    R0,#00H
          DJNZ   R0,$          ;DELAY A WHILE
          MOV    SP,#60H
          MOV    P1,#FFH
          MOV    P3,#FFH
          MOV    INTR_CNT,#00H
          CLR    BUZZER        ;P3.4=0,BUZZER OFF
          CLR    LED           ;P3.7=0,LED OFF
          CALL   SET_TMR0
          MOV    BINARY,#00H   ;BINARY DATA FOR DISPLAY
```

```
;
$LOOP   MOV     A,BINARY
        ADD     A,#01H
        MOV     BINARY,A
        CJNE    A,#100,$S100    ;IF A>=100 THEN A=0
        MOV     BINARY,#00H     ;CLEAR BINARY DATA
;
$S100   ACALL   CONV            ;CONVERT BINARY TO BCD
;
        ACALL   DELAY
        ACALL   DELAY
        ACALL   DELAY
        SJMP    $LOOP
;
CONV    MOV     A,BINARY
        MOV     B,#10
        DIV     AB              ;A=A/10,B=A MOD 10
        ANL     A,#0FH          ;A=A AND 0FH
        JNZ     $1
        MOV     A,#0FH          ;IF A=00H THEN A=0FH LEADING BLANK
$1      SWAP    A
        ORL     A,B             ;A AS BCD FORM
        MOV     BUFFER,A        ;SAVE AT BUFFER
        RET
;
DELAY   MOV     R7,#00H
$1      MOV     R6,#00H
        DJNZ    R6,$
        DJNZ    R7,$1
        RET
;
T0_ISR  SETB    SCOPE
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD. 256
        PUSH    A
        PUSH    PSW
;
        MOV     A,BUFFER
        ANL     A,#0FH          ;GET LOW NIBBLE
        SETB    ACC.5           ;BIT5=1
        MOV     P1,A            ;SHOW 1 DIGIT
        MOV     R0,#30H
        DJNZ    R0,$            ;DELAY
;inter-blanking
        CLR     ACC.4
        CLR     ACC.5
        MOV     P1,A            ;ALL DIGIT BLANK
        MOV     R0,#04H
        DJNZ    R0,$            ;DELAY
;
        MOV     A,BUFFER
        ANL     A,#F0H          ;GET HIGH NIBBLE
```

```
        SWAP    A           ;SWAP A
        SETB    ACC.4       ;BIT4=1
        MOV     P1,A        ;SHOW ANOTHER DIGIT
        MOV     R0,#30H
        DJNZ    R0,$        ;DELAY
;
        CLR     ACC.4
        CLR     ACC.5
        MOV     P1,A        ;ALL DIGIT BLANK
;
;INTR_CNT+1
        INC     INTR_CNT
        MOV     A,INTR_CNT
        CJNE    A,#100,$T0_END
        MOV     INTR_CNT,#00H
$T0_END POP     PSW
        POP     A
        CLR     SCOPE
        RETI
;
SET_TMR0
        MOV     A,#01H      ;TIMER MODE 1
        MOV     TMOD,A
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD.256
        CLR     TF0         ;CLEAR TIMER0 FLAG
        SETB    TR0         ;TIMER0 START COUNT
        SETB    ET0         ;ENABLE TIMER0 INTERRUPT
        SETB    EA          ;ENABLE SYSTEM INTERRUPT
        RET
;
```

程序操作说明

程序每隔一小段时间就把 BINARY 内的值加 1，加到十进制的 100 为止，加完后立即调用 CONV 子程序将二进制码换成十进制码的 BCD 值，并存入 BUFFER 显示缓冲区上，10ms 定时中断再从中取出值来显示。定时中断的内容与上个程序几乎完全相同。本程序对于 0～9 的数字显示有特别处理，当数值小于 10 时，十位数的 0 会被填成空白。对 4511 显示解码 IC 而言，送出四个 1（0FH）代表该数字不显示。二进制转 BCD 的子程序应该在主程序上执行，而定时中断纯粹是把该显示的数据送到七段显示幕上。请记住一个原则：中断服务程序的时间要越短越好，而且执行的时间要能够预期。换句话说就是尽量减少不可预期的循环。

检查步骤

检查显示数字 0 时应该是_0 而不是 00，这种显示方式刚好与我们一般的习惯吻合。

讨论

学习汇编语言时，不可避免地会学到二进制转十进制的子程序，这是因为在单片机的领域中，二进制计算才是它最擅长的，但是我们人习惯的却是十进制，所以任何运算最终的结果若是要给人看的话，一定要转成十进制的数值才行。

相关学习主题

16 位的 BINARY 转 BCD 码的写法推导、8051 的乘除法练习。

14-6 练习：按键操作的确认

目的

按键操作的确认。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

每按下按键一次，显示值加 1。

程序范例：

```
;PROGRAM NAME B11-6.ASM
MS10      EQU   9217            ;APPROXIMATELY 10MS INTERRUPR
INTR_CNT  EQU   30H             ;CNT STORIAGE
BUFFER    EQU   31H             ;DISPLAY BUFFER
BINARY    EQU   32H             ;BINARY DATA STORIAGE
;
SCOPE     REG   P3.3            ;CHECK BY SCOPE
BUZZER    REG   P3.4
LED       REG   P3.7
KEY       REG   P3.7
;
          ORG   0000H
          JMP   START
          ORG   0003H           ;[DISABLE]  EXTERNAL INT0 INTERRUPT
          RETI
          ORG   000BH           ;[ENABLE]  TIMER0 OVERFLOW INTERRUPT
          JMP   T0_ISR
;
          ORG   0013H
          RETI                  ;[DISABLE] EXTERNAL INT1 INTERRUPT
          ORG   001BH
          RETI                  ;[DISABLE] TIMER1 OVERFLOW INTERRUPT
          ORG   0023H
          RETI                  ;[DISABLE] SERIAL INTERRUPT
;
          ORG   30H
START     MOV   R0,#00H
          DJNZ  R0,$            ;DELAY A WHILE
          MOV   SP,#60H
          MOV   P1,#FFH
          MOV   P3,#FFH
          MOV   INTR_CNT,#00H
          CLR   BUZZER          ;P3.4=0,BUZZER OFF
          CLR   LED             ;P3.7=0,LED OFF
```

```
        CALL    SET_TMR0
        MOV     BINARY,#00H     ;BINARY DATA FOR DISPLAY
;
$LOOP   SETB    KEY             ;KEY=1
        JB      KEY,$S100
        MOV     R7,#08H
        DJNZ    R7,$            ;WAIT A LITTLE WHILE
        SETB    KEY
        JB      KEY,$S100
;KEY PRESS TRUE
        MOV     A,BINARY
        ADD     A,#01H
        MOV     BINARY,A
        CJNE    A,#100,$S100    ;IF A>=100 THEN A=0
        MOV     BINARY,#00H     ;CLEAR BINARY DATA
        ACALL   CONV
$WAIT   SETB    KEY
        JNB     KEY,$WAIT       ;WAIT UNTIL KEY RELEASE
;
$S100   ACALL   CONV            ;CONVERT BINARY TO BCD
;
        ACALL   DELAY
        ACALL   DELAY
        ACALL   DELAY
        SJMP    $LOOP
;
CONV    MOV     A,BINARY
        MOV     B,#10
        DIV     AB              ;A=A/10,B=A MOD 10
        ANL     A,#0FH          ;A=A AND 0FH
        JNZ     $1
        MOV     A,#0FH          ;IF A=00H THEN A=0FH LEADING BLANK
$1      SWAP    A
        ORL     A,B             ;A AS BCD FORM
        MOV     BUFFER,A        ;SAVE AT BUFFER
        RET
;
DELAY   MOV     R7,#00H
$1      MOV     R6,#00H
        DJNZ    R6,$
        DJNZ    R7,$1
        RET
;
T0_ISR  SETB    SCOPE
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD. 256
        PUSH    A
        PUSH    PSW
;
        MOV     A,BUFFER
        ANL     A,#0FH          ;GET LOW NIBBLE
```

```
        SETB    ACC.5       ;BIT5=1
        MOV     P1,A        ;SHOW 1 DIGIT
        MOV     R0,#30H
        DJNZ    R0,$        ;DELAY
;inter-blanking
        CLR     ACC.4
        CLR     ACC.5
        MOV     P1,A        ;ALL DIGIT BLANK
        MOV     R0,#04H
        DJNZ    R0,$        ;DELAY
;
        MOV     A,BUFFER
        ANL     A,#F0H      ;GET HIGH NIBBLE
        SWAP    A           ;SWAP A
        SETB    ACC.4       ;BIT4=1
        MOV     P1,A        ;SHOW ANOTHER DIGIT
        MOV     R0,#30H
        DJNZ    R0,$        ;DELAY
;
        CLR     ACC.4
        CLR     ACC.5
        MOV     P1,A        ;ALL DIGIT BLANK
;
;INTR_CNT+1
        INC     INTR_CNT
        MOV     A,INTR_CNT
        CJNE    A,#100,$T0_END
        MOV     INTR_CNT,#00H
$T0_END POP     PSW
        POP     A
        CLR     SCOPE
        RETI
;
SET_TMR0
        MOV     A,#01H      ;TIMER MODE 1
        MOV     TMOD,A
        MOV     TH0,#(65536-MS10)/256 ;RELOAD TH0
        MOV     TL0,#(65536-MS10).MOD. 256
        CLR     TF0         ;CLEAR TIMER0 FLAG
        SETB    TR0         ;TIMER0 START COUNT
        SETB    ET0         ;ENABLE TIMER0 INTERRUPT
        SETB    EA          ;ENABLE SYSTEM INTERRUPT
        RET
;
```

程序操作说明

按键的检查程序在主循环上执行，当测到 P3.7=0 时并不立即执行加 1 的操作，而是等待一下，再做第二次 P3.7 的状态确认，还是真的话就把设置值加 1。

检查步骤

用示波器观察按键的真实信号。

讨论

这个程序执行后，我们发现快速按一下键时，有时候显示值并没有加 1，你知道原因吗？是键盘检测的程序有问题还是定时中断的影响？

相关学习主题

多个按键的操作确认、矩阵式按键的扫描、多重中断的影响。

14-7 练习：学习波形 Duty Cycle 的计算与显示

目的

学习波形 Duty Cycle 的计算与显示。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

在 AT2051 控制板上有一个温度感测 IC SMTl60，它是以输出波形的 Duty Cycle 来代表其测到的温度值，这个程序就是把其百分比值显示出来。

程序范例：

B11-7.ASM 程序请查看随书光盘中的 CH14_B11-7.ASM 文件。

程序操作说明

这个程序是有一点复杂度的，它纯粹用软件的方式读取 SMTl60 送回来的数字信号，每次读取五个完整的波形，并且进行(HI×100)/(HI+LO)的运算，计算出的二进制结果在 04H 上，再配合先前的二进制转 BCD 码转换程序，就可以将 Duty Cycle 值显示出来。

检查步骤

用示波器观察 SMTl60 送出的频率与 Duty Cycle，然后与显示值比较，误差值应该小于 2%。

讨论

示波器上测量到的 Duty Cycle 值与程序计算值几乎相同，另外，我们也用示波器观察 SCOPE（P3.2）脚的时间，当它为 HI 时表示正在进行温度波形的测量与计算，这段时间约是 7.5ms 左右，有时会稍为长一些，推测是受到定时中断的影响。在程序当中有一段期间是禁止系统做任何中断的，这段期间刚好是读取温度信号的时期。若不这样做的话，中断再短也会干扰到我们的计数值，导致计算值的错误。

相关学习主题

4 字节的乘除法计算分析、温度的测量方法研究。

14-8 练习：学习温度值的换算与显示

目的

学习温度值的换算与显示。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

温度感测 IC SMTl60，它是以输出波形的 Duty Cycle 来代表其测到的温度值，当我们知道 Duty Cycle 后，再套用 212×Duty Cycle-68 这个公式就是把摄氏温度值显示出来。

程序范例：

B11-8.ASM 程序请查看随书光盘中的 CH14_B11-8.ASM 文件。

程序操作说明

这个程序是先取得 HI 与 LO 的记数值，然后进行一连串的整数的运算（212×HI）/（HI+LO）-68，计算出的温度二进制结果在 04H 上，再配合先前的二进制转 BCD 码转换程序，就可以将温度值显示出来。

检查步骤

用示波器直接观察 SMT160 送出波形的 Duty Cycle，然后用计算机推算出摄氏温度值，计算值与显示值的误差应该在 1℃以内。

讨论

示波器上测量到的 Duty Cycle 推算出的温度值与程序计算值吻合，到目前为止，我们所写的汇编语言程序经过编译后，实际的文件长度不到 500 字节，程序内有定时中断、七段显示器控制、温度信号测量、乘除法计算等等，AT89C2051 的程序空间有 2048 字节，这表示我们只用了四分之一的空间而已，还有四分之三够大的空间等待我们去利用。

相关学习主题

4 字节的乘除法计算最佳化分析、温度的测量与校正。

14-9 练习：温度值每秒读取两次的写法

目的

温度值每秒读取两次的写法。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

先前的温度读取程序是用调用 Delay 的方式，间隔开温度测量的时间，这种写法很难决定其读取周期。比较正确的方法是从定时中断程序下手，每隔半秒钟就把 SAMPLING 位设成 1，代表此时可以做温度的度量了。

程序范例：

B11-9.ASM 程序请查看随书光盘中的 CH14_B11-9.ASM 文件。

程序操作说明

这个程序的精髓在 10ms 的定时中断内再多加两道判断式，当 INTR_CNT 等于 0 或 50 时将 SAMPLING 设成 1，主程序一直在检查该位元的状态，只要一变为时就进行一次温度的测量。由于测量与公式计算的时间不到 8ms 就做完了，做完显示温度值的转换后，随即把 INTR_CNT 清除为 0，禁止主程序再度测量温度，直到下个半秒钟到达为止。

检查步骤

你可以把温度感测 IC 靠近冰水或温水，以便观察其中的变化。

讨论

显示的温度值如果一直与标准值差 1 度时，你知道要如何更改程序吗?

相关学习主题

温度公式的推导、温度的校正。

14-10 练习：另一种温度测量的写法

目的

另一种温度测量的写法。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器（必备）。

操作说明

先前的温度测量程序是对连续 5 个温度波形进行 HI 与 LOW 时间的计数，再套用公式计算出摄氏温度来，可是在试用时发现温度变化很大。我们改用另一种计数的方式，不过须先假设：SMT160 测得的温度曲线是连续的，如果温度变化不大时，前后间隔 0.1s 取得的温度值应该几乎相同。图 14-5 是我们 HI 与 LOW 记数的特别安排，比较特别的是 HI 是分开计算的。

程序范例

B11-10.ASM 程序请查看随书光盘中的 CH14_B11-10.ASM 文件

程序操作说明

这个程序与先前的程序在测量上有明显的不同。它先求出十个 HI 的计数值（用 DPTR 计数时精确度更高），再求出连续十个脉冲的计数值，最后才算出温度值。这种写法使得计数值加大，温度值的变动也大幅减少。温度的准确度明显地升高许多。

检查步骤

观察 SCOPE（P3.3）的波形，第一个脉冲是连续测量十个 HI 所花的时间，第二个脉冲则是测量十个 LOW 所花的时间，全部只花了 6.1ms，图 14-5 为用示波器确认计算的波形总数。

讨论

y=(212×hi)/(hi+low)-68 公式的探讨。

为何套用相同的公式，两个程序测得的温度误差会相差如此多?主要症结就在计数值取样的多少，以下配合图表说明。

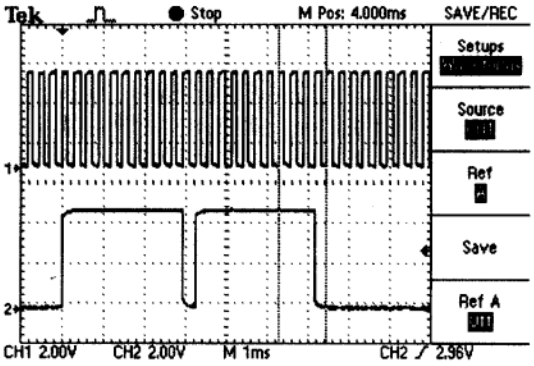


图 14-5　用示波器确认计算的波形总数

相关学习主题

温度的测量方式探讨、其他温度感测元件。

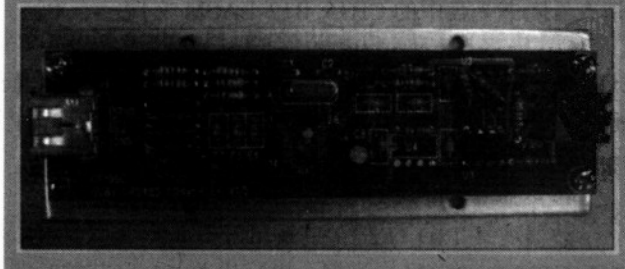
```
x=30        y=-4.4      x=31        y=-2.28
x=32        y=-0.16     x=33        y=1.96
x=34        y=4.08      x=35        y=6.2
x=36        y=8.32      x=37        y=10.44
x=38        y=12.56     x=39        y=14.68
x=40        y=16.8      x=41        y=18.92
x=42        y=21.04     x=43        y=23.16
x=44        y=25.28     x=45        y=27.4
x=46        y=29.52     x=47        y=31.64
x=48        y=33.76     x=49        y=35.88
x=50        y=38        x=51        y=40.12
x=52        y=42.24     x=53        y=44.36
x=54        y=46.48     x=55        y=48.6
x=56        y=50.72     x=57        y=52.84
x=58        y=54.96     x=59        y=57.08
x=60        y=59.2      x=61        y=61.32
x=62        y=63.44     x=63        y=65.56
x=64        y=67.68     x=65        y=69.8
x=66        y=71.92     x=67        y=74.04
x=68        y=76.16     x=69        y=78.28
x=70        y=80.4      x=71        y=82.52
x=72        y=84.64     x=73        y=86.76
x=74        y=88.88     x=75        y=91
x=76        y=93.12     x=77        y=95.24
x=78        y=97.36     x=79        y=99.48
x=80        y=101.6     x=81        y=103.72
```

习　　题

1）在练习 14-1 中，为何系统 RESET 后必须延迟一小段时间？

2）当程序中使用 ISR 时，其使用原则是什么？
3）在练习 14-3 中，要如何消除七段显示器上的残影？
4）在练习 14-5 中，如何检查显示 0 的方式正确与否？
5）在练习 14-6 中，如果按键快速连按的次数会不会等于显示的次数？原因为何？
6）何谓 Duty Cycle？
7）AT2051 控制板上的温度感测 IC 是利用何种原理测量温度值？
8）练习 14-10 的温度测量值为何较为准确？试说明其原理。

15



AT2051 控制板的 RS485 接口无法直接连到 PC 上，必须通过上面这块转换板，转成 USB 的专属信号格式之后，才进入 PC 的主机板

第 15 章　AT2051 进阶练习（二）

这章的程序示范中，我们将教你如何打开 RS485 的通信接口以及程序的正确写法。如果你以为串行通信不就是把数据往 SBUF 送，那你的程序一定会出现问题的。因为除了送数据到 SBUF 外，还要考虑到外部设备的接收硬件与速度，还有传递的内容是否要做调整等等，详情请看本章的解说与范例。

前一章中我们示范了 AT2051 控制板的许多示范程序，在这些程序当中我们也间接地学会了定时中断、七段显示器的显示、蜂鸣器与 LED 显示的正确写法，还有温度的测量方法探讨等等。本章开始我们要将重点放在串行通信上，如何把测量到的摄氏温度值送给外部的 PC。

15-1　练习：启动 RS485 串行通信接口

目的

启动 RS485 串行通信接口

准备工具

- AT2051 控制板
- DCl2V 电源供应器或电池
- AT89C2051 烧录器
- 2CH 数字式示波器

操作说明

把温度值用一个字节的二进制制的方式送出，如果现在是 32℃时，转换成十六进制为 20H，则从 SBUF 送出 0010 0000B，这个信号要用示波器观看。

程序范例

B12-1.ASM 程序请查看随书光盘中的 CH15_B12-1.ASM 文件。

程序操作说明

这个程序同时启动了 Timerl 与 Timer0，Timerl 当作串行传输的 Baud Rate 产生器，Timer0 则做 10 ms 的定时中断。串行端口则指定 9600 bps 传输速度，每次得到一个温度值后，就把温度值从串行端口送出，本程序并未启动串行中断。

检查步骤

当温度是 32℃时，示波器观察 TxD（P3.1）的输出应该如图 15-1 所示，我们应该同时确认一下脉冲的总时间是否正确？以 9600 b/s 为例，串行传送一个字节的时间是 10 位（1 Start bit+8 Data bit+l Stop bit），Start bit 是 0 很容易观察，但 Stop bit 是 1 较不易察觉，所以 10 位的时间为 1.041ms=(1/9600)×9，由于 Stop bit 较难看出，我们通常都是查看 Start bit 与 Data bit

的总时间约 937.5μs，图 15-1 测量到的时间是 950μs，两者只相差 12.5μs 而已。图 15-2 是 RS 485 差动输出端 SIO 与 $\overline{SIO}$ 的信号，很明显的两个信号刚好是反的。

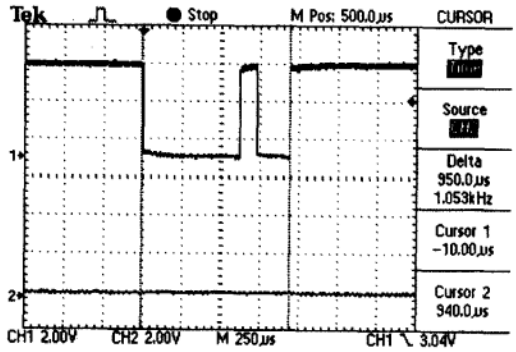


图 15-1　温度是 32℃时，示波器观察 TxD（P3.1）的输出数据

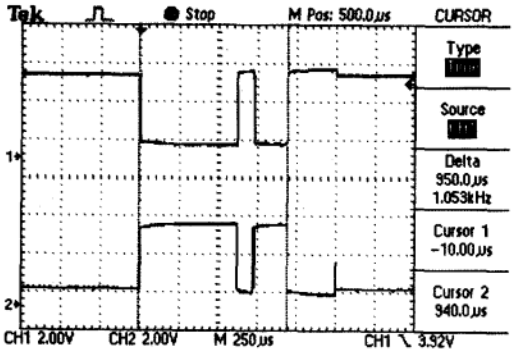


图 15-2　RS 485 差动输出端 SIO 与 $\overline{SIO}$ 的波形配合线路

讨论

这个程序只能用示波器查看的确不方便，不过我们在此可以确认一点：传输速度是 9600 b/s，传送的二进制数据也正确。RS485 的接收端如果可以接受二进制数据的话，就可以显示温度值了。在实际的应用上，我们习惯将温度转成标准 ASCII 的数据再送出。

相关学习主题

传输速度如何提高到 19200 b/s、如何把温度值转换成 ASCII 码。

15-2　练习：练习温度值转成 ASCII 字符串的写法

目的

练习温度值转成 ASCII 字符串的写法。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。

- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

若温度值为 31℃时，从 RS485 串行传输接口中送“T=31C”等 4 个字样。

程序范例

B12-2.ASM 程序请查看随书光盘中的 CH15_B12-2.ASM 文件。

程序操作说明

ASC_CONV 子程序将 BCD 码值分隔开，并加上 30H（即数字“0”）并且在字符串首尾加上“T=”与“C”，所以如果 PC 机上如果也有 RS485 接口的话，应该可以见到类似“T=31C”等的字样。

检查步骤

用示波器查看 TxD（P3.1）的波形，并计算总共传输的时间。

讨论

这个程序若要实际验证的话，必须在 PC 端加装 RS485 的转换接口，如图 15-3 所示，左边是旗威科技公司的 RS232 转 RS485 接口，右边则是 RS485 转 USB 的接口，只要写一个简单的串行接收程序就可以看到 AT2051 送回来的温度字符串值。

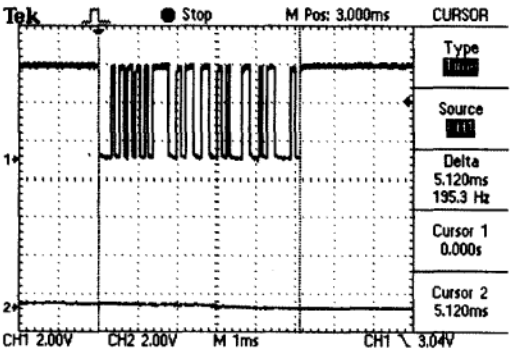


图 15-3　传送 5 个串行数据 T=31C

相关学习主题

串行中断的写法。

15-3　练习：串行传输的写法一

目的

学习定时中断法。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

每隔 10 ms 就回送一个串行数据，总共送 7 次。

程序范例

B12-3.ASM 程序请查看随书光盘中的 CH15_B12-3.ASM 文件。

程序操作说明

串行程序隐藏在定时中断里，每次定时中断发生时，除了做七段显示器的扫描外，最后再加上 TX_START 的位判断，若为 1 则从 ASCBUF 内取一个字节送到 SBUF 上，并且把 TX_CNT 加 1，程序控制最多送 7 字节，最后两个码是跳行的控制码，所以我们在 PC 端可以看到温度值持续被送回来，而且旧的数据会一直往屏幕上滚，最新的温度值始终在屏幕下方。

检查步骤

用示波器可以明显看到串行数据每隔 10 ms 才被送出，而且送出七笔后就停下来，直到下次温度值产生为止。

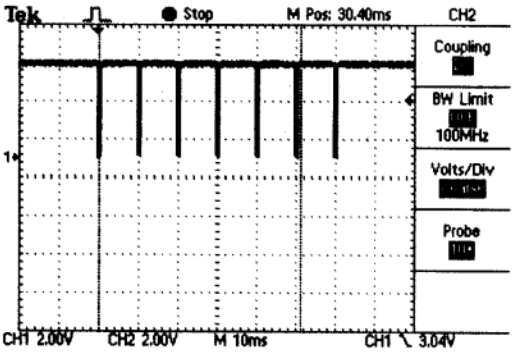


图 15-4　每 10ms 送回一个 Byte 的串行数据

讨论

这种串行传输的写法是我们惯用的写法之一，传输的间隔非常明确，唯一的缺点是传输时间较长（7 字节约要 70 ms），并不适合做大量数据的传输用。

相关学习主题

各种串行传输写法的比较。

15-4　练习：串行传输的写法二

目的

学习串行中断法。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器（必备）。

操作说明

串行中断的标准写法。

程序范例

B12-4.ASM 程序请查看随书光盘中的 CH15_B12-4.ASM 文件。

程序操作说明

当我们把要传输的数据放好之后，串行这时是不产生中断的，直到程序把 TI 被设成 1 为止，在串行中断的 ISR 中，TI=1 代表 SBUF 内部是空的可以接受新的数据，所以我们把第一个数据放入，同时把 TI 清除成 0。当 TI 又成为 1 时代表一个数据已经成功送出，可以再送出下个数据了。

检查步骤

当用示波器查看 TxD 的波形时，几乎所有的串行数据都连在一起，如图 15-5 所示。如果这时你想要看出每个字节的内容，是会有一点困难的，除非你使用的示波器有非常大的存储空间。

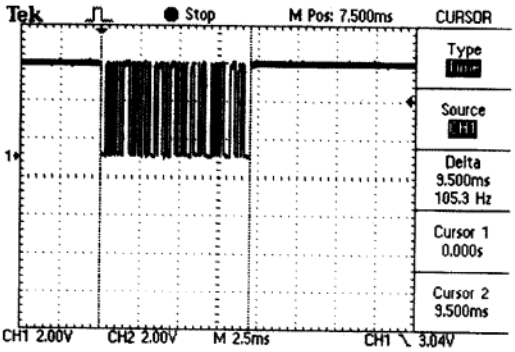


图 15-5　串行中断法

讨论

串行传输的写法跟我们常见的应用程序有些差异，当所有该传送的数据都放好后，把 TI 设成 1 让串行中断产生，所有传送的操作都偷偷地执行，连传输结束的判断也是在串行中断 ISR 上处理，看起来操作相当单一。这种写法的传输速度很快，7 字节只要 7 ms 就解决了，这种速度对 PC 而言都是算慢的，所以 PC 可以很从容地显示这些外部传来的数据。

相关学习主题

两种串行方式的比较。

15-5　练习：将温度的精确度提高到小数点后一位

目的

将温度的精确度提高到小数点后一位。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。

♦ AT89C2051 烧录器。

♦ 2CH 数字式示波器。

操作说明

七段显示器仍然显示小数点前的两个数值，但串行传送值就可以把小数点后的值顺便送出。

程序范例

B12-5.ASM 程序请查看随书光盘中的 CH15_B12-5.ASM 文件。

程序操作说明

当程序要增加一位数字的显示时，绝对不是修改几个字节而已，最重要的公式将改为 T=2127×DUTY-681，新的 T 温度值将占 2 字节，要把 16 位的数据变成 000～999 的 BCD 码也是一项考验，最后串行传输的缓冲区也要扩大到 9 字节，以方便送出“T=32.1C”外加两个 CR 与 LF 跳行特别码。

检查步骤

进行修改前，我们应该在纸上列出所有待修正的子程序以及增添的运算空间，然后才真正进行程序的修正。在实际验证上，我们可以把温度感测器靠近冰水或温水，看看温度的变化情形如何。另外观察串行送出来的温度值是否正确？

讨论

我们用最笨的方法来做 16 位的 BCD 码转换程序，该段程序不长也不难理解，可是它计算的时间最久。由于温度值约在 000～999 间变化，所以其计算的循环最多也只到 999 次而已，在尚未找到更快的写法之前，这个子程序还是有其利用的价值存在。

相关学习主题

BCD 码转换程序的探讨、串行通信中断与 10 ms 定时中断的比较。

15-6 练习：串行除错程序的加入

目的

串行除错程序的加入。

准备工具

♦ AT2051 控制板。

♦ DC12V 电源供应器或电池。

♦ AT89C2051 烧录器。

♦ 2CH 数字式示波器。

操作说明

温度感测 IC 输出波形的 HI 与 HI+LO 的计数值分别列出，用人工的方式转成十进制值，再套用公式计算出温度值来。

程序范例

B12-6.ASM 程序请查看随书光盘中的 CH15_B12-6.ASM 文件。

程序操作说明

写汇编语言程序一定要学会串行通信程序，而且要先把这部分程序写妥，当我们对某些转换后的值有疑问时，就可以直接把该值转换成 HEX 值后，经由串行接口传出。当然此时

我们选择是否将原有的串行中断暂时关闭。

检查步骤

本程序共回送 3 个 16 位值及摄氏温度值，我们在屏幕上看到回送的值分别是：

```
0320H 0690H 014BH T=33.1C
0320H=3×256+2×16=800 HI 的计数值
0690H=6×256+9×16=1680 HI+LO 的计数值
014BH=1×256+4×16+11=331 计算完尚未转成 BCD 码的值
```

T=（2127×800）/1680-681=331 刚好就是第 3 个回送值 014BH，这证实了我们的乘除计算式是正确的。如果在 331 的 1 之前加一个小数点，正是 33.1℃，在 8051 的控制领域中，整数的加减乘除才是最直接的。我们只花不到 700 字节的程序空间，完成了温度的测量与显示，同时又包括了连线通信等等，也只有汇编语言才能达成这个不可能的任务，若是用其他高级语言来处理，程序空间肯定要超过 4KB 以上。

讨论

DEBUG 这个程序将 16 位值直接显示出来，它的用处是相当广的，我们可以拿它来监视变量的值，经过复杂计算后的值或是重要的计数值。当我们完成整个程序的除错后，仍旧可以保留 DEBUG 程序，只是在 DEBUG 进入点直接改成 RET 即可。以后有机会维护程序时再把 RET 移开即可重新除错。

相关学习主题

DEBUG 程序可否显示程序计数值（Program Counter）如何与 PC 做串行通信、标准 RS485 通信的写法。

习　　题

1）如何以示波器观察由控制板所送出的 ASCII 码？

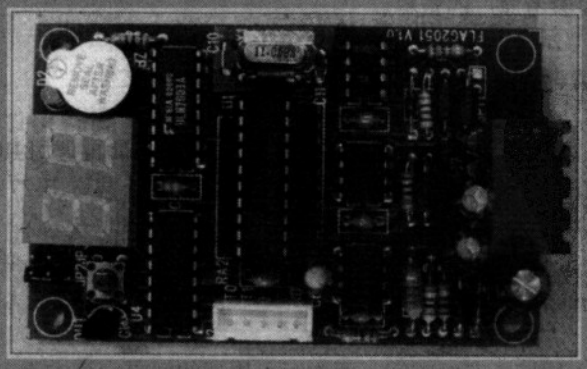
2）承上题，利用查表，描绘出当示波器接收到“T”时的波形。

3）串行传输的操作如何进行？请简要描述。

4）当 PC 接收到换行的信号时，示波器上所观察到的波形是什么？

5）如果要把测量的精度从整数提高到小数点下一位，就程序而言应该做何修正？请简述其修改的方法。

16



在工业控制的应用上，我们用双绞线连接远在数百米外的 AT2051 控制板，依然能顺利取得温度值

第 16 章　AT2051 进阶练习（三）

标准的 8051 内部是没有 E^2PROM 的，所以当有特定的参数要存储时，一定要外加 E^2PROM，若使用串行的 E^2PROM 可以使硬件的接线减至最少。本章的应用大都围绕在 E^2PROM 24LCl6 的读写方法分析，最后还可以通过 RS485 的远端 PC 来修改 E^2PROM 的内容，所以程序会较大，但程序本身都已模块化，解读起来不会很吃力。

前两章里我们示范了 AT2051 控制板的许多示范程序，这章开始我们要将要探讨如何把数据放到串行 E^2PROM（可擦除只读存储器）当中，当然也可以把测量到的温度值暂时存在 E^2PROM 里，当 PC 需要时才传出。AT2051 控制板上使用的串行 E^2PROM 编号是 24LC16，其容量为 2K×8 位。我们还可以把有关温度的参数值存在 24LC16 内，进行进一步的温度校正与调整。

16-1　练习：写入一个字节的数据到 E^2PROM 24LC16 内

目的

写入一个字节的数据到 E^2PROM 24LC16 内。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器（必备）。

操作说明

写入的时序一定要参考原厂的资料。

程序范例

B13-1.ASM 程序请查看随书光盘中的 CH16_B13-1.ASM 文件。

程序操作说明

在数据写入前，必须先定义好 R0、Rl 与寄存器 B 的值，然后再调用 IIC_WRITE 子程序，就可以将数据存储到串行 E^2PROM 24LC16 中，SCLK（P1.7）共送出 27 个脉冲，分别送出控制码（Control Byte）、存储地址（Address）与存储数据（Data Byte），每个字节送出之后 24LC16 会有一个回应（Acknowledge）信号，所以要多送出三个 SCLK 给 24LC16。如果一切顺利的话，过了 10 ms 后该数据就会转存到 E^2PROM 内。

检查步骤

观察 E^2PROM 写入周期所花的时间，两次写入周期的间隔应该大于 10 ms，否则可能有数据丢失的机会。

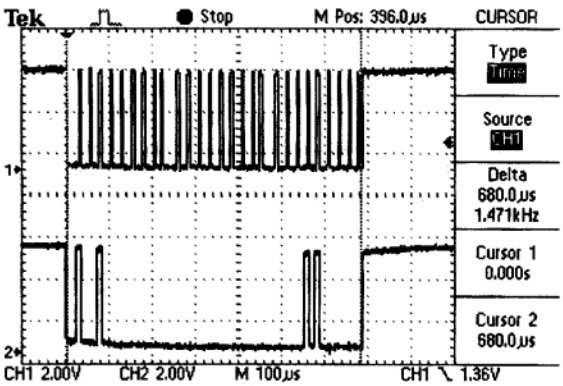


图 16-1　680 µs 的时间内返回了 27 位

讨论

串行 E^2PROM 241LC16 总共有 2KB 的存储空间，存放一般的设定值是绰绰有余的。可是每次存储时要等待 10 ms 即百分之一秒，对 8051 而言 10 ms 就可以做许多事情了，这也代表串行的 E^2PROM 不适合做经常的高速读写，时常返回但偶尔的写入就很恰当。许多仪器都是利用类似的串行 E^2PROM 来存放系统的参数或设定值，而不是用 SRAM 加电池备份，就是取串行 E^2PROM 的优点。

相关学习主题

相关 E^2PROM 的数据搜索、E^2PROM 的返回写入波形分析。

16-2　练习：E^2PROM 的读回写法分析

目的

E^2PROM 的读回写法分析。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器（必备）。

操作说明

将 00～99 的 BCD 码依序存入 E^2PROM 的地址 0 当中，并且逐次读回及显示出来。

程序范例

B13-2.ASM 程序请查看随书光盘中的 CH16_B13-2.ASM 文件。

程序操作说明

这个程序例中有 E^2PROM 写入与返回两项功能，为了使示波器观察方便起见，我们把 SCOPE 触发点安插在读回（IIC_READ）的子程序上。

检查步骤

查看读回的串行数据与脉冲间的对应情况。

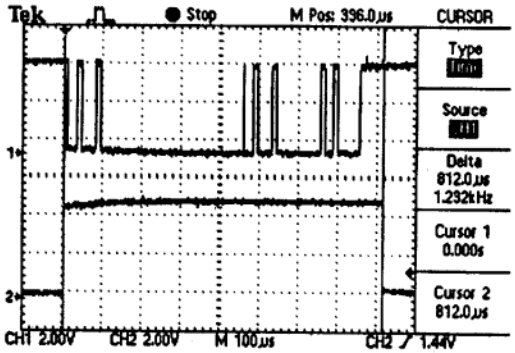


图 16-2　CHl SDATA CH2 SCOPE 串行 E^2PROM 读回时间约是 0.8 ms

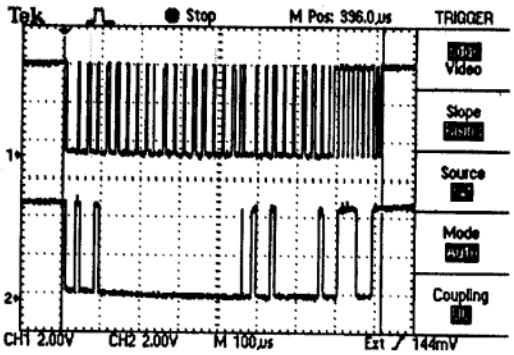


图 16-3　运用示波器的外部触发，可清楚看到 IIC_READ 的读回波形，若 SDATA 最后有数据出现，代表 24LC16 确实有所回应

讨论

串行 E^2PROM 数据的写入与读回程序往往是软件工程师的最不喜爱碰触的，原因是验证困难。通常我们要先完整的看三次数据手册，把重点挑出来然后才开始写汇编语言程序，并且一定要配合示波器查看 SCLK 与 SDATA 的波形是否符合原厂的规范。最困难的是要在适当的地方加入适当的软件 Delay。分别完成写入与返回的程序后，再把程序做进一步的精简。别人写的程序看似很容易，但是实际上亲自做了才知道精髓就在这里。

相关学习主题

常见的串行 E^2PROM 写法分析。

16-3　练习：ID 值读取的写法

目的

ID 值读取的写法。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。

- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

从 E^2PROM 当中返回 ID 值，并且在七段显示器上显示。

程序范例

B13-3.ASM 程序请查看随书光盘中的 CH16_B13-3.ASM 文件。

程序操作说明

开机就序后，程序会去读取 E^2PROM（地址 0）内的值，若介于 1～9 之间就不做调整，反之把 ID 值设成 1，接着显示 ID 值一秒钟后才进入温度的测量程序。

检查步骤

显示 ID 值时蜂鸣器会跟着叫，此时串行通信送回的数据中会多一个 ID 值，成为 T1=32C。

讨论

ID 在 RS485 通信中是很重要的一环，既然 AT2051 送出温度值的同时也可以夹带 ID 值，那我们的串行通信上要稍做修改，改为调用对应的 ID 后，才送出温度值。在 RS485 的通信中，只用两条对绞线就可一对多通信，而其中就是靠 ID 来做各个设备的分辨。

相关学习主题

温度有哪些参数可以存放在 E^2PROM 24LC16 当中。

16-4 练习：如何判断 E^2PROM 是否存在

目的

如何判断 E^2PROM 是否存在。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

判断 E^2PROM 24LC16 是否存在，可先返回某个特定位置内的值，将该值加 1 后存入相同位置。再次返回时若正确则判定 E^2PROM 存在，反之则不存在。

程序范例

B13-4.ASM 程序请查看随书光盘中的 CH16_B13-4.ASM 文件。

程序操作说明

若 24LC16 不存在时，显示器会闪烁显示 99 等字样。

检查步骤

若在 AT2051 控制板上装有 24LC16 时，开机后先显示 ID 值后直接进入温度测量程序。反之 24LC16 不存在时，AT2051 控制板上会闪烁 99 字样，同时也会伴随蜂鸣器的叫声，直到电源关闭为止。

讨论

当系统察觉 24LC16 不存在时，可以立刻停止操作并且显示出错误码来，也可以继续执

行程序，但改采预设的参数值。如果你的程序是控制精密的仪器的话，除了运用 24LC16 存放设定值外，更要有其他的保护程序或硬件配合，才能使系统绝对稳定。

相关学习主题

系统程序如何自保及检测 E^2PROM、E^2PROM 的安全性探讨。

16-5 练习：ID 值的在线更改

目的

ID 值的在线更改。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

打开电源显示 ID 值的同时若按下按键，可立即更改 ID 值。

程序范例

B13-5.ASM 程序请查看随书光盘中的 CH16_B13-5.ASM 文件。

程序操作说明

程序启动后，首先检查 24LC16 是否存在，是的话返回 ID 值并做显示，在一秒钟的显示时间内，若察觉到 KEY（P3.7）=0 键被按下时，程序会把 ID 值加 1 并再次存入 24LC16 内。

检查步骤

设定完新的 ID 值后，串行通信的内容也会跟着变动。

讨论

这个程序的串行通信部分要稍做修改，改为收到 ID 询问码时才回应，这样就能多台 AT2051 并联通信了。

相关学习主题

RS485 的优点与缺点。

16-6 练习：配合 ID 调用的串行通信程序

目的

配合 ID 调用的串行通信程序。

准备工具

- AT2051 控制板。
- DC12V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

当由串行端口收到 ID 的二进制码时，才发送温度字符串。

程序范例

B13-6.ASM 程序请查看随书光盘中的 CH16_B13-6.ASM 文件。

程序操作说明

程序允许定时与串行中断，当串行中断发生时，串行 ISR 程序先检查 RI 位，若为真则外部读入传来的串行值，并且判断是否该值就是 AT2051 控制板的 ID 码，对的话就开始把输出字串依次送出去。

检查步骤

你可以在 Windows 的 VB 环境或 DOS 模式的 Turbo C 下送出 01H～09H 的二进制码，AT2051 控制板若收到正确的 ID 值时，就可以随即把温度值送出。

讨论

B13-6 的程序就是一个典型的 RS485 通信程序，有一台主机与众多设备通过两线式的 RS485 连在一起。主机会送出 ID 值去询问哪个设备已经备妥数据可回送，被调用的设备就立即回应其测量到的数值或物理量。

相关学习主题

RS485 的硬件分析、8051 串行传输的极限速度。

16-7 练习：串行通信程序的除错

目的

串行通信程序的除错。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器（必备）。

操作说明

先前一个程序的整体操作都正确，但是在回送数据字串时，有时后会多送出几个字节来，假设其 ID 值是 5，则回送的字串为“T5=31C”。可是有时候会送成“T5=T5=31C”或“T5=3T5=31C”，这代表程序中的 TX_CNT 指针有可能被修改过才会有这种结果。

程序范例

B13-7.ASM 程序请查看随书光盘中的 CH16_B13-7.ASM 文件。

程序操作说明

把原先 TX_CNT=00H 的指令改到串行中断 ISR 中就没问题了。

检查步骤

每次 PC 送出 ID 询问时，观察 AT2051 控制板回送的串行数据。用示波器查看串行数据的波形，从 PC 发出询问 ID 到 AT2051 控制板回应 7 字节的时间约在 9～10 ms。

讨论

先前的程序写法为何会偶尔有错误才是我们想急于了解的，请看下面的附图说明。其中的重点就是当传输开始时，某次温度的测量与 ASCII 码转换刚刚好完成，原程序会再度把 TX_CNT 设成 0，使得重新传出字符串，这才使我们看“T5=T5=3lC”的字样。你觉得这种

说法可以接受吗？

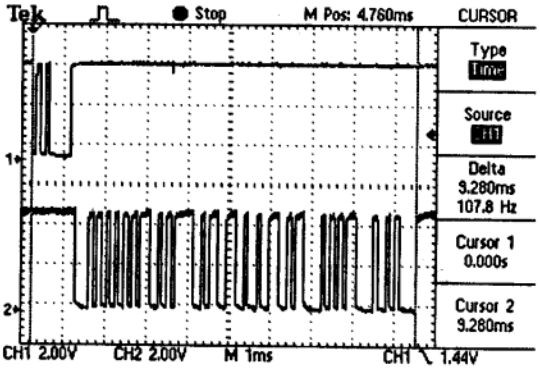


图 16-4　从 PC 端询问，到获得回应的串行波形

相关学习主题

串行传输会影响温度的测量值吗？

16-8　练习：RS485 通信程序的完整版

目的

RS485 通信程序的完整版。

准备工具

- AT2051 控制板。
- DCl2V 电源供应器或电池。
- AT89C2051 烧录器。
- 2CH 数字式示波器。

操作说明

先前的程序只送一个 ID 值就可以回应温度值。不过，许多仪器设备都尽量采用 ASCII 码来连线，所以我们接下来的示范程序就要采用相同的模式，也就是送出去的字符串变成“ID5?”，这时 ID 值是 5 的 AT205l 控制板就要送回温度值。

程序范例

B13-8.ASM 程序请查看随书光盘中的 CH16_B13-8.ASM 文件。

程序操作说明

远端命令的分析是串行通信中最重要的部分，我们要有足够的空间来存放接到的字符串，当收到 CR 与 LF 结束码时，要立即进行命令语法的分析，若是被调用的话，则要把事先准备好的 ASCII 温度字符串送出去，反之则保持沉默，不送出任何信息。

检查步骤

用示波器观察 AT2051 收到串行数据到回送温度值所花的时间，并且在 PC 的屏幕上应该可以看到正确的温度值，可是有时候送出 ID 的询问命令却没回应?有时候回送的温度值竟然是 81C（正确值应该是 31C），这是什么原因呢？

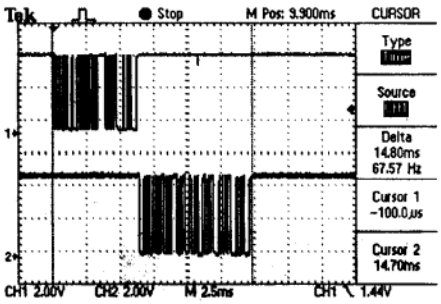


图 16-5　不到 15ms 的时间就完成一次送与收的任务

讨论

到目前为止，我们所写的汇编语言程序约是 1KB 左右，主要的程序模块有：

（1）READ_TEMP 温度测量模块，每秒一次。

（2）T0_ISR 定时中断，每 10 ms 中断一次，同时更新七段显示器的内容。

（3）SERIAL_ISR 串行通信，接收外部传来的命令，被指定到时要立刻回送温度值。

当这三个重要模块纠缠在一起时，会有什么结果呢？前几章我们所看到的程序都很简单，所以不会有问题出现，但是加上串行中断后，我们的程序若处理不妥当时，就会有毛病出现！例如，串行中断在传送数据时，READ_TEMP 刚好完成一个温度值的转换，也正要对 ASCBUF 的内容更新，这样就很可能造成传输数据的错误。

首先，我们要确认 ISR 内不会影响原有程序的寄存器内容，接着要分析出哪个模块最重要，当它在执行时，不会受到别人的影响，只要确定避开这些危险因素，就可以使我们的程序更坚固。经过仔细地检查后，我们发现在串行中断 ISR 中并未对 01H 与寄存器 B 做 PUSH/POP 指令，少了这两个操作可能会使系统变得不稳定，经过修正后，传输的数据就完全没有错误了。

B13-9.ASM 只列出 SIO_ISP 部分的程序内容，其程序请查看随书光盘中的 CH16_B13-9.ASM 文件。

相关学习主题

当各种中断接连发生时，你的程序该如何应对。

B13-10.ASM 分别把串行中断与定时中断所占用的时间显示出来，我们以 CH1 代表 SIO_ISR，CH2 代表 T0_ISR，你可以看到其中复杂的变化情形。其程序请查看随书光盘中的 CH16_B13-10.ASM 文件。

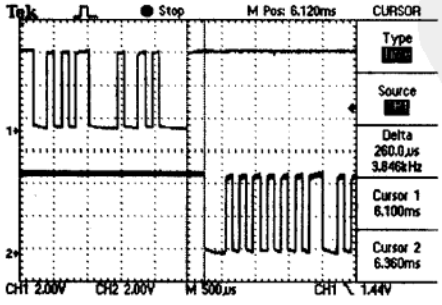


图 16-6　AT2051 收到最后一个字节（OAH）后，过了 260 µs 的指令分析后，开始将温度值返回

图 16-7 我们预期串行中断执行的时间很短，而定时中断是每 10 ms 一次，可是由图看来，似乎串行中断一直没有停过，这一定是 RI 或 TI 标志位没有被清除的结果，这样看来，我们又找到一个隐藏性的臭虫。如果这个 bug 不去除的话，系统会有一半以上的时间被串行中断占用，整体的执行速度一定会变慢的。

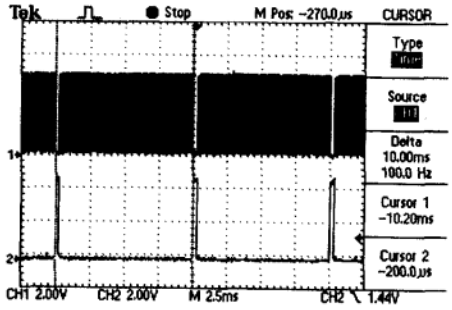


图 16-7　似乎不太正确的串行 ISR

B13-11.ASM 经过再次修正的串行中断 ISR 程序，其程序请查看随书光盘中的 CH16_B13-11.ASM 文件。

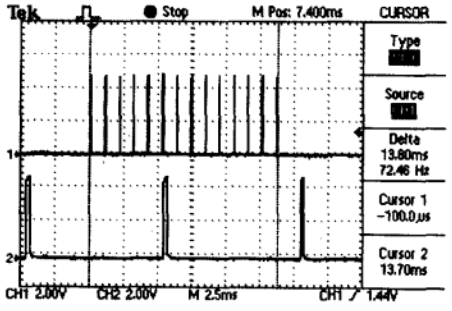


图 16-8　CH1 为串行 ISR，CH2 为 TO_ISR，总共产生 14 次串行中断，前 6 次是 RI 中断，后 8 次是 TI 的中断，串行中断间还是有定时中断产生

B13-12.ASM 是我们在串行中断时避开定时中断的写法，其程序请查看随书光盘中的 CH16_B13-12.ASM 文件。

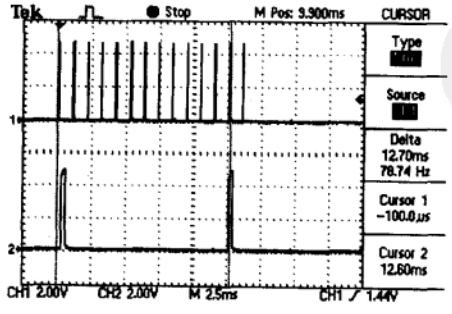


图 16-9　由 CH2 可以看出这两次定时中断的时间已经不是 10 ms 了（约是 13 ms），这表示有一段时间被系统禁止做 Timer0 的定时中断

习　题

1）如何写入 1 字节的数据到 E^2PROM 24LCl6 中？简述其写入的操作。

2）承上题，E^2PROM 24LCl6 的存储空间大小为多少？

3）验证 E^2PROM 的数据写入与返回，需借助示波器查看何类波形是否符合原厂的规范？

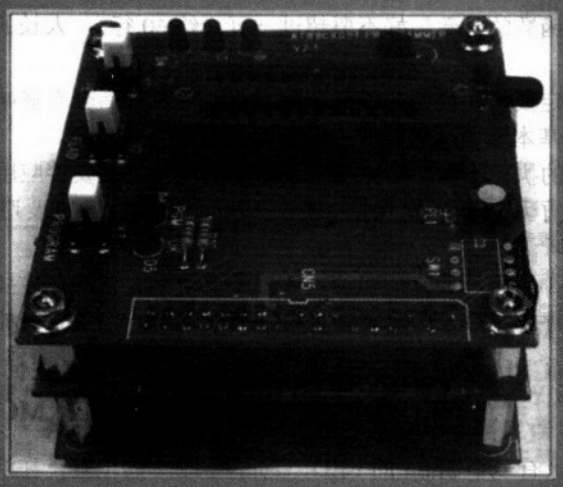
4）利用串行传输的方式进行多项不同数据的传输时，最常遇到的问题是什么？又 Delay 在串行传输中扮演哪种角色？

5）串行传输的速度是否与 CPU 的运行速度有关？

6）承上题，串行传输的中断命令会否影响 CPU 的运算性能？

7）在串行传输返回数据的过程中，如果发生数据字符串多送或不足的情况时，其发生最可能的原因是什么？

17



USB 烧录器上有 32KB 的 SRAM，烧录 AT89C4051 只用到八分之一的数据存储器空间

第 17 章　汇编语言的写法分析与除错

许多 8051 的书籍教你写汇编语言程序，但是本章教你如何除错。程序大家都会写，但是写一个好的程序，不但让别人看懂，还要让别人能帮你除错，这就难了许多。在本章里我们交待了许多简单好用的除错技巧，只要系统有保留一些简单的输出电路，我们就可以进行软硬件的除错。

17-1　汇编语言的难点

汇编语言是所有控制程序的基础程序，它直接控制 CPU 内部的寄存器，稍有不慎就会使程序进入跑不出来的死循环中，这时只有关机重来一种选择了。也就是这个原因才使得许多初学者以为一定要用仿真器（ICE，In Circuit Emulator）才能学习汇编语言，其实没有 ICE 照样能写汇编语言，虽然刚开始会比较辛苦，但是只要学会本章的几种除错方法，依然可以“快快乐乐”地写汇编语言程序，不过刚开始的艰苦绝对是避免不了的。

17-2　写程序的重点

当你要写任何汇编语言程序时，我们列出以下几个重点中的重点，请在写程序的同时务必要纳入考虑：

重点一：子程序或函数的行数最好不要超过一页（约 60 行），太长的话日后阅读或维护较困难。

重点二：子程序一定要加注说明，说明调用方式以及执行后会有影响的寄存器。

重点三：中断 ISR 基本上要越短越好，绝对避免进入无穷循环。

重点四：中断 ISR 的执行不能影响原来寄存器的值，所以要妥善应用 PUSH/POP 指令。

重点五：主程序及子程序都一定保持一个入口一个出口，一定要避免远程的跳转指令（LJMP）。

重点六：做完一项小修改后要立即验证，千万不要等到大修改后才要做功能检测。

重点七：对所有程序与指令产生合理的怀疑，有疑问时一定要查阅相关工具书籍。

17-3　LED 除错法

在 AT2051 控制板上有一个 3mm 的 LED，它占用 P3.7 的位置，在汇编语言程序的声明加上 LED REG P3.7 后，我们就可以使用 SETB LED 与 CLR LED 这两个位设置与清除指令了。由于线路上 P3.7 同时又是 KEY 按键的输入点，若要检查按键的状态时，一定要加一个 SETB LED 的指令，来检查按键的状态。在 AT2051 控制板重启之后，因为硬件的缘故 LED

一开始是亮的，我们习惯先把 LED 熄灭，然后在特定的时刻再把 LED 点亮，至于灭掉的时机就看程序上的设置了。当我们怀疑某段子程序并未执行时，可以在这个子程序的进入点多加一行 SETB LED 指令，若预期这个子程序执行的时间超过 1s 时，可以在该子程序结束后顺便把 LED 关闭。

当程序验证完毕后，请不必急于将 SETB LED 这行指令删除，可以在该行指令最前端加上分号即可，等到所有程序都确认无误后再删除。

17-4 蜂鸣器除错法

AT2051 控制板的 P3.4 经过 JP3 跳线接到蜂鸣器上，若不把 JP3 短路的话，是无法让蜂鸣器叫的。我们要在程序最前头声明 BUZZER REG P3.4，才可使用 SETB BUZZER（蜂鸣器叫）与 CLR BUZZER（蜂鸣器停止鸣叫）的指令。假设我们怀疑定时中断没有执行，便可以在定时中断的 ISR 加上鸣叫的指令 SETB BUZZER，离开 ISR 时才把蜂鸣器关闭。如果蜂鸣器没有叫的话，表示程序的定时中断启用有问题，应该详细研究这一部分的设置程序。

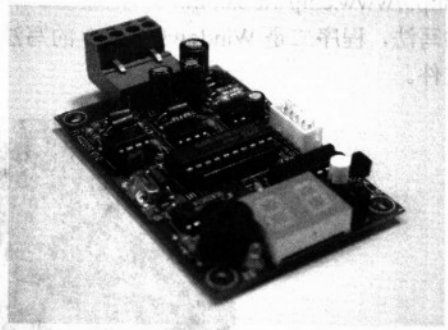


图 17-1 只要电路板上有一颗 LED 或蜂鸣器就可以当成除错的工具

17-5 DISPLAY 除错法

前面讲了两种程序除错的方法，可是有时候某些除错场合不是只看看指示灯或听听声响就可以解决的，我们想知道的是某个值是多少，不只是真或假而已。这个时候就可以借助 AT2051 控制板上的两个数字，把数值很正确地显示出来。

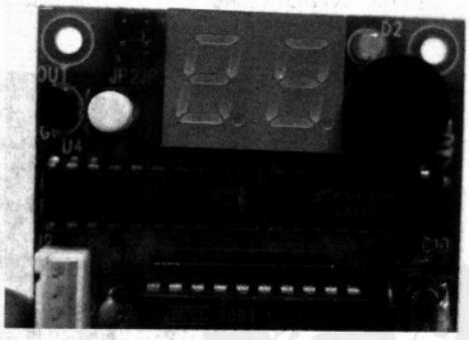


图 17-2 无论是哪一种显示器，在开发初期都可以用来做除错工具

不过，一定有人会问：我的值可以从 0 变化到 255，两个数字怎么显示？分两次显示吧！其实在除错阶段辛苦一点又何妨？数值 208 的显示可分成两次来进行，第一次显示 2，十进制填为空白，显示时间约 1s。第二次显示 08，显示时间也是 1s。对某个位置的内容有疑问时，就调用该子程序将数值显示出来。虽然这个写法会暂时停顿在显示值上，但是只要一完成除错后，就可将该段子程序删除，以免影响执行速度。

17-6 串行通信除错法

学 8051 的汇编语言一定要学会 8051 的串行通信，学 8051 的程序除错，想当然必定不能忽略串行通信的除错方法。数字显示或音响显示都不会比串行通信的方式来得直接有效。所谓的串行通信除错就是把要看的值转成标准的 ASCII 码，再从串行端口上传出。AT2051 控制板送出的是 RS485 的差动串行信号，只要你的个人电脑上装有 RS485 接口，就可以将此差动信号再度转为数字的串行信号，然后呈现在 PC 的屏幕上。图 17-4 为 RS485 转 RS232 接口，图 17-5 为 RS485 转 USB 接口。上述两种接口的详细规格数据可在旗威科技公司的网站（http://www.chipware.com.tw）上取得。下面是 PC 端的接收程序范例，程序一是 DOS 模式下的写法，程序二是 Windows 下 VB 的写法。在本书所附的光盘内也有相关的程序文件或安装文件。

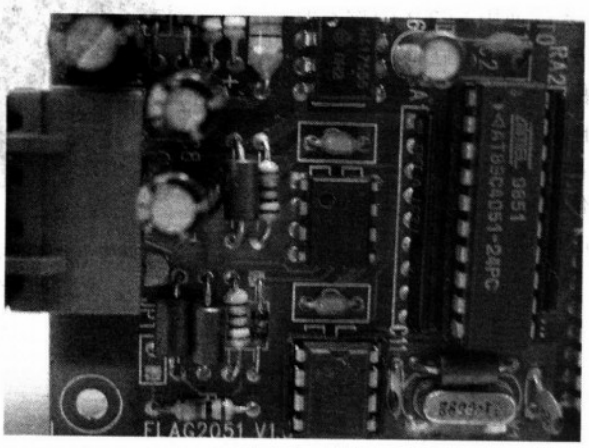


图 17-3 AT2051 控制板上也有串行通信接口，可以随时传回状态值，这是除错时最方便的工具

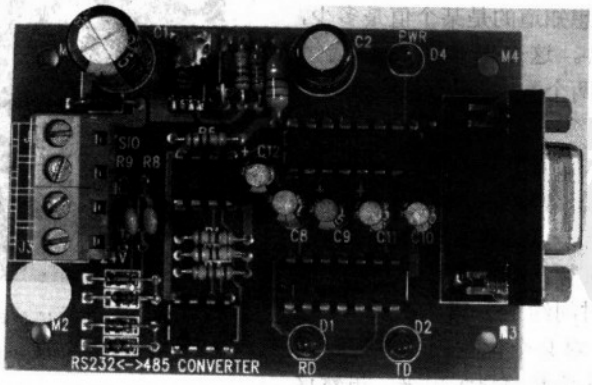


图 17-4 RS485 转 RS232 接口

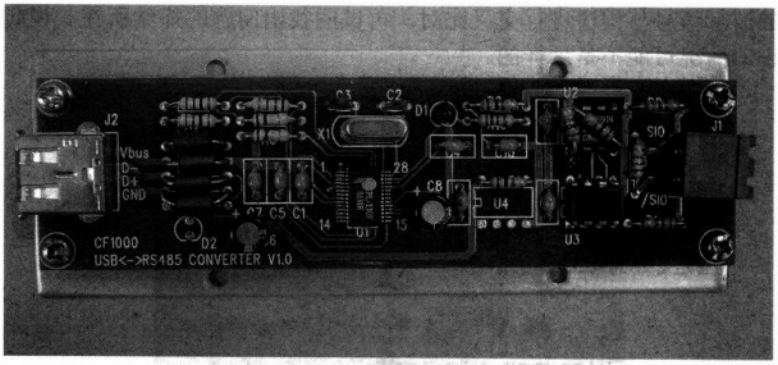


图 17-5　RS485 转 USB 接口

17-7　仪器协助除错法

本书主推这种除错方法，该方法靠一台数字示波器做辅助工具。只要系统有多余的硬件脚位，就能用 SETB 及 CLR 指令把系统的状态值显示出来。我们可以用示波器的波形储存的特殊功能记录子程序执行的时间，也可以查看中断 ISR 所花的时间，当然也可以用示波器确认收与送的串行信号。有些数字信号是出现一会儿就不见了，一般仿真示波器很难看出其中的变化，但是，数字式示波器却拥有此功能。我们觉得 21 世纪开始，数字式的示波器已经完全取代简易的仿真示波器。有了它的帮忙，可以使我们除错的时间减到最少。

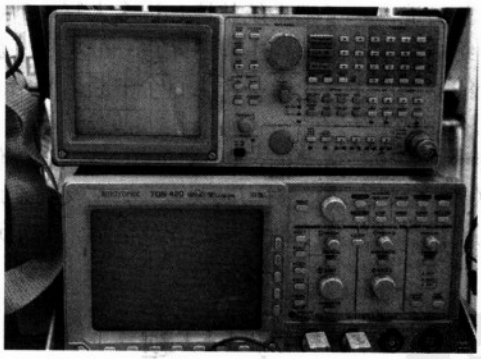


图 17-6　上方为频谱分析仪，下方为 100M 的示波器，若要 8051 软硬兼施的话，一定要学会利用示波器去除错

17-8　高级仪器除错法

这种除错方法适用对系统与程序相当熟悉的高级工程师，因为除非你对整个系统很熟悉，否则高级仪器摆在面前也不知道如何下手！所谓高级仪器就是示波器与逻辑分析仪（Logic Analyzer），示波器捕获特定的波形，逻辑分析仪则随时观察所有输出入的状态与时序变化，

然后对所有的波形信号与程序进行交叉比对，再试图找出问题的症结所在。说起来好像很容易，可是真正的情况可能不是这样的。

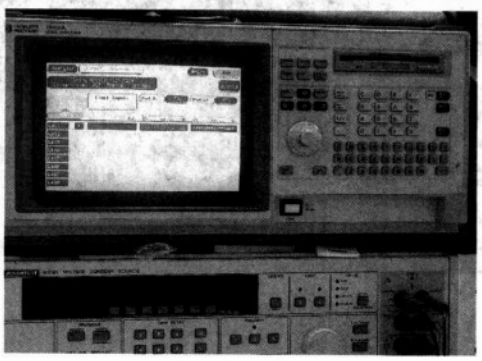


图 17-7　我们可以借用逻辑分析仪来做系统详细的除错

以 AT2051 控制板为例，当整个系统程序完成后，我们碰到一些问题：串行通信有时候不能把温度测量值完整地送出。整个程序里有定时中断每隔 10 ms 就发生一次，可是串行中断同时发生的时候，程序会变成何种样子是很难想象的，我们的除错方式就是分别把定时中断的执行时间与串行中断的执行时间显示出来，只要一看到波形就知道问题出现在哪里了。比如说，开机时定时中断都做得很好，可是串行中断后，定时中断的输出脉冲就不见了，那我们几乎可以断定串行中断的写法绝对有问题。

当我们写的程序越来越大时，除错的复杂度也会变得非常困难，主要的原因就是程序只要有一小段处理不妥，就可能一下子把系统整垮！许多人写串行中断程序以为先允许串行中断，让串行数据源源不绝地进来，然后在主程序才逐一分析程序就行了，可是实际情况是串行信号线受到其他信号的影响产生数十个甚至上百个中断信号给 8051，造成系统堆栈不够而溢出，在此之后系统就不正常了。

有些时候适当的禁止中断也是必要的，以 AT2051 控制板为例，当我们正在把重要数据写入串行 E^2PROM 的关键时刻，如果定时中断又发生，那又会造成何种严重的后果呢？比较正确的做法是关键时刻一律禁止任何中断，可是在此之后，一定还要解除此一禁令，否则系统会因无法再次中断，导致许多事情没有被执行。

习　题

1）在写汇编语言的程序时，需考虑哪些重点？
2）编写子程序时，要注意哪些事项？
3）程序的功能验证时机是什么？
4）如何利用 LED 来除错？
5）如何利用 BUZZER 来除错？
6）如何利用 DISPLAY 来除错？
7）串行通信的除错原理为何？
8）何谓高级仪器除错法？适用场合是什么？

18

两线式 RS485 的通信还是有一些绝缘存在，例如不能回应过快，也应该把传送数据的间隔拉开，这些都是用示波器看出来的

第 18 章　8051 例程归纳整理

8051 问世已超过 20 年了，许多常用的子程序（或称例程）都是随手可得的，在本章里我们做了整理，除此之外，你也可以上网去下载类似的子程序，例如“自动波特率推算”以及 LCM 的控制示范程序等。我们认为本章的重头戏是 4 字节的乘除运算，懂得如何运用的话，你就可以写更复杂的控制程序了。

写汇编语言程序就好像是登山爬岩一样，遇到紧要关头时要谨守“三点不动一点动”的原则，写程序时要牢记每次只做一部分修改后，就立即验证系统是否正常，千万不要等到修改了五、六个部分后，才想要一次测试这五六个部分的程序是否有 bug，因为程序间的互动效果可能会使除错变得非常复杂，所以请记得每次只修改一小部分，并且不要迟疑立即验证。

在本章里我们特地将许多相当有用的程序集合在一起，并且做了一些基本说明，由于这些程序的长度都不很长，最好每个程序都仔细看一次，大概知道其用意和作法，相信对写其他 8051 的应用程序时会有一些帮助的。

18-1　清除 4 个内部 DATA MEMORY 地址

```
;FUNCTION NAME:CLEAR_INTERNAL_DATA_MEMORY
;FUNCTION:CLEAR 4 BYTES DAYA MEMORY 08H,09H,0AH,0BH
;
CLEAR_INTERNAL_DATA_MEMORY
        MOV     R0,#08H         ;DATA MEMORY ADDR
        MOV     R2,#04H         ;COUNT
$1      CLR     A
        MOV     @R0,A
        INC     R0
        DJNZ    R2,$1
        RET
;
```

说明：

本程序清除内部 DATA MEMORY 地址（08H）、（09H）、（0AH）和（0BH）共 4 个字节的内容，R2 寄存器内的值是计数值，本程序可以改成清除整段 DATA MEMORY 内的值，此时只要把 R2 设成 80H 即可。

18-2　清除 4 个外部 DATA MEMORY 地址

```
;FUNCTION NAME:CLEAR_EXTERNAL_DATA_MEMORY
;FUNCTION:CLEAR 4 BYTES EXTERNAL DATA MEMORY, ADDR INDEX BY DPTR
;
```

```
CLEAR_EXTERNAL_DATA_MEMORY
        MOV     R2,#04H
        CLR     A
$1      MOVX    @DPTR,A
        INC     DPTR
        DJNZ    R2,$1
        RET
;
```

说明：

本程序清除 DPTR 所指的四个外部 DATA MEMORY 内的值，DPTR 的内容必须在调用前就已设置好。本程序也可以修改成对某段外部数据存储器设置成某个定数值。R2 是个计数值，内部存放清除的个数，由于 R2 的最大值是 255，所以最多可清除 256 个外部 DATA MEMORY 的值，如果想一次清除 2K（2048）Bytes 的外部数据存储器时，只需要先定好 DPTR 值，再将 R2 设成 00H，然后调用本程序 8 次即可。

18-3 将外部数据存储器上 4 个字节值存入内部数据存储器

```
;FUNCTION NAME:DATA_LOAD_4BYTE
;FUNCTION:LOAD 4 BYTES DATA MEMORY FROM EXTERNAL DATA MEMORY
;(R0+3)(R0+2)(R0+1)(R0)=(DPTR+3)(DPTR+2)(DPTR+1)(DPTR)
;
DATA_LOAD_4BYTE
        MOV     R0,#08H
        MOV     R2,#04H
$1      MOVX    A,@DPTR
        MOV     @R0,A
        INC     R0
        INC     DPTR
        DJNZ    R2,$1
        RET
;
```

说明：

由于 8051 内部数据存储器的空间不够，大部分数据仍然存在外部数据存储器为主，当这些数据要运算前，就必须将这些数据先放到内部数据存储器上，再和其他值做加减乘除运算，当然最后的运算结果还是要再转存到外部数据存储器中。在许多工业用的 8051 控制器上，外部数据存储器另外用充电电池做预备电源，所以这些资料和数据不会因电源关闭而消失。近年来由于 E^2PROM 的价格也逐渐下降，也有部分数据备份电路改采 E^2PROM，但是其写入的时间就不能像 SRAM 那么快了。

18-4 将 4 个内部数据值转存到外部数据存储器中

```
;FUNCTION NAME:DATA_SAVE_4BYTE
;FUNCTION:STORE 4 BYTES DATA MEMORY INTO EXTERNAL DATA MEMORY
;(DPTR+3)(DPTR+2)(DPTR+1)(DPTR)=(R0+3)(R0+2)(R0+1)(R0)
;
```

```
DATA_SAVE_4BYTE
        MOV     R2,#04H
$1      MOV     A,@R0
        MOVX    @DPTR,A
        INC     R0
        INC     DPTR
        DJNZ    R2,$1
        RET
;
```

说明：

本程序将 R0 所指定的地址值转存到外部的数据存储器中，总共转存了 4 个字节，外部数据存储器的开始地址则事先存入 DPTR 指针寄存器中，通常做完运算后会再调用本程序做数据储存的操作。

18-5 内部数据存储器内 4 字节相加（不含正负符号）

```
;FUNCTION NAME:DATA_ADD_4BYTE
;FUNCTION:ADD 4 BYTES IN DATA MEMORY
;(R0+3)(R0+2)(R0+1)(R0)=(R0+3)(R0+2)(R0+1)(R0)+(R1+3)(R1+2)(R1+1)(R1)
;
DATA_ADD_4BYTE
        MOV     R2,#04H
        CLR     C
$1      MOV     A,@R0
        ADDC    A,@R1
        MOV     @R0,A
        INC     R0
        INC     R1
        DJNZ    R2,$1
        RET
;
```

说明：

在内部数据存储器中做运算速度最快，本程序将 R0 和 R1 所指地址内的值相加，总共加了 4 字节，在做相加前特别清除了 CY 标志位，以免之前的标志位值影响运算的结果。

18-6 内部数据存储器的值和外部数据存储器的值相加

```
;FUNCTION NAME:DATA_ADD_4BYTE_EXTERNAL
;FUNCTION:ADD 4 BYTES
;(R0+3)(R0+2)(R0+1)(R0)=(R0+3)(R0+2)(R0+1)(R0)+(DPTR+3)(DPTR+2)(DPTR+1)(DPTR)
;
DATA_ADD_4BYTE_EXTERNAL
        MOV     R2,#04H
        CLR     C
$1      MOVX    A,@DPTR
```

```
        ADDC    A,@R0
        MOV     @R0,A
        INC     R0
        INC     DPTR
        DJNZ    R2,$1
        RET
;
```

说明：

本程序的执行速度较上个程序要慢，相加后的结果仍在 R0 所指的地址上，当程序还要再做其他运算时，可能还不需要储存运算结果，所以可以调用本程序做初步的加法运算。

18-7 内部数据存储器的 4 字节相减

```
;FUNCTION NAME:DATA_SUBTRACT
;FUNCTION:SUBTRACT 4 BYTES,RESULT IN DATA MEMORY
;(R0+3)(R0+2)(R0+1)(R0)=(R0+3)(R0+2)(R0+1)(R0)-(R1+3)(R1+2)(R1+1)(R1)
;
DATA_SUBTRACT
        MOV     R2,#04H
        CLR     C
$1      MOV     A,@R0
        SUBB    A,@R1
        MOV     @R0,A
        INC     R0
        INC     R1
        DJNZ    R2,$1
        RET
;
```

说明：

这是一个没有正负符号的 4 字节减法运算，由 R0 地址所指的内容减去 R1 地址所指的内容值，结果存在 R0 所指的地址内，开始相减前程序先做一个 CLR C 指令清除 CY 标志位，以免运算后和真实值差 1。

18-8 将内部数据存储器内的值取补码

```
;FUNCTION NAME: DATA_COMPLEMENT
;FUNCTION: COMPLEMENT 4 BYTES IN (07) (06) (05) (04)
;
DATA_COMPLEMENT
        MOV     R0,#04
        MOV     R2,#04H     ;COUNT
        SETB    C           ;CY=1
$1      MOV     A,@R0
        CPL     A
        ADDC    A,#00H      ;A=A+CY
```

```
        MOV     @R0,A
        INC     R0
        DJNZ    R2,$1
        RET
;
```

说明：

本程序将（07H）、（06H）、（05H）和（04H）共 4 字节值取其补码，取补码前程序先设置 CY 标志位，然后将取到的值反相后的 00H 和 CY 标志位，共重复 4 次，结果仍然在（07H）、（06H）、（05H）和（04H）上。

18-9 对外部数据存储器做 16 位的加法运算

```
;FUNCTION NAME: DPTR_ADD_1BYTE
;FUNCTION: (DPTR+1)(DPTR)=(DPTR+1)(DPTR)+ACC
;
DPTR_ADD_1BYTE
        MOV     B,A
        MOVX    A,@DPTR
        CLR     C
        ADD     A,B
        MOVX    @DPTR, A
        MOV     B,A
        INC     DPTR
        MOVX    A,@DPTR
        ADDC    A,#00H
        MOVX    @DPTR,A
        RET
;
```

说明：

写 8051 的应用程序中，时常碰到要将外部存储器上的 16 位值加上另一个 8 位值，这时就可利用本程序做加法运算，请注意第一个加法是用 ADD 指令，第二个加法则是用 ADDC 指令，结果仍存在外部存储器中。

18-10 对外部存储器做减法运算

```
;FUNCTION NAME: DPTR_SUB_1BYTE
;FUNCTION: (DPTR+1)(DPTR)=(DPTR+1)(DPTR)-ACC
        MOV     B,A
        MOVX    A,@DPTR
        CLR     C
        SUBB    A,B
        MOVX    @DPTR,A
        MOV     B,A
        INC     DPTR
        MOVX    A,@DPTR
```

```
        SUBB    A,#00H
        MOVX    @DPTR,A
        RET
;
```

说明:

本程序将外部存储器上的 16 位值减去 ACC 累加器值，结果仍放在外部存储器内，由于 SUBB 指令一定会减去 CY 标志位值，所以程序在减法运算前，先执行 CLR C 指令，以免计算错误。

18-11 内部数据存储器做值的比较

```
;FUNCTION NAME: INTERNAL_DATA_COMPARE
;FUNCTION: COMPARE 2 BLOCKS INTERNAL DATA MEMORY(R2)TIMES
;
INTERNAL_DATA_COMPARE
$1      MOV     A,@R0
        MOV     B,@R1
        CJNE    A,B,$2
        INC     R0
        INC     R1
        DJNZ    R2,$1
$2      RET
;
```

说明:

本程序对两段内部数据存储器做比较，总共比较了 R2 个字节，比较前应该先定义好 R0 和 R1 的值，以免比较了错误的结果，实际应用时，若最后的结果完全相同时，可再调用数据相符的程序，反之则程序就会出错。

18-12 外部数据存储器做整段值的比较

```
;FUNCTION NAME: EXTERNAL_DATA_COMPARE
;FUNCTION: COMPARE 2 BLOCKS EXTERNAL DATA MEMORY(R2)TIMES
;BLOCK 1 START AT DPTR
;BLOCK 2 START AT R7,R6
;
EXTERNAL_DATA_COMPARE
$1      MOVX    A,@DPTR
        MOV     B,A             ;BLOCK 1 DATA IN B
        PUSH    DPH
        PUSH    DPL
        MOV     DPH,R7
        MOV     DPL,R6
        MOVX    A,@DPTR         ;BLOCK 2 DATA IN A
        INC     DPTR
        MOV     R7,DPH
        MOV     R6,DPL          ;BLOCK 2 INDEX+1
```

```
        POP     DPL
        POP     DPH
        INC     DPTR        ;BLOCK 1 INDEX+1
        CJNE    A,B,$2      ;COMPARE 1 BYTE
        DJNZ    R2,$1
        MOV     A,#00H      ;NO ERROR
        SJMP    $2+2
$2      MOV     A,#01H      ;ERROR
        RET
;
```

说明：

这个程序比较外部数据存储器中的两个区，区 1 的开始位置存在 DPTR 上，区 2 的开始位置则存在 R7 和 R6 上，由于 8051 单片机 CPU 只有一组 16 位的地址索引，所以本程序中有计多指令都在做 DPTR 值的移动操作。当初 Intel 的工程序若多加一个 16 位索引寄存器，或许上面的程序就可以少掉好几行了。

18-13 内部数据存储器区与累加器做比较

```
;FUNCTION NAME: ACC_COMPARE
;FUNCTION: COMPARE ACC WITH (R0)IN DATA MEMORY(R2)TIMES
;
ACC_COMPARE
$1      MOV     B,@R0
        CJNE    A,B,$2
        INC     R0
        DJNZ    R2,$1
        CLR     A
        SJMP    $3          ;IF FOUND THEN A=0
$2      MOV     A,#01H      ;ELSE    A=1
$3      RET
;
```

说明：

内部数据存储器开始比较的地址存在 R0 寄存器上，总共比较了 R2 个字节，结果存在 ACC 上。若 ACC=0 代表该区内的值和 ACC 值完全相同，反之，ACC=1 时代表有值和 ACC 不相同。

18-14 4 字节不含正负符号的乘法运算

```
;FUNCTION NAME: MUL_4BYTE
;FUNCTION: UNSIGNED 4 BYTE MULTIPLY
;(07H)(06H)(05H)(04H)=(07H)(06H)(05H)(04H)X(0FH)(0EH)(0DH)(0CH)
;WORKING AREA DATA MEMORY:(0BH),(0AH),(09H),(08H)
;
MUL_4BYTE
        LCALL   CLEAR_INTERNAL_DATA_MEMORY
```

```
        MOV     R3,#33      ;SHIFT COUNT
        CLR     C
$1      MOV     R1,#0BH
        MOV     R2,#08H
$2      MOV     A,@R1
        RRC     A
        MOV     @R1,A
        DEC     R1
        DJNZ    R2,$2
        JNC     $3
        CLR     C
        MOV     R0,#08H
        LCALL   DATA_ADD_4BYTE
$3      DJNZ    R3,$1
        RET
;
```

说明：

本乘法运算并未使用 8051 的 MUL 指令，仍然沿用类似加法的乘算过程，常用的 8 位 CPU 如 Z-80、6502 或 8085 也都是采用这种计算过程，对乘法演算过程有兴趣的读者，最好跟着程序算一次，你就会有“原来就是如此”的感觉，这种演算步骤在任何情况都是准确的，不管 CPU 如何替换，但是其计算的原理是不会改变的。本程序另外利用 4 个地址（0BH）、（0AH）、（09H）和（08H）做临时计算区，请不要将程序重要的数据存在这 4 个临时运算区中。

18-15　4 字节不含正负符号的除法运算

```
;FUNCTION NAME: DIV_4BYTE
;FUNCTION:UNSIGNED 4 BYTE DIVIDE
;(07H)(06H)(05H)(04H)=(07H)(06H)(05H)(04H)/(0FH)(0EH)(0DH)(0CH)
;WORKING AREA DATA MEMORY:(0BH),(0AH),(09H),(08H)
;
DIV_4BYTE
        LCALL   CLEAR_INTERNAL_DATA_MEMORY
        MOV     R3,#32      ;SHIFT COUNT
        CLR     C
$1      MOV     R1,#04H
        MOV     R2,#08H
$2      MOV     A,@R1
        RLC     A
        MOV     @R1,A
        INC     R1
        DJNZ    R2,$2
        MOV     R0,#0BH
        MOV     R1,#0FH
        MOV     R2,#04H
```

```
        LCALL   INTERNAL_DATA_COMPARE
        JC      $3
        MOV     R0,#08H
        MOV     R1,#0CH
        LCALL   DATA_SUBTRACT
$3      CPL     C
        DJNZ    R3,$1
        MOV     R1,#04H
        MOV     R2,#04H
$4      MOV     A,@R1
        RLC     A
        MOV     @R1,A
        INC     R1
        DJNZ    R2,$4
        RET
```

说明：

这是一个 4 字节的除法运算，被除数存在（07H）、（06H）、（05H）和（04H）上，除数则摆在（0FH）、（0EH）、（0DH）和（0CH）中，程序另外占用（0BH）、（0AH）、（09H）和（08H）做临时计算存放区，结果则存在（07H）、（06H）、（05H）和（04H）里，程序中完全没有用到 DIV 指令，这不禁让人感到怀疑，Intel 当初特地开发的 MUL 和 DIV 指令是否真的有其实用性，还是纯粹是商业上的要求？

18-16 对外部数据存储器内的值做异或运算产生一个校验码

```
;FUNCTION NAME: GENERATE_CHECK_BYTE
;FUNCTION: GENERATE 1 CHECK BYTE FOR 4 BYTES DATA IN EXTERNAL DATA MEMORY
;CHECK BYTE=(DPTR+3)XOR(DPTR+2)XOR(DPTR+1)XOR(DPTR)XOR55H
;
GENERATE_CHECK_BYTE
        MOV     B,#00H
        MOV     R2,#04H;COUNT
$1      MOVX    A,@DPTR
        XRL     A,B
        MOV     B,A
        INC     DPTR
        DJNZ    R2,$1
        XRL     A,#55H
        MOVX    @DPTR,A
        RET
```

说明：

在一个充分被保护的控制系统中，要求所有储存的数据或数据都是正确的，只要数据被干扰或未经合法的写入时，系统都能立刻检测出来，此时的检测程序写法就非常重要了。本

程序将存在外部存储器的一连串 4 个字节做值异或的运算，然后再和值 55H 做一次异或（XOR），并将异或后的结果存入。经过这样的运算后，外部存储器中 4 个字节内任何一个位被修改时，都会导致校验码的错误，无形之中就保障值的正确性了，只要我们做重要计算前，先确认计算值的校验码是否正确，即可保证计算结果的正确。以单片机控制的 ABS 防锁死刹车系统为例，汽车刹车时，以每秒 8 次的速度对刹车系统做 ON-OFF 的操作，其内部程序就做了万全的保护，不能有任何闪失的情况发生，这类的控制系统内部一定有类似本程序的校验码存在。

18-17 确认外部数据存储器（4 字节）的校验码是否正确

```
;FUNCTION NAME:TEST_CHECK_BYTE
;FUNCTION:TEST CHECK BYTE IS OK OR NOT
;
TEST_CHECK_BYTE
        MOV     B,#00H
        MOV     R2,#04H      ;COUNT
$1      MOVX    A,@DPTR
        XRL     A,B
        MOV     B,A
        INC     DPTR
        DJNZ    R2,$1
        XRL     A,#55H
        MOV     B,A          ;TEST BYTE
        MOVX    A,@DPTR      ;CHECK BYTE
        CJNE    A,B,$2
        MOV     A,#00H       ;IF CORRECT THEN RETURN 0
        SJMP    $3           ;ELSE      RETURN 1
$2      MOV     A,#01H
$3      RET
;
```

说明：

上个程序的校验码存入外部数据存储器后，可再靠本程序来确认数据是否有被修改过，4 个字节的数据加上校验码共有 40 位，其中只要有一个位不正确时，都会得到一个完全不同的校验码。此程序可用在系统重要数据的确认上，若校验码正确则继续程序，反之则进入数据错误的循环中，并发出错误信息给操作人员，以便进行维修。

18-18 在内部数据存储器内产生 4 个随机数

```
;FUNCTION NAME: GENERATE_RANDOM_NUMBER
;FUNCTION: GENERATE RANDOM NUMBER (4 BYTES)
;
RAND1     EQU    40H
RAND2     EQU    RAND1+1
RAND3     EQU    RAND1+2
```

```
RAND4     EQU   RAND1+3
;
GENERATE_RANDOM_NUMBER
          MOV   A,RAND1
          RRC   A
          RRC   A
          RRC   A
          XRL   A,RAND3
          RRC   A
          RRC   A
          XRL   A,RAND2
          RRC   A
          XRL   A,RAND4
          RRC   A
          CPL   C
          MOV   R0,#RAND1
          MOV   R2,#04H
$1        MOV   A,@R0        ;SAVE RANDOM NUMBER
          RLC   A
          MOV   @R0,A
          INC   R0
          DJNZ  R2,$1
          RET
;
```

说明：

当系统控制需要随机数时，可以由此程序中得到 4 个随机数，本程序应该安排在定时中断程序中，每秒固定产生 10 组以上的随机数码，而不是需要随机数时才来调用本程序，随机数产生器都是靠异或和移位运算而得到分布非常均匀的随机数码。

18-19 检查外部数据存储器（16 位）是否为 0000H

```
;FUNCTION NAME: CHECK ZERO
;FUNCTION: CHECK IF (DPTR+1) (DPTR) =0000H
;
CHECK_ZERO
          MOV   R2,#02H
$1        MOVX  A,@DPTR
          JNZ   $2
          INC   DPTR
          DJNZ  R2,$1
$2        RET
;
```

说明：

本程序在调用前应当先设置好 DPTR 的值，R2 上的值是做计数值用，也可改成其他值以检查多个地址上的值是否为 00H。

18-20 检查外部数据存储器（16 位）的值是否为 1000

```
;FUNCTION NAME: CHECK_EQUAL
;FUNCTION: CHECK IF(DPTR+1)(DPTR)=03E8H(DECIMAL 1000)
;
CHECK_EQUAL
        INC     DPTR
        MOVX    A,@DPTR      ;CHECK(DPTR+1)
        CJNE    A,#03H,$1
        DEC     DPL
        MOVX    A,@DPTR
        CJNE    A,#E8H,$1
        CLR     A
        SJMP    $2           ;IF EQUAL THEN RETURN 0
$1      MOV     A,#01H       ;ELSE       RETURN 1
$2      RET
;
```

说明：

类似检查值的程序都由高位开始检查，若不正确时也就不必比较低位上的值了。

18-21 检查外部数据存储器（16 位）的值是否比 5000 大

```
;FUNCTION NAME:CHECK_LARGE
;FUNCTION:CHECK IF(DPTR+1)(DPTR)>1388H(DECIMAL 5000)
;
CHECK_LARGE
        INC     DPTR
        MOVX    A,@DPTR
        CJNE    A,#13H,$1
        DEC     DPL
        MOVX    A,@DPTR
        CJNE    A,#88H,$1
$1      JNC     $2
        CLR     A            ;IF SMALLER THEN RETURN 0
        SJMP    $3
$2      MOV     A,#01H
$3      RET
;
```

说明：

若被比较的值大于 5000 时，ACC 值被设成 1，反之，ACC 值被清除成 0，程序中的比较值 1388H 是如何得到的？

方法有 3 种：

（1）利用 PC 机的程序计算出来。

（2）利用科学用计算器转换十进制码而得到。

(3) 用普通计算器按了许多键后才得到的结果。

表 18-1 是常用的十进制值和其十六进制值的对照表。

表 18-1 常用的十进制值和其十六进制值的对照表

十 进 制 值	十六进制值	十 进 制 值	十六进制值
100	0064H	4000	0FA0H
200	00C8H	5000	1388H
300	012CH	6000	1770H
400	0190H	7000	1B58H
500	01F4H	8000	1F40H
1000	03E8H	9000	2328H
2000	07D0H	10000	2710H
3000	0BB8H		

18-22 将外部数据存储器（16 位）值转换成 6 个 BCD 码

```
;FUNCTION NAME: HEX_TO_BCD
;FUNCTION: CONVERT 2 BYTE HEX TO 6 BYTE BCD CODE
;HEX DATA   IN(DPTR+1)(DPTR)
;BCD RESULT IN (09H)(08H)(07H)(06H)
;
HEX_TO_BCD
        MOV     R0,#04H
        MOVX    A,@DPTR
        MOV     @R0,A
        INC     R0
        INC     DPTR
        MOVX    A,@DPTR
        MOV     @R0,A
        INC     R0
        LCALL   CLEAR_INTERNAL_DATA_MEMORY+2    ;CLEAR(06)(07)(08)(09)
        MOV     09H,#16         ;COUNT
$1      MOV     R0,#04H         ;HEX DATA
        MOV     R2,#02H
        CLR     C
$2      MOV     A,@R0           ;SHIFT LEFT 1 BIT WITH CARRY
        RLC     A
        MOV     @R0,A
        INC     R0
        DJNZ    R2,$2
        MOV     R2,#03H
$3      MOV     A,@R0           ;A=(06H)
        ADDC    A,A             ;A=A+CARRY
        DA      A               ;DECIMAL  ADJUST
        MOV     @R0,A
```

```
        INC     R0
        DJNZ    R2,$3
        DJNZ    09H,$1
RET
```

说明：

8051 内部运算都是二进制的，但是如果要把值显示出来时，就需要本程序做十进制的转换。只要有碰到二进制到十进制的数值转换时，必定免不了会使用 DA　A 指令，本程序正是一个典型的例子。

16 位的最大值是 65535，所以本程序执行后，真正的结果存在（08H）、（07H）和（06H）上，而（09H）则始终保持 00H。

18-23　将 ACC 值（<99）转换成两个 BCD 码

```
;FUNCTION NAME: BYTE_TO_2BCD
;FUNCTION: CONVERT 1 BYTE(<99)TO BCD CODE
;HEX DATA IN ACC
;BCD RESULT IN ACC
;WORKING AREA :(04H)(05H)
;
BYTE_TO_2BCD
        MOV     04H,A
        MOV     05H,#00H
        MOV     R2,#08H
$1      MOV     A,04H
        ADD     A,A
        MOV     04H,A
        MOV     A,05H
        ADDC    A,A
        DA      A
        MOV     05H,A
        DJNZ    R2,$1
        MOV     A,05H
        RET
;
```

说明：

本程序只能将小于 99 的累加器值转换成两位数的 BCD 码，结果仍存放在累加器 ACC 内。

18-24　将累加器的值转换成 3 个 BCD 码

```
;FUNCTION NAME:BYTE_TO_3BCD
;FUNCTION:CONVERT 1 BYTE TO 3 BCD CODE
;HEX DATA IN ACC
;BCD RESULT IN (06)(05)(04)
;
BYTE_TO_3BCD
```

```
        MOV     B,#100
        DIV     AB              ;A=A/100,B=A MOD 100
        MOV     06H,A           ;SAVE HUNDRED'S DIGIT
        MOV     A,#10
        XCH     A,B
        DIV     AB              ;A=A/10
        MOV     05H,A           ;TEN'S DIGIT
        MOV     04H,B           ;ONE'S DIGIT
        RET
;
```

说明：

本程序摘自 Intel 8051 使用手册的程序说明一章，此时使用 DIV 指令的确可以简化程序的写法，转换后的结果存在（05H）和（04H）两个地址中。

18-25 检查一段外部数据存储器（2KB）的读写功能

```
;FUNCTION NAME: CHECK_RAM_RW
:FUNCTION: CHECK RAM READ/WRITE FUNCTION (2048 BYTES)
;RAM ADDRESS IN DPTR
;
CHECK_RAM_RW
        MOV     R0,#08H
$1      MOV     R1,#00H
$2      MOVX    A,@DPTR         ;READ
        XRL     A,#FFH
        MOV     B,A
        MOVX    @DPTR,A         ;WRITE
        MOVX    A,@DPTR         ;READ AGAIN
        CJNE    A,B,$3          ;READ/WRITE ERROR
        XRL     A,#FFH
        MOVX    @DPTR,A         ;WRITE AGAIN
        INC     DPTR
        DJNZ    R1,$2
        DJNZ    R0,$1
        MOV     A,#00H          ;IF NO ERROR THEN RETURN 0
        SJMP    $3+2
$3      MOV     A,#01H
        RET
;
```

说明：

本程序可以检查 DPTR 开始的 2KB 存储器地址的读出及写入功能，若完全正确时，ACC 值等于 0，反之 ACC 值等于 1，若该段地址确实为 RAM 区时，程序执行完后，RAM 区内的值仍然和程序执行前相同。

18-26 计算 2KB 程序空间的校验和（CHECKSUM）

```
;FUNCTION NAME: CALCULATE 2K PROGRAM MEMORY CHECKSUM
```

```
;FUNCTION: CHECKSUM(05)(04)=(DPTR+2048)+··· +(DPTR)
;
CHECKSUM
        MOV     R4,#00H
        MOV     R5,#00H
        MOV     R2,#08H
$1      MOV     R1,#00H
$2      MOVC    A,@A+DPTR       ;READ PROGRAM DATA
        ADD     A,04H
        MOV     04H,A
        CLR     A
        ADDC    A,05H
        MOV     05H,A
        INC     DPTR
        DJNZ    R1,$2
        DJNZ    R2,$1
        RET                     ;RESULT IN(05)(04)
;
```

说明：

本程序在调用前必须先设置 DPTR 的值，总共计算了 2048 个程序存储器内的值，结果存在（05H）和（04H）上，这段程序通常加在程序起始阶段，用以判断程序内容是否有被其他人改动过。

18-27　清除外部数据存储器共 2048 个地址

```
;FUNCTION NAME: CLEAR_EXTERNAL_MEMORY
;FUNCTION: CLEAR 2K BYTES EXTERNAL DATA MEMORY
;EXTERNAL DATA MEMORY ADDRESS IN DPTR
;
CLEAR_EXTERNAL_MEMORY
        CLR     A
        MOV     R2,#08H
$1      MOV     R1,#00H
$2      MOVX    @DPTR,A
        INC     DPTR
        DJNZ    R1,$2
        DJNZ    R2,$1
        RET
;
```

说明：

本程序通常加在程序启始阶段，对外部的数据存储器进行内容清除的操作，可是如果外部存储器已有充电电池做电源供应时，这项清除操作就得视情况加入或取消了。

18-28　将 1 个字节值转换成 ASCII 码，供数据显示用

```
;FUNCTION NAME: HEX_TO_ASC
```

```
;FUNCTION:1 BYTE HEX CODE TO ASCII CONVERSION
;HEX CODE IN ACC
;RESULT IN ACC
;
HEX_TO_ASC
        MOV     DPTR,#ASCII_TABLE
        PUSH    A
        ANL     A,#F0H
        SWAP    A
        MOVC    A,@A+DPTR
        SWAP    A
        MOV     B,A
        POP     A
        ANL     A,#0FH
        MOVC    A,@A+DPTR
        ORL     A,B
        RET
;
ASCII_TABLE
        DB      '0123456789ABCDEF'
;
```

说明：

当要把系统中的二进制值显示出来时，一定要调用本程序，程序实际上只是个查表的操作，输入值和结果都放在 ACC 累加器上。

18-29 将 ASCII 码转换成二进制

```
;FUNCTION NAME: ASCII_TO_BINARY
;FUNCTION: CONVERT 1 BYTE ASCII CODE TO BINARY CODE
;ASCII CODE IN ACC
;RESULT IN ACC
;EXAMPLE: 'F6'RETURN 0F6H,'3E'RETURN 03EH
;
ASCII_TO_BINARY
        PUSH    B
        PUSH    A
        ANL     A,#F0H
        SWAP    A
        LCALL   ASC_CONV
        SWAP    A
        MOV     B,A             ;D7-D4
        POP     A
        ANL     A,#0FH
        LCALL   ASC_CONV        ;D3-D0
        ORL     A,B
        POP     B
        RET
;
```

```
ASC_CONV:
        SUBB    A,#'9'+1        ;A=A-3AH
        JNC     $1              ;'9'<ACC<'F'
        ADD     A,#10           ;A=A-30H
        SJMP    $2              ;'0'<=ACC<='9'
$1      ADD     A,#('9'+1-'A'+10)
$2      RET
;
```

说明：

本程序将 ASCII 码转换成二进制码，刚好和上个程序的操作相反，当数据通信以 ASCII 码为主时，必定需要此程序的运行，才能获得正确的二进制值。

习　题

1）如何清除内部 DATA MEMORY 地址？并试写一个程序。

2）如何将内部 DATA MEMORY 的数据值转存到外部 DATA MEMORY 中？并试写一个程序。

3）如何对外部 DATA MEMORY 做 16 位的加法运算？并试写一个程序。

4）如何运算不含正负符号的除法运算？并试写一个程序。

5）如何确认外部 DATA MEMORY 的检验码是否正确？并试写一个程序。

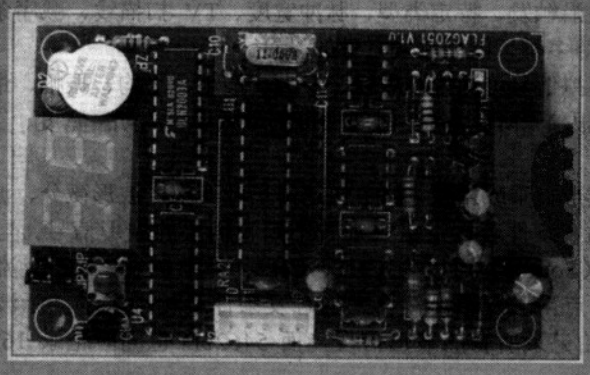
6）如何计算 2KB 程序空间的 checksum 值？并试写一个程序。

7）如何将 ASCII 码转化成二进制码？并试写一个程序。

8）简单说明汇编语言程序的修改原则。

工具善用篇

19



AT2051 控制上有声音很大的蜂鸣器，我们可以用程序设定温度的上限值，当温度超过时就发出声音

第 19 章　混合式示波器的认识与使用

在 8051 单片机的开发过程中，我们经常使用示波器来检测信号及软硬件除错。当产品进入生产阶段时，我们也是使用示波器来确认振荡线路是否正常，以及串行通信的数据是否有所漏失。在产品保修时，我们还是应用示波器来观看该出现的信号是否出现。学习示波器就好像汽车驾照一样一定会用到的。

写程序需要用示波器吗？当然问题并不如此简单，若纯粹是写资料库的程序时，除了一台稳定的 PC 外，是不需要其他的硬件配合。可是若换成需要软硬件高度配合的 8051 程序时，我们倒认为有一台示波器来当辅助工具会更快让我们进入角色中。一般的数字式示波器有 2～4 个 CH 测量频道（channel），我们这里示范的示波器 Agilent54622D 除了有 2CH 的模拟信号外，还有 16 个数字 CH 可显示，能够很容易地把模拟与数字的信号状态同时显现出来，原厂把这类型的示波器称为混合式的示波器（Mixed Signal Oscilloscope）。

19-1　仪 器 规 格

（1）模拟频道：200M Sampling/s，每个 CH 的容量是 2M。

（2）数字频道：400M Sampling/s，每个 CH 最大可达 8M，可测量最小脉冲宽度 5ns。

（3）模拟垂直调整范围：1mV/div～5V/div。

（4）水平调整范围：5ns/div～50s/div。

（5）触发条件：边沿触发、特定类型、脉冲宽度、TV 扫描线、顺序条件式、持续条件式、I^2C 及自动搜索等等多种智能型触发条件的设置。

（6）测量功能：模拟信号可测量峰对峰、最大、最小、平均、振幅、最高值、最低值、过激值 overshot、均方根值 RMS 等等。时间轴可测量频率、周期、宽度、工作周期、上升时间、下降时间等等。

（7）波形运算：波形相乘或相减、FFT 快速傅里叶变换，波形微分、波形积分等等。

（8）触发条件：触发条件若设定错误则看不到我们要的波形信号，与一般示波器相同的 54622D 的触发源（Source）可以为 CH1、CH2、外部触发、AC 交流信号触发，也可以指定 16 个数字输入点当触发源。接下来的触发条件就相当多了，示波器一定要等到触发条件成立后，才开始记录所有的波形信号，而 54622D 也同时记录 16 个数字信号的所有变化。

图 19-1　54622D 示波器的外观

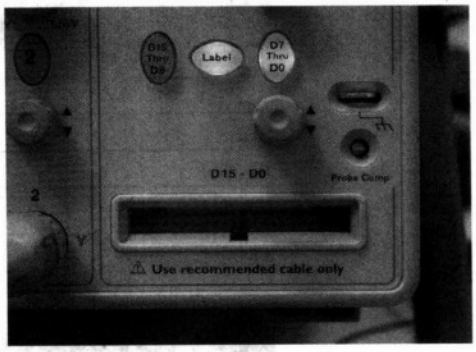


图 19-2　54622D 的 16 字节数字输入端与其接线方式

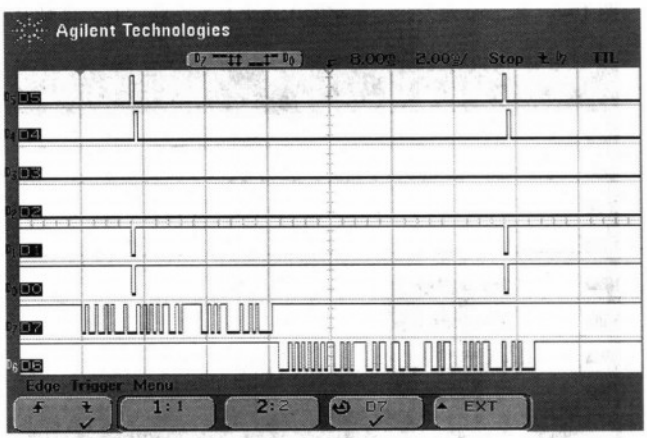


图 19-3　Edge 触发的条件设置

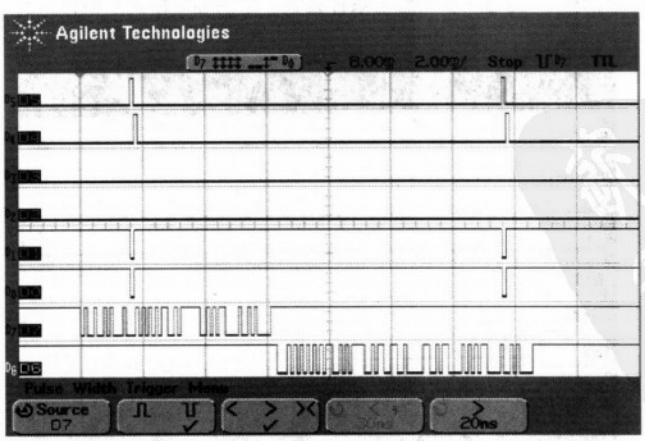


图 19-4　Pulse Width 脉冲的宽度也可以当成触发的条件

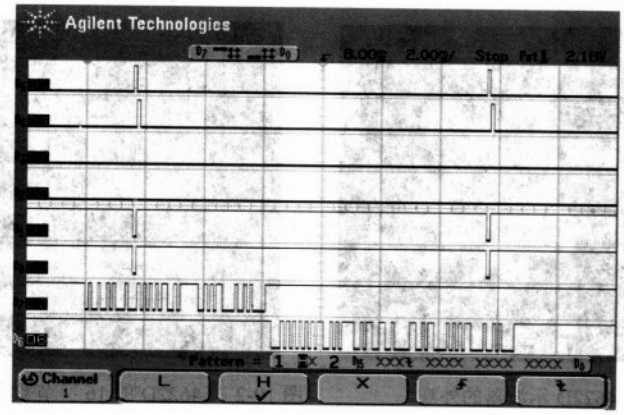


图 19-5　Pattern 触发，当数字输入端上是某个状态时才做触发

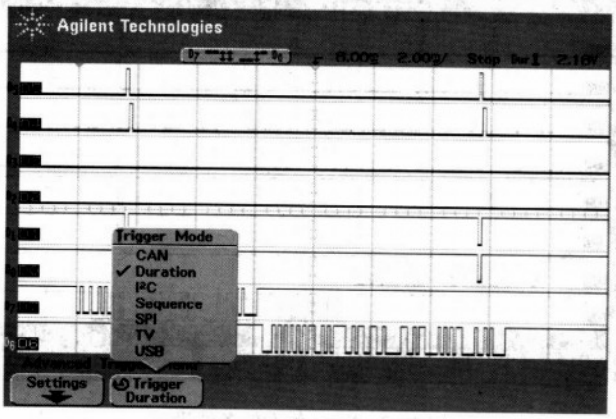


图 19-6　更进一步的触发条件设置，这些触发模式是很特殊的

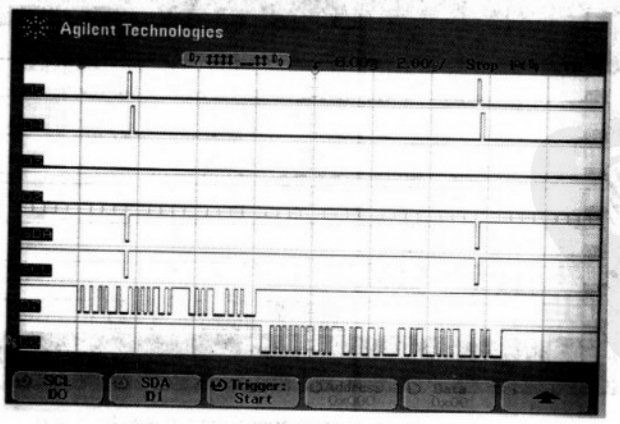


图 19-7　以 I²C 做触发时，还有许多状态可以选择

19-2 基本测量示范

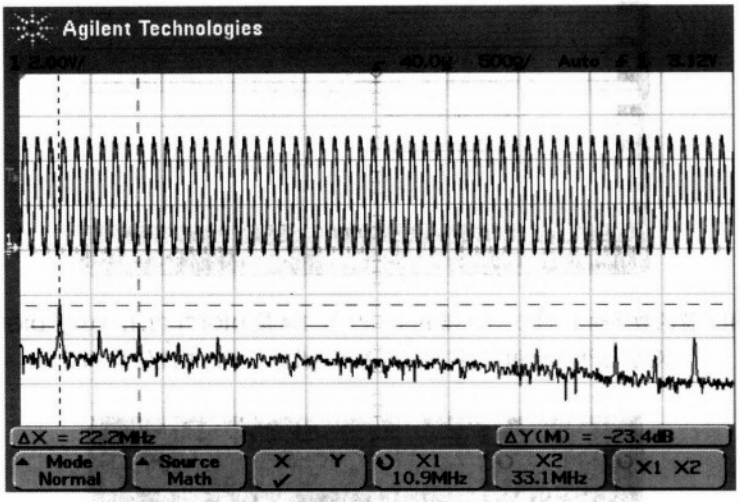


图 19-8 测量 AT2051 控制板的石英振荡输出 CH1，并且把 FFT 功能打开，可以看到 11.0592MHz 的基频及奇次谐波 33MHz

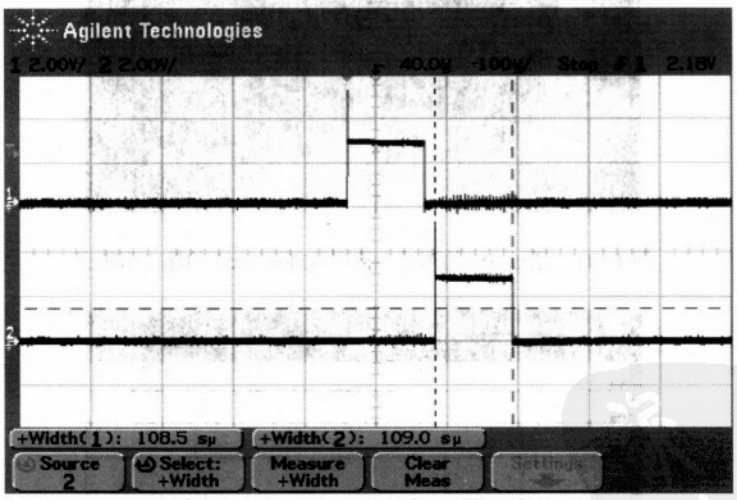


图 19-9 测量 AT2051 控制板的七段显示输出，CH1 为 DIGIT1 输出，CH2 为 DIGIT2 输出，我们另外度量其为 HI 的时间，都为 109 μs 左右。DIGIT1 与 DIGIT2 间有一小段时间隔开

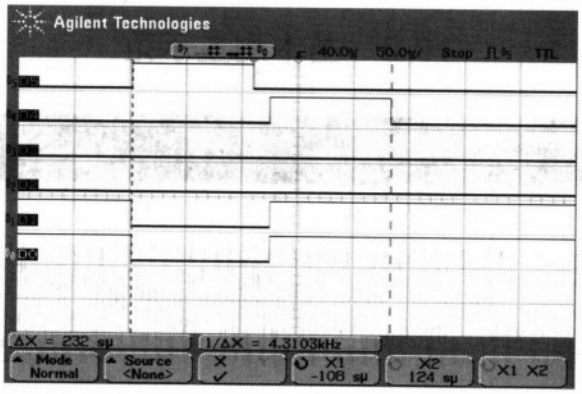


图 19-10　用 6 位的数字输入（D0～D3 接 P1.0～P1.3，D4 接 DIGIT1 P1.4，D5 接 DIGIT2 P1.5），观察七段显示输出与位数间的关系，此时显示的温度值是 30℃

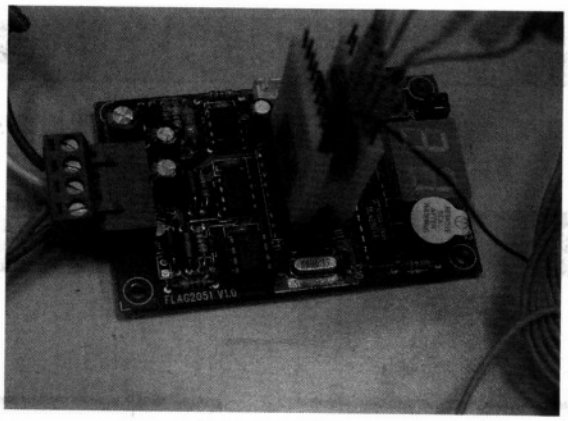


图 19-11　通过 3M 的测试夹接到 AT89C2051 的输出引脚上

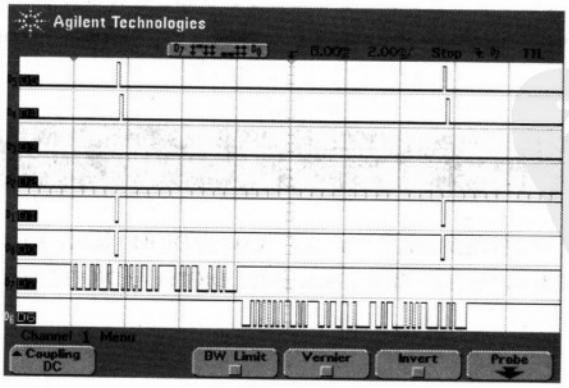


图 19-12　用 8 位的数字输入（D6 接 8051 的 RxD，D7 接 TxD）观察串行通信与定时中断间的关系

19-3 特殊信号测量

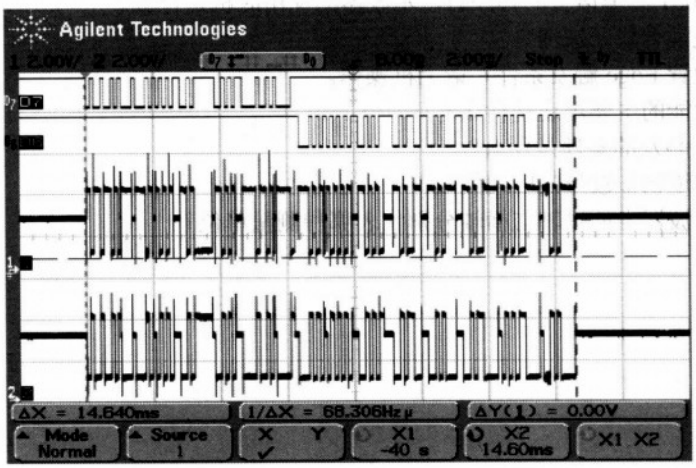


图 19-13 串行信号与差动信号的观察

CH1：差动的正向输出 SIO
CH2：差动的负向输出 SIO
D7：AT89C2051 的 RxD 引脚
D6：AT89C2051 的 TxD 引脚

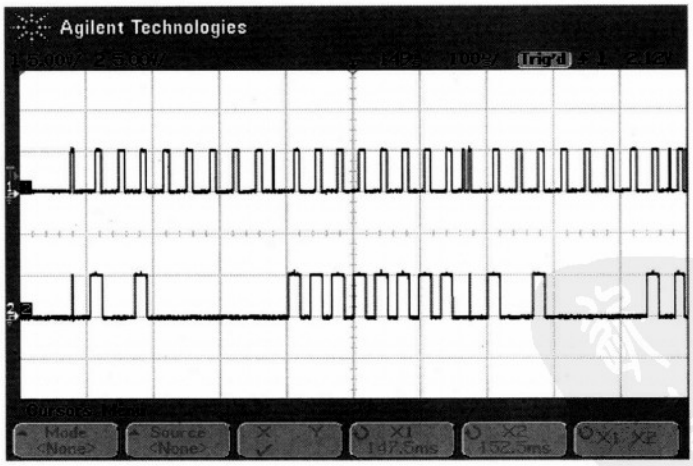


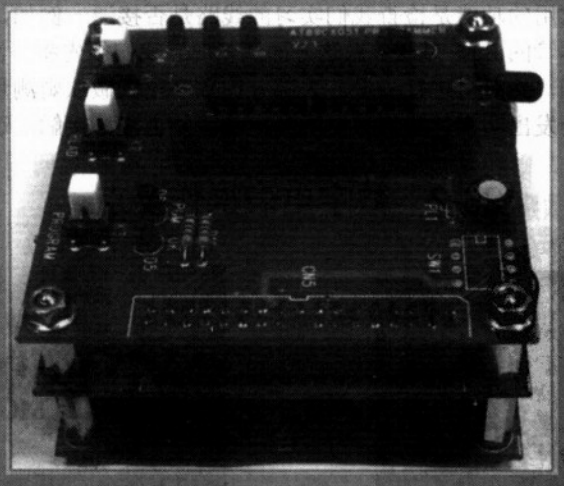
图 19-14 I2C 信号的观察

CH1：SCL 的信号，CH2：SDA 的信号。

习　题

1）编写 8051 汇编语言的程序时，示波器的使用时机是什么？
2）什么是混合式示波器？
3）如何设置 Edge 触发条件？请上机操作。
4）FFT 功能的主要用途是什么？
5）Pattern 触发的条件是什么？
6）如何知道已知波形的时间间隔？
7）一般示波器上的纵轴与横轴分别代表哪种测量单位？

20



旗威科技公司的 USB 烧录器，透过 USB 接线与 PC 连接，不需要外加任何电源，体积小携带方便

第 20 章　数字电表的使用

我们把数字电表列为工程师的绝对必备工具，当硬件初步被制造好后，我们会用数字电表来确认稳压电路是否正常，以及线路是否有异常的开路或断路。如果 8051 输出的数字信号不是很快时，也可以借助数字电表来观察信号的变化。平常，我们会把数字电表放在电脑桌上，当对任何直流电压有疑问时，立刻就拿数字电表进行验证。

每位设计工程师旁都有一些检测工具，其中最方便的工具就属数字电表了，我们最常拿它来测量直流电压、交流电压以及做电阻值的测量。一台好的数字电表至少可以陪伴工程师超过十年以上的时间，所以谨慎选择一台高精度且稳定耐用的数字电表绝对比随意购买来得重要，选对仪器比买错后又重新购买要划算。

20-1　数字电表功能

数字电表内部最重要的就是一个精准的直流电压表，各个测量文件最后都会用硬件转成直流电压后，才进行测量及做适当的单位显示。所以数字电表准不准确，看其 DCV 直流电压挡的精度就知道了。在实际测量中，我们用直流电压挡的几率也最高。使用率第二高的是 ACV 交流电压挡，我们通常用 ACV 挡来检查市电是否接入，也同时观看送来的交流电是否有不正常的压降。通常在偏远的乡村 AC220V 的电压变动率才会较大。

数字电表另一个常用的功能是检查我们实习的线路是否接通。例如，查看这个电阻是否有顺利接到另一个晶体管的 Base 基极。做这方面测量时，我们须先将待测电路断电，然后把数字电表切换到欧姆挡，然后再进行测量。大部分的数字电表做欧姆测量时，若测得的值很小，让内部的蜂鸣器会发出叫声，我们只要听到声响就知道测棒接触的这两点是相通的。

20-2　数字电表操作要点

图 20-1　好的电表会在输入端标示其等级

图 20-2　好的电表测棒会标示其测量等级

图 20-3　数字电表外部全部用高绝缘性且耐摔的材质包围

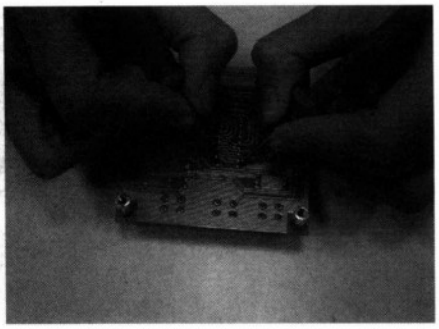


图 20-4　不正确的测棒握法，双手的拇指应该摆在测棒保护环的后面，这种握法存在被电击的可能

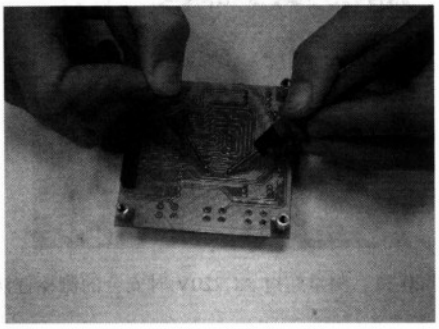


图 20-5　正确的测棒握法，双手要离电路板有一点距离

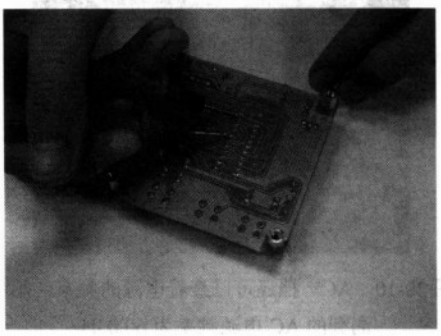


图 20-6　单手测棒的握法，就好像拿筷子一样

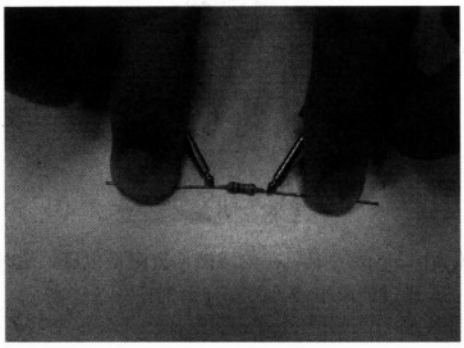


图 20-7　测量电阻错误的握法，人体刚好与电阻并联

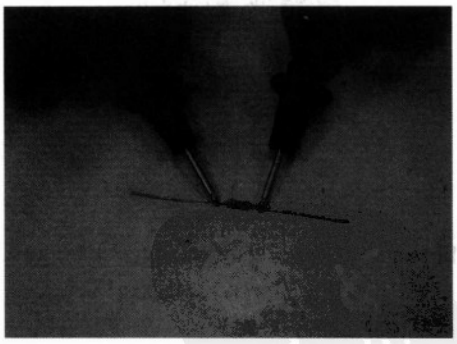


图 20-8　测量电阻正确的握法，元件下方应该放绝缘性高的材质

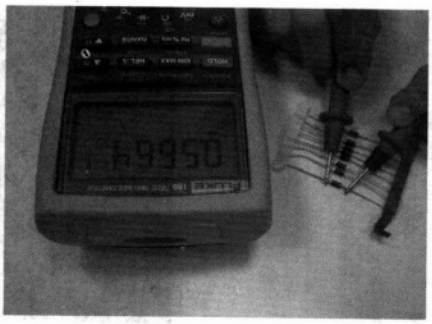


图 20-9　测量二极管的导通性能的方法，图中测到的电压为 0.5664V

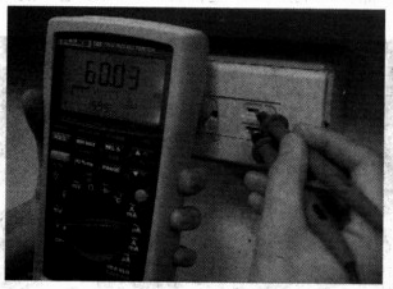


图 20-10　ACV 挡还可以查看电源的频率，图中测到的 AC 电源频率为 60.03Hz

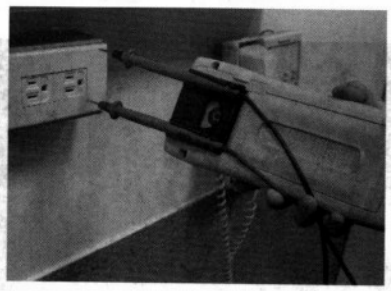


图 20-11　测量超过 AC220V 时安全的测棒握法

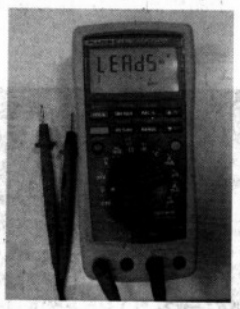


图 20-12　测量电流时电表要和电路串联，输入端也要换到电流输入端

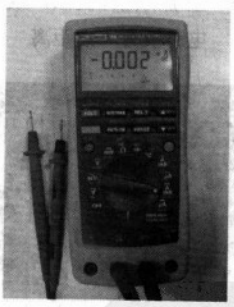


图 20-13　绝对避免在电流挡位去测量电压，这时通过的电流很大，会使内部保险丝熔断

20-3　数字电表使用时的特别注意事项

（1）仪表外壳若有破损或绝缘不良时，一定要停止测量。

（2）测量的电压不可以超过数字电表规格上的最大极限。

（3）操作交流电压 30V（有效值）、42V（峰值）或直流 60V 以上的电压时，应该小心测量，测量这类的电压可能会有被电击的危险。

（4）测量高压电容前，必须先行放电，以免损坏数字电表的输入电路。

（5）电流挡操作完毕后，应该尽速将挡位切换到其他档，以免发生错误的测量，致使内部 FUSE 熔断。

20-4 AT2051 控制板操作示范

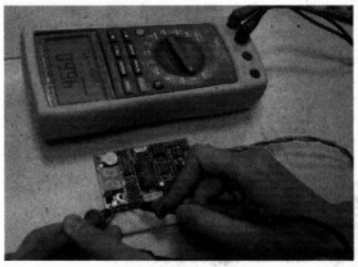


图 20-14 测量 CPU 的电源电压

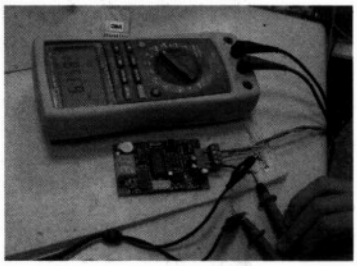


图 20-15 测量 AT2051 控制板的消耗电流，静态测量约为 6mA 上下

图 20-16 SMTl60 温度感测元件的输出周期测量，测量到的值约是 46.6%

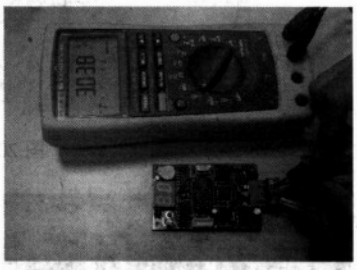


图 20-17 检查 SIO 对地的电平（不进行数据传输时）

图 20-18 检查 $\overline{\text{SIO}}$ 对地的电平（不进行数据传输时）

20-5 电表的校正

数字电表在出厂前已经用软硬件调整到最佳的状态，有些电表除了操作手册外，还会附上一份正式的测试报告（见图 20-19），说明该电表是否符合规格。数字电表除了电量不够需

要更换电池外，另外还要定期进行测量值的校正，以便确定所有读值的准确度。

Production Verification Test Data

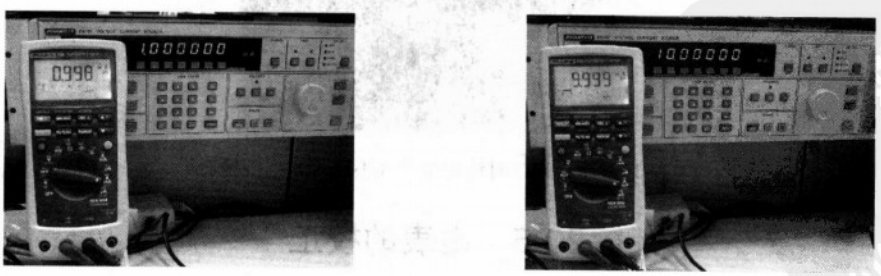
MODEL: 189 **SERIAL NUMBER: B1010059**

Function	Applied Stimulus	Response	Calibrator Uncertainty	High Limit	Low Limit
VAC	4.5 V @ 1 kHz	4.4994	8.75E-04	4.5220	4.4780
	4.5 V @ 100 kHz	4.3691	5.65E-03	4.8640	4.1360
	45 V @ 1 kHz	45.011	1.04E-03	45.220	44.780
	45 V @ 100 kHz	44.139	1.42E-01	48.640	41.360
	450 V @ 65 Hz	450.32	1.45E-01	452.20	447.80
	600 V @ 65 Hz	599.2	1.99E-01	606.4	593.6
mVAC	45 mV @ 100 kHz	0.041884	6.80E-05	0.051790	0.038210
	400 mV @ 100 kHz	0.38436	4.05E-04	0.43240	0.36760
	3000 mV @ 1 kHz	3.0001	3.85E-04	3.0160	2.9840
	3000 mV @ 100 kHz	2.9951	2.23E-03	3.2440	2.7560
VDC	0 V	-0.0001	1.00E-06	0.0010	-0.0010
	4.5 V	4.5008	6.90E-06	4.5021	4.4979
	-45 V	-45.001	9.40E-04	-44.954	-45.047
	-450 V	-449.98	9.60E-01	-449.53	-450.47
	-600 V	-600.1	1.23E-02	-599.2	-600.8
AC+DC	4 V @ 1 kHz	4.0024	8.00E-04	4.0240	3.9760
	1 V @ 20 kHz	0.9964	2.40E-04	1.0240	0.9760
mVDC	45 mV	0.044997	1.90E-05	0.048055	0.041945
	-45 mV	-0.044993	1.90E-05	-0.044935	-0.045065
	450 mV	0.45000	1.00E-05	0.48016	0.41984
Ohms	4500 Ohms	4500.0	1.90E-01	4502.5	4497.5
	45 kOhms	45002	1.96E+00	45025	44975
	450 kOhms	450000	2.14E+01	450250	449750
	4.5 MOhms	4496100	6.85E+02	4502500	4497500
	30 MOhms	2.9992E+07	1.01E+04	3.0015E7	2.9985E7
nS	10 nS	9.95E-09	5.02E-12	1.02E-8	9.80E-9
nF	80 nF	8.02E-08	3.00E-10	8.17E-8	7.83E-8
	5 uF	4.99E-06	2.25E-08	5.10E-6	4.90E-6
AAC	0.25 A @ 1 kHz	0.2500	1.20E-04	0.2550	0.2442
	6 A @ 1 kHz	5.999	6.00E-03	6.095	5.905
	2.5 mA @ 1 kHz	0.002501	2.65E-06	0.002539	0.002461
	329 mA @ 10 kHz	0.33005	7.40E-04	0.32490	0.31510
	490 uA @ 1 kHz	4.000E-04	5.50E-07	4.032E-4	3.968E-4
	1500 uA @ 1 kHz	0.0015002	1.65E-06	0.0015116	0.0014883
ADC	4 A	3.998	2.33E-03	4.021	3.979
	-6 A	-5.998	3.93E-03	-5.969	-6.032
	-400 mA	-0.39998	1.20E-04	-0.38938	-0.40062
	400 uA	4.000E-04	7.00E-08	4.012E-4	3.988E-4
	-5000 uA	-0.0049998	7.00E-07	-0.0049873	-0.0050127

图 20-19　原厂的测试报告

(a)　(b)

图 20-20　直流电压校正方法，检查 1.000V 与 10.000V 的准确度



(a)　(b)

图 20-21　直流电流校正方法，检查 10.00mA 与 1.000mA 的准确度

习　题

1. 如何利用数字电表测量交流电压？
2. 如何利用数字电表判断晶体管引脚？
3. 当测量的电压值超过何种范围时容易被电击？分直流、交流作答。
4. 测量电流值应注意哪些事项？
5. 就同一台电表来说，测量电压与测量电流的误差比较，何者误差值较大？
6. 测试棒的正确握法是什么？其正确的原因是什么？
7. 测量电子元件的电阻时，为什么不可以用手直接触碰元件引脚？
8. 接地端的测棒其颜色通常是什么？

21

旗威科技公司的分散式控制器都具备RS485接口，如果与PC连接就要有USB转485的转换电路板

第 21 章　USB 烧录器的安装与使用

本章我们要实际操作 AT89CX051 USB Programmer，经过这一章的训练，相信可以让读者在操作 USB 烧录器时有“如鱼得水、如虎添翼”般流畅、顺手，缩短您设计时耗费在等待烧录的时间。欲知详情，请进入本章的学习。

21-1　旗威 USB 烧录器

AT89CX051 USB Programmer 是由旗威科技公司（Chipware Systems Inc）专为 Atmel 的 AT89C 1051U/2051/4051 单片机微控制器所特别设计的 USB 接口烧录器，除了可凭借 PC 的联机操作之外，还具备单机操作的功能。不仅如此，为了方便使用者对烧录器的简易自我故障排除，在烧录程序中还附有硬件诊断（Diagnostic）功能，是目前市面上规格相近的烧录器中唯一具备 USB 接口且功能最齐全的一款烧录器，更重要的是：它还具备电流过载保护，如果不小心将芯片插反了，烧录器会立即断电，以免芯片烧毁，同时也保护烧录器的电路安全。

21-2　烧录器的安装

1. 系统需求

由于 USB 接口必须在 Windows 下运行，因此操作系统需求为 Microsoft Windows 98/ME/2000/XP 等版本，因为烧录器是在 Microsoft Windows 系列中开发出来的，所以并不支持 DOS/Linux/MacOS 等操作系统。

2. 驱动程序安装

（1）Windows 98/ME。如果你所使用的操作系统是 Windows 98/ME，建议安装的方式是先将烧录器接上 PC 机，这时烧录器上的 LED 会亮起来，并发出哔哔声，然后将鼠标移到“我的电脑”，单击鼠标右键，然后选择“属性”命令，此时会打开“系统属性”对话框。

选择“设备管理器”并单击下方“刷新”按钮，这时画面会出现“添加新硬件向导”对话框，单击“下一步”按钮。

此时画面上会出现“从磁盘安装”对话框，接着单击“浏览”将安装路径指定到驱动程序中的 Windows 98/ME 文件夹，并单击“下一步”按钮，接下再单击“完成”按钮。

（2）Windows 2000/XP。原则上 Windows 2000/XP 具备 USB 设备的即插即用功能，因此当你将烧录器接上 PC 机后，系统就会自动检测并跳出“添加硬件向导”对话框，接着按照在 Windows 98/ME 下的安装步骤依次安装，差别在必须指定路径到 Windows 2000/XP 的文件夹，如此便算大功告成。

如果你在安装的过程中，系统并没有如期地自动搜索硬件时，你可以经由系统的“控制面板”，找到“添加/删除硬件”，并以强制指定的方式将驱动程序从程序磁盘安装，这种方式

一样可以将硬件顺利安装。

3. 常见问题

如果你在安装过程稍微注意一下，会发现烧录器上有 24 pins，但是 AT89CX051 系列单片机只有 20 pins，多出来的 4 pins 是要做什么用的呢？其实，多出来的 4 pins 都是空引脚，我们可以用来作为存放 IC 的指针用，这个功能在稍后烧录功能说明部分中会详细提到。

21-3 烧录程序的安装

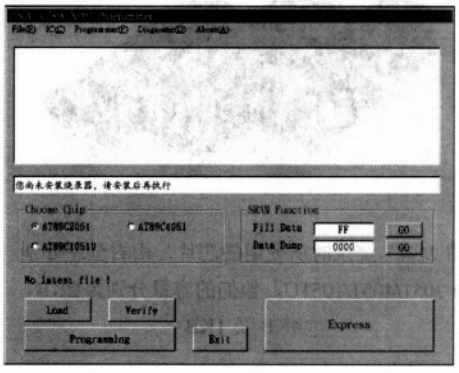
1. 程序界面简介

本书所介绍的烧录程序为 AT89CX051 Programmer V1.0，除了基本的烧录功能外，还包含了烧录器寄存区（SRAM）的管理以及硬件诊断（Diagnostic）功能，虽然没有绚丽的操作界面，但平实且齐全的功能是众多烧录程序所无法媲美的。

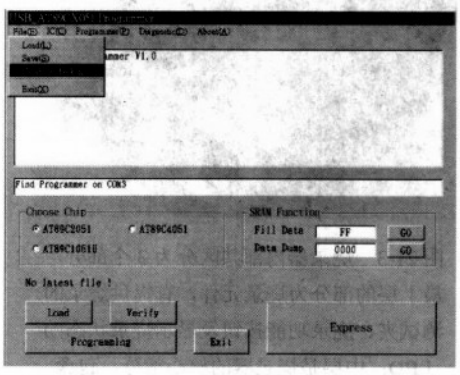
2. 烧录程序安装

烧录程序的安装相当简单，首先将程序光盘中的 setup. exe 执行文件执行，再选择所要安装的文件夹位置，依照安装向导的指示一直单击“下一步”按钮就完成了。

安装完成后执行烧录程序 USB_PROG . EXE，程序会主动寻找烧录器所安装的位置，如果没有找到，请确认 USB 的插座是否接触不良，将 USB 插座拔起再接上，执行烧录程序中[File1]=>[ComPort check]命令，找到之后便可开始使用了。



（a）烧录程序当未找到安装程序时，窗口会出现“您尚未安装烧录器”



（b）ComPort check 菜单画面

图 21-1 烧录程序安装示意图

21-4 烧录功能说明

在提到烧录之前，还是要提醒一下 IC 的摆放方式，AT89CX051 系列芯片都是 20 pins 的，可是烧录器上却有 24 pins，如果摆放错误是无法进行烧录的。正确的摆放方式是将 IC 向下对齐（见图 21-1），上面留 4 pins 是空出来的，而 IC 有凹陷的位置是向上朝着空 pins 的方向，这样的设计可以确保你的 IC 不会反插造成短路。虽说烧录器本身有电流过载的保护，但经常

性的 IC 反插会使烧录器的寿命减短，甚至造成过载保护的失效。

这里顺便提供一个口诀帮助你记忆：“开口朝上，向下对齐”，这样就不会记错了！

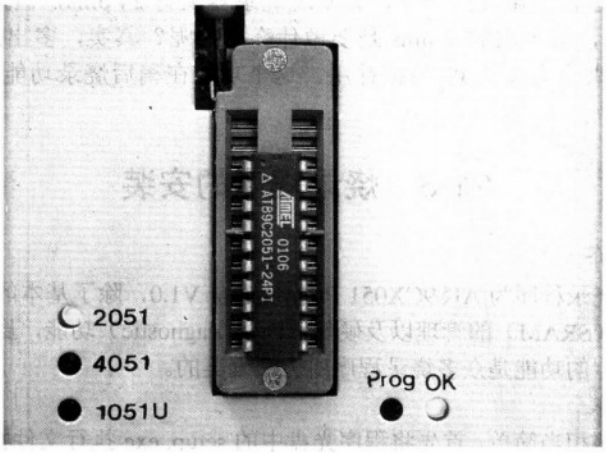


图 21-2　烧录器摆放方式

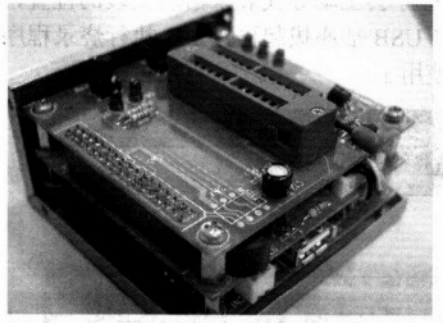


图 21-3　烧录器内部共区分为 3 个部分，最上层的部分为烧录元件，有提供烧录的测试夹、烧录功能按钮及显示烧录状态的 LED；中层是烧录器的核心部分，包含烧录的控制元件及烧录程序，还有提供声音信息的蜂鸣器；最下层是 USB 的通信接口，也是烧录电压的主要来源

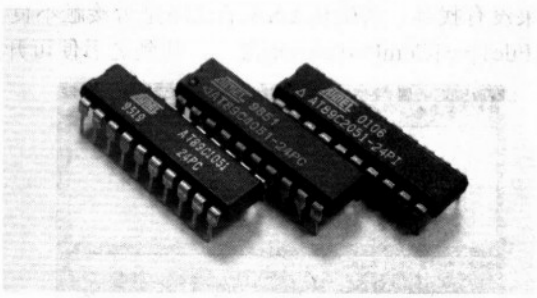


图 21-4　烧录时所使用的芯片，由右至左分别为 2051/4051/1051U，他们的容量分别为 2KB、4KB 及 1KB

1. Files（文件菜单）

包含 Load、Save、ComPort check 和 Exit 功能。

2. IC（芯片菜单）

包含 Signature、Blank Check、Erase Chip、Programming、Lock bit、Read、Verify 和 Express 功能。

3. Programmer（烧录器菜单）

包含 SRAM Clear、SRAM Fill、SRAM Dump 和 SRAM Checksum 功能。

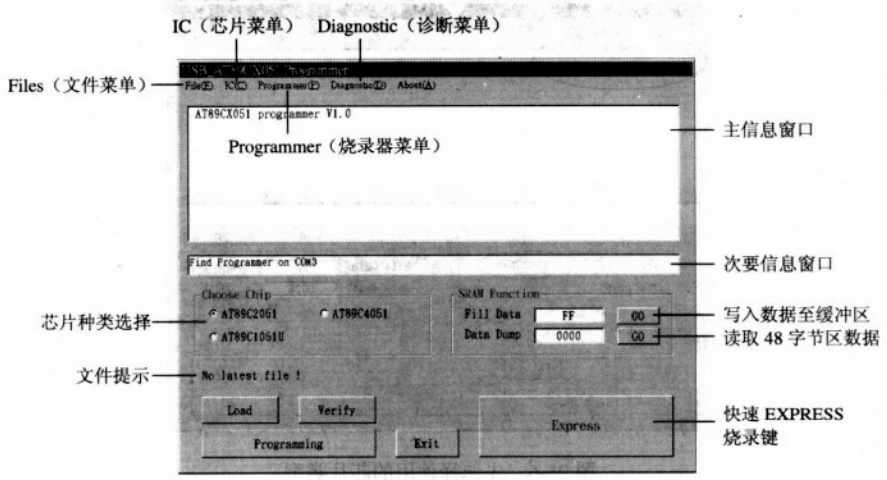


图 21-5　USB 烧录程序的启动图

4. Diagnostic（诊断菜单）

包含 Test 8K SRAM、Test LED、Test Buzzer、Test Vpp at 12V、Test Vpp 和 Firmware Version 功能。

5. 芯片种类选择

用来切换烧录器所要烧录的 2051/4051/1051U 等 3 种芯片。

6. 文件提示

这里会显示你最后一次下载的文件名称及其路径，其功能在后面的章节有详细的介绍。

7. 主信息窗口

用来显示烧录器操作的信息窗口。

8. 次要信息窗口

用来显示错误、警示或提醒信息。

9. 写入数据至缓冲区

在此输入两位数十六进制码，可将烧录器中 SRAM 的内容改成你所输入的数值。

10. 读取 48 字节区数据

在此输入 4 位十六进制码的起始地址，可查看 SRAM 中以此地址开始计算 48 字节的内容。

11. EXPRESS 快速烧录键

本程序最常用的功能按钮，也是协助你完成芯片设计所不可或缺的快速按钮。

21-5　Files 文件菜单

1. Load（载入）

Load 的功能，是将写好的程序载入烧录器的缓冲区中，载入的格式为二进制数据文件（扩展名为.tsk），载入之前请先确认你所要烧录的芯片是 2051、4051 还是 1051U，选择后再执行载入。

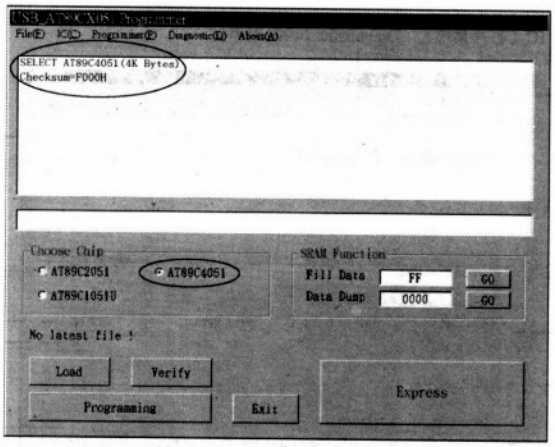


图 21-6　先选择使用的芯片类型

执行 Load 时，屏幕上会出现“打开文件”对话框，此时点选你所要载入的 TSK 文件，并单击“打开”按钮即可开始载入，等到次要信息窗口出现 Load completed 的字样时，完成载入。

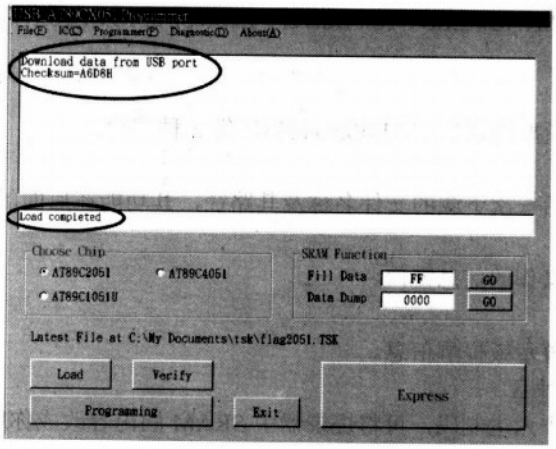


图 21-7　Load 成功后出现的画面

2. Save（保存）

Save 的功能，是要将烧录器缓冲区中的数据存回 PC 机，保存的格式是二进制数据文件（扩展名为.tsk），执行 Save 时，屏幕上会出现“保存文件”对话框，此时选择你所要保存的位置，并输入文件名，单击“保存”按钮，即开始保存，等到次信息窗口出现 Save completed 的字样时，即保存完成。

3. ComPort check（连接确认）

ComPort check 是个比较特殊的功能，当你的系统因不明原因，导致 PC 机与烧录器的连接中断时，或是因为 USB 插座未接牢，使得烧录程序搜索不到烧录器时，你都可以执行这项指令使 PC 机和烧录器重新获得连接。

21-6 IC 芯片菜单

1. Signature（芯片信息）

每一颗 IC 出厂的时候都会有一个出厂编号，这个编号是用来告诉我们它是哪一家厂商所制造的，这项功能除了可以验明 IC 真伪以外，也可以用来判断你所买到的 IC 是新品，还是 Remark 或仿冒的 IC。因为芯片经过多次的烧录或是锁码、解码过程，会不小心将这个出厂编号破坏掉。

2. Blank Check（空白芯片确认）

任何一颗芯片在出厂时应该都是空白的，也就是芯片里所有地址的数据都是 FFH，这个功能是为了确保你所烧录好的芯片不会不小心再被洗掉；或是用来确认芯片里面已经没有数据存在，进行烧录时不会因残留未清除的数据导致控制元件的运行不正常。不过，如果你的芯片是经过锁码的（Lock bit），那么就算里面有满满的数据，检查后得到的果还是空白 FFH 的。

在 Blank Check（空白确认）进行时，主信息窗口会出现“B”的字母，每出现一个“B”代表检查完 1KB 的地址是空白的，如果检查的结果这颗芯片是空白的，那么主信息窗口会显示下列信息：

“This chip is actually BLANK”

如果检查到不是空白的地址，则主信息窗口会显示：

“This chip is NOT blank at [0010H] 03H”

这代表程序在 0010H 的地址上检查到第一个不是空白的数据，而这个非空白的数据是 03H。

3. Erase Chip（清除芯片数据）

这个功能就是把芯片里所有的数据强制性地填成空白（FFH），如果你手边有一颗不确定是否为锁码或是空白的芯片，这个功能绝对可以帮你做确认。当 Erase Chip 的操作结束后，烧录程序会立刻对芯片进行空白确认，如果你在主信息窗口所得到的信息并不是空白芯片的话，那么请利用 Diagnostic 菜单中的 Test Vpp at 12V 来检查烧录器的烧录电压是否不正常。

4. Programming（烧录芯片）

这个功能包含了 4 个操作 Erase（清除）、Blank Check（空白确认）、Program（烧录）、Verify（数据验证），这个过程也是烧录芯片必备的 4 项操作，如果烧录过程中出现任何错误的信息，一定只有以下两种情况：

（1）非空白芯片（The chip is NOT blank）代表烧录器的烧录电压不足，或是芯片损毁。

（2）数据验证错误（Verify FAIL）代表芯片本身是锁码芯片，在 Erase 过程中因为烧录电压不足，导致无法清除锁码的数据，到了数据验证时便发生错误信息；另一个可能是损毁的芯片本身就是空白的，但是损毁的地址无法将数据填入，造成验证的错误。

5. Lock bit（锁码）

辛苦设计的数据，如果不加以保护，那么别人只要两三秒就把自己的心血都复制走了，当芯片 Lock bit 之后，所有的内容都无法读取，也无法将新程序写在旧程序的后段，而 Blank check 则会认为这是一颗空白的芯片，与旧数据验证时也会发生错误，读取芯片数据时所得到的数据全部都是 FFH。当 Lock bit 的操作完成时，主信息窗口会出现下列信息：

“Program and Verify are Disabled”

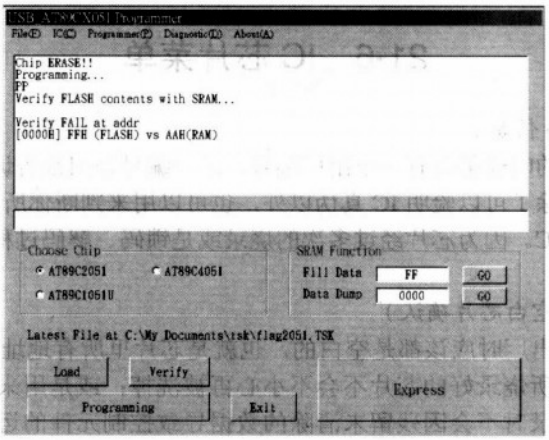


图 21-8　执行 Programming 后的画面

这就是告诉你芯片已经被锁码，任何人不能看到里面的数据。

6. Read（读取芯片数据）

设计中的芯片操作得不太正常时，要怎么确认芯片里哪一段地址的数据出了问题呢？对了！就是把它读取出来，这个操作会把芯片的数据存到烧录器的缓冲区中，以便做数据的确认与修改。不过先决条件就是不可以锁码。

同 Blank check 进行时一般，Read 功能在进行时，主信息窗口会出现“R”的字母，每出现一个“R”代表读取完 1KB 地址的数据，如果你选用的芯片是 2051，那么主信息窗口会显示两个“R”，然后出现一个 Checksum 值，这个值在稍后的章节里会有详细的介绍。

7. Verify（数据验证）

要怎么确定我所烧录的数据是正确无误的呢？这个功能可以帮上你一个很大的忙！他会把芯片中每一个地址的数据和烧录器缓冲区中的数据进行验证，如果发现不同的地方会立刻显示在信息窗口中。

在 Verify 进行时，主信息窗口也会出现“V”的字母，每验证 1KB 地址的数据是正确无误的话，就会出现一个“V”，如果验证的数据无误，主信息窗口会显示：

“Verify OK”

如果验证数据有误，则主信息窗口会显示：

“Verify FAIL at addr

[0000H]03H（FLASH）vs 13H（RAM）”

这代表在芯片中 0000H 的地址上有一笔 03H 数据，和烧录器上的 13H 数据验证不符。

8. Express（快速烧录）

假如你觉得芯片从 Load、Erase、Blank check 到 Program、Verify、Lock bit 的时间实在是太长了，那么这个功能保证让您满意！它把上述所有的操作简化成一个单一的操作，只要你确定你所写的程序是正确无误的，那么只要几秒钟时间就可以完成一个芯片的烧录。

在此要特别提到文件提示的功能，在图 21-7 中所圈选的部分，是提示你进行 Express 时所会下载的文件，也是你最后一次下载的文件，如果你所要烧录的程序并不是提示中的文件，请先用 Load 功能下载。

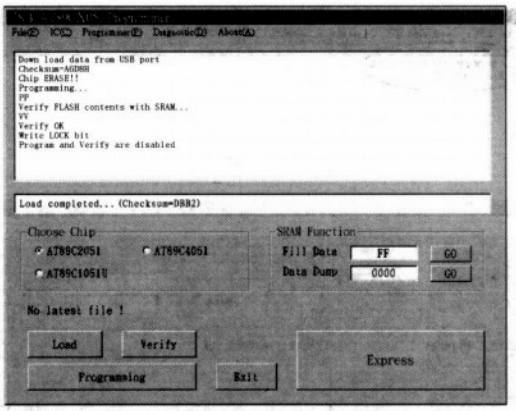


图 21-9　Express 功能特别说明，按 Express 后，烧录程序会将文件提示中最近一次 Load 的文件下载到烧录器的 SRAM 中，等运行成功后，会出现 Load Completed 的信息，接着经过 Erase、Blank check、Program、Verify、Lock bit 的过程，芯片就烧录完成了，只要按一个键即把烧录的事情搞定

21-7　Programmer 烧录器菜单

1. SRAM Clear（清除缓冲区数据）

当你想要读取芯片的数据时，别忘了先把缓冲区的数据清除掉！以免将来保存了不正确的芯片数据在文件里。

2. SRAM Fill（写入数据至缓冲区）

当你想测试一个新的芯片是否可以正常烧录时，这是一个最快速的确认方法：先在 SRAM Function 的 Fill Data 文字框中写入两位数十六进制码，再单击 GO 按钮便可将 SRAM 的内容全都写成你所输入的数值，再把数据烧进芯片里，假如 Verify 正确的话，那就成功了！如果你直接点选菜单中的 SRAM Fill，那么 SRAM 则会填入目前在 SRAM Function 的 Fill Data 文字框中所设置的数值，默认值为 FFH。

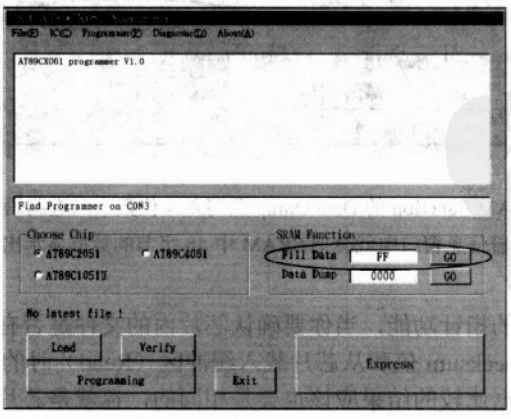


图 21-10　Fill Data 时要先输入两位数十六进制码，然后再单击 GO 按钮

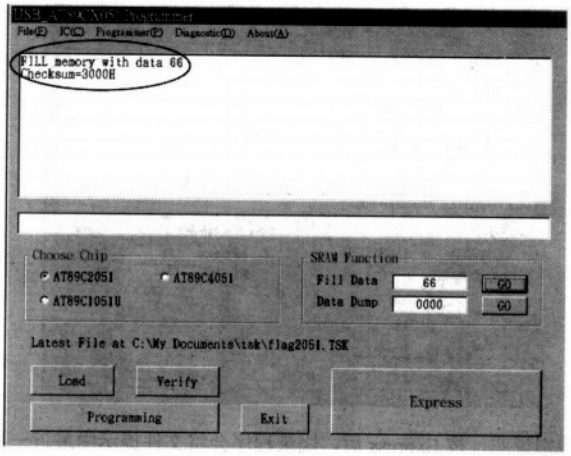


图 21-11　SRAM Fill 后就会全部变成你要的数值

3. SRAM Dump（读取缓冲区数据）

想知道缓冲区里的数据到底放了些什么吗？别急，在 SRAM Function 的 Data Dump 文字框中写入 4 位数十六进制码，这串数字便是你所想知道的缓冲区的起始地址，单击“GO”按钮，就会把数据显示在信息窗口中，不过只能显示 48 字节的数据。

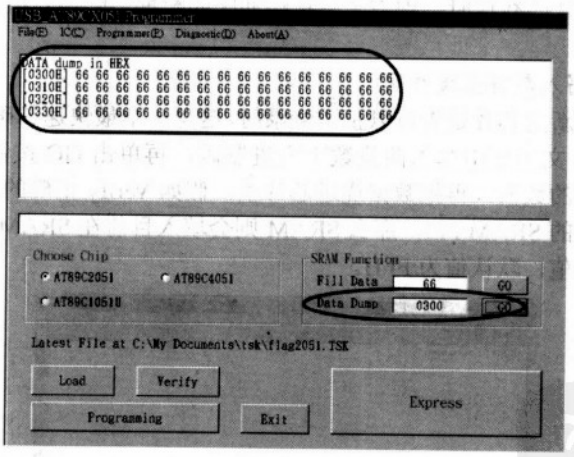


图 21-12　SRAM Function 的 Data Dump 要写入 4 位数十六进制码的起始地址，
接着信息窗口中就会把 SRAM 中 48 字节的数据显示出来

4. SRAM Checksum

这是一个相当重要的指针功能，当你要确认芯片内的文件是否有修改时，只要把文件载入缓冲区（Load）的 Checksum 值和从芯片载入缓冲区（Read）时的 Checksum 值比较一下，如果数值相同，那么数据比较的结果应该也会是相同的，也就是芯片的内容已经和程序本身同步更新了；如果 Checksum 值不同，那么芯片内的数据应该是修改前的数据。有了这项功

能，便可省略 Verify 所要花费的时间。

21-8 Diagnostic 诊断菜单

1. Test 8K SRAM（测试缓冲区）

这个选项会在缓冲区里写上 01、02、…、FE、FF 的数据，经过 Data Dump 确认无误，代表缓冲区正常。

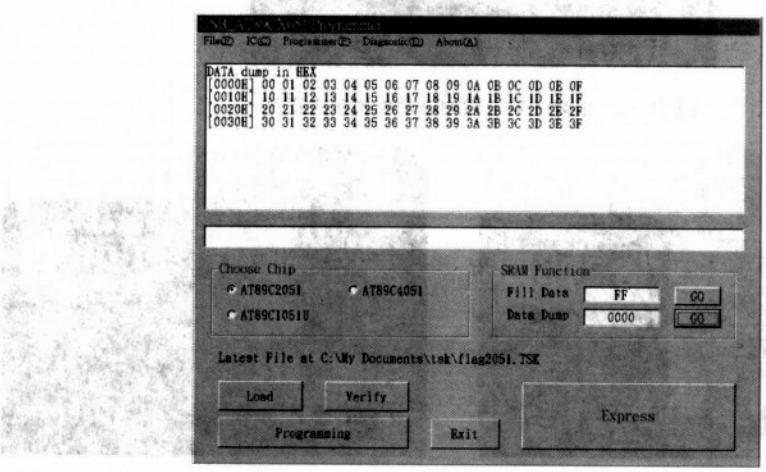


图 21-13　Test 8K SRAM 后，再以 Data Dump 查看 SRAM 数据的画面

2. Test LED（测试 LED）

当你使用这个功能时，你会发现烧录器上的 LED 闪烁着。如果你的 LED 没有任何的反应，那么你的烧录器应该是有故障了。

3. Test Buzzer（测试蜂鸣器）

启动这项功能时，蜂鸣器会发出一长音。

以下关于烧录电压的一些测试，请先把芯片从烧录器上取下，以免不小心烧毁

4. Test Vpp at 12V（测试烧录电压）

要测试电压必须先准备一个测量直流电压的电表，将负端（黑线）放在测试夹的第 12 引脚，正端（红线）放在第 3 引脚，并将电表选择在直流电压（DCV）的测量（如果选错了可能会造成电表的保险丝烧掉），再单击此功能，如果电表的数值约略在 11.5～12.5V 之间，那么你的烧录电压是正常的。选择此项功能会提供持续约 7～8s 的烧录电压，随后会立即降回一般电位。

5. Test Vpp at High（测试逻辑高电位）

USB 接口所提供的标准电压是 5V，允许的工作电压在 4～5.5V 之间，如果你所测量到的电压不足 4V，那么你的 USB 设备可能出了点问题，或者 PC 机的电源供应器应该要检查看看了。

6. Test Vpp at Low（测试逻辑低电位）

同上面的电压测量，测得的电位应该是小于 0.1V 的。

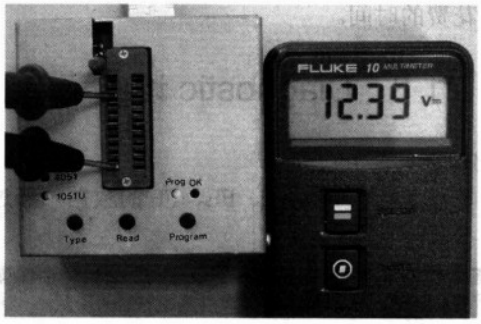


图 21-14　用数字电表量测烧录电压是否正常

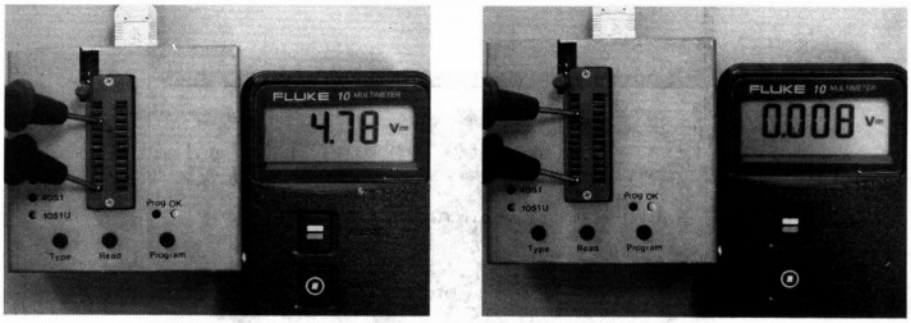


图 21-15　用电表测量烧录器所提供的高低电压是否正常

7. Firmware Version（诊断程序版本）

显示烧录器的软件版本。

8. Version（烧录程序版本）

USB 烧录软件程序的版本。

21-9　USB 烧录器特殊用法

1. 单机操作

如果你只是单纯地想复制芯片数据，觉得启动烧录程序实在是很麻烦，这里教你一个小秘诀：如果你的 PC 机支持远程开机的功能，那么你只要将烧录器接上 PC 机的 USB 插座，便可实施单机操作。烧录器上提供了 3 个按键，分别为 Type、Read、Program，其功能分别为“选择芯片种类”、“读取芯片数据”、“烧录芯片”，上面所提供的 LED 是显示烧录的状态，如果烧录失败，LED 会以闪烁的方式警告，成功则显示 OK！

2. 分段烧录

假如我有一颗 4051 的芯片，可是我要修改的数据只有最前段不到 1KB，我要怎么缩短烧录时间呢？其实烧录器所提供的芯片选择是以数据量的大小来做决定的，如果你所修改的部分不到 1KB，且确定其地址在前段，那么你只要选择 1051U 的芯片进行烧录，便可大幅缩短烧录时间（烧录 4KB 跟烧录 1KB 的时间绝对是有明显的差别的）。

21-10 USB 烧录器注意事项

1. 不要随意拆卸烧录器

烧录器本身具备绝缘保护及电流过载保护，如果你对烧录器本身的运作原理还在摸索的阶段中，建议不要轻易将烧录器自行拆卸，否则容易因不当的操作导致 USB 电压短路，轻则 PC 机不正常关机，重则导致硬盘、光驱的毁坏，请使用者注意。

2. 不要连续单击不同的功能

在烧录程序的操作尚未完成前，请不要急着把下一个操作启动，不然可能会造成数据传输的错误，导致烧录失败或烧录的数据不正确。

3. 不正常连接或连接中断时请先启动连接

如果烧录器并未连接，请不要执行任何的操作，先以 ComPort check 将烧录器重新连接再开始使用，或是单击 Exit 按钮，退出烧录程序，以免造成程序的错误。

4. 请勿打开烧录程序后同时操作单机按键

由于烧录器本身并不具备检测信号来源的功能，因此不论你使用烧录程序或是单击按键，烧录器都会视为同一个指令来源，如果同时进行的话，会造成烧录程序的操作误判，产生不可预期的错误信息，虽不会影响烧录的进行，但还是尽量避免，所有操作的正确性以烧录器上显示的灯号为主要参考依据。

5. 请注意烧录超过百次以上的芯片

有些 AT89C2051 或 AT89C4051 经过上百次接近千次的烧录后，验证数据时是正确的，但是实际执行却有问题，亦即接上电源后操作完全出不来，这时就要更换一颗新的芯片了，因为该 IC 算是“阵亡”了。

习　题

1）在本章所提的烧录程序中，其文件菜单包含哪些功能？
2）承上题，IC 菜单中包含哪些功能？
3）AT89C2051 Lock bit 的功能是什么？
4）Express 包含了哪些操作？
5）在烧录器菜单中的 SRAM Fill 功能该如何操作？
6）承上题，SRAM Dump 功能该如何操作？
7）在测试烧录电压前，应先确认的操作是什么？
8）在使用 USB 烧录器时，有哪些操作应尽量避免？

附　录

本书的附录多达四十多页，这些都是 8051 单片机学习之余，你要经常参考与查阅的主题，这些资料经过我们仔细地筛选与编排，内容绝对是宝贵的，我们希望本附录成为你写程序时最佳的参考依据之一。

附录 A　ASCII 表

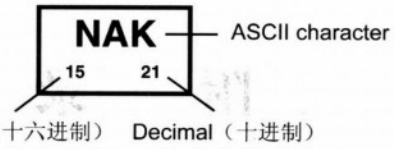
	0	1	2	3	4	5	6	7
0	NUL 00 0	DEL 10 16	SP 2D 32	0 30 48	@ 40 64	P 50 80	' 60 96	p 70 112
1	SOH 01 1	DC1 11 17	! 21 33	1 31 49	A 41 65	Q 51 81	a 61 97	q 71 113
2	STX 02 2	DC2 12 18	" 22 34	2 32 50	B 42 66	R 52 82	b 62 98	r 72 114
3	ETX 03 3	DC3 13 19	# 23 35	3 33 51	C 43 67	S 53 83	c 63 99	s 73 115
4	EOT 04 4	DC4 14 20	$ 24 36	4 34 52	D 44 68	T 54 84	d 64 100	t 74 116
5	ENQ 05 5	NAK 15 21	% 25 37	5 35 53	E 45 69	U 55 85	e 65 101	u 75 117
6	ACK 06 6	SYN 16 22	& 26 38	6 36 54	F 46 70	V 56 86	f 66 102	v 76 118
7	BEL 07 7	ETB 17 23	' 27 39	7 37 55	G 47 71	W 57 87	g 67 103	w 77 119
8	BS 08 8	CAN 18 24	(28 40	8 38 56	H 48 72	X 58 88	h 68 104	x 78 120
9	HT 09 9	EM 19 25	) 29 41	9 39 57	I 49 73	Y 59 89	i 69 105	y 79 121
A	LF 0A 10	SUB 1A 26	* 2A 42	: 3A 58	J 4A 74	Z 5A 90	j 6A 106	z 80 122
B	VT 0B 11	ESC 1B 27	+ 2B 43	; 3B 59	K 4B 75	[5B 91	k 6B 107	{ 7A 123
C	FF 0C 12	FS 1C 28	, 2C 44	< 3C 60	L 4C 76	\ 5C 92	l 6C 108	| 7B 124
D	CR 0D 13	GS 1D 29	- 2D 45	= 3D 61	M 4D 77	] 5D 93	m 6D 109	} 7C 125
E	SO 0E 14	RS 1E 30	. 2E 46	> 3E 62	N 4E 78	^ 5E 94	n 6E 110	~ 7D 126
F	SI 0F 15	US 1F 31	/ 2F 47	? 3F 63	O 4F 79	_ 5F 95	o 6F 111	DEL 7F 127

高位

低位

Example

NAK	
HEX	DECIMAL



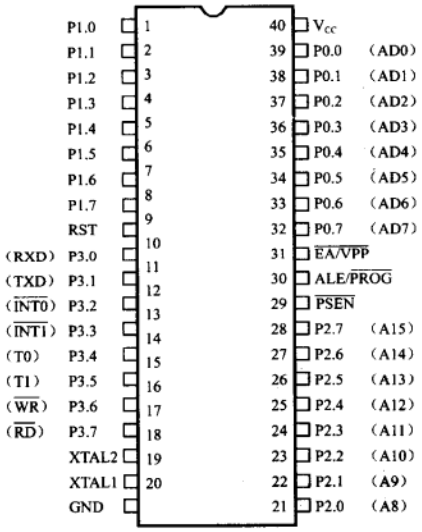
如何利用本表格

如果我们收到了 30H 这个十六进制码时，代表何种意义呢？请先查列 3，再查行 0，两条线交插刚好是 0（数字 0），这表示我们收到的是一个数字。如下表所示。许多可与电脑连接的仪器设备由于只传输数据，所以送回的值都介于 30H～39H 间，查 ASCII 表的结果，刚好就是数字 0～9。另外，如果我们送一个 45H 给 PC 时，PC 的屏幕上就会显示出 E 的字样。

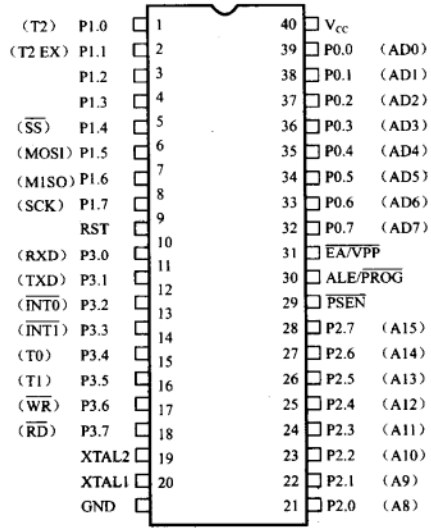
step 1

step 2	0	1	2	3	4	5	6	7
0	NUL 00 0	DEL 10 16	SP 2D 32	0 30 48	@ 40 64	P 50 80	' 60 96	p 70 112
1	SOH 01 1	DC1 11 17	! 21 33	1 31 49	A 41 65	Q 51 81	a 61 97	q 71 113
2	STX 02 2	DC2 12 18	" 22 34	2 32 50	B 42 66	R 52 82	b 62 98	r 72 114
3	ETX 03 3	DC3 13 19	# 23 35	3 33 51	C 43 67	S 53 83	c 63 99	s 73 115
4	EOT 04 4	DC4 14 20	$ 24 36	4 34 52	D 44 68	T 54 84	d 64 100	t 74 116
5	ENQ 05 5	NAK 15 21	% 25 37	5 35 53	E 45 69	U 55 85	e 65 101	u 75 117
6	ACK 06 6	SYN 16 22	& 26 38	6 36 54	F 46 70	V 56 86	f 66 102	v 76 118
7	BEL 07 7	ETB 17 23	' 27 39	7 37 55	G 47 71	W 57 87	g 67 103	w 77 119
8	BS 08 8	CAN 18 24	(28 40	8 38 56	H 48 72	X 58 88	h 68 104	x 78 120
9	HT 09 9	EM 19 25	) 29 41	9 39 57	I 49 73	Y 59 89	i 69 105	y 79 121
A	LF 0A 10	SUB 1A 26	* 2A 42	: 3A 58	J 4A 74	Z 5A 90	j 6A 106	z 80 122

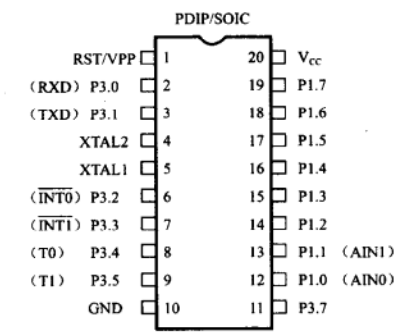
附录 B　8051 相关 IC 引脚图



8051/8031
微控制器

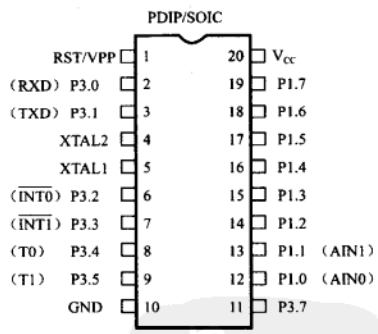


BK FLASH+2KB E²PROM
AT89S8252
微控制器



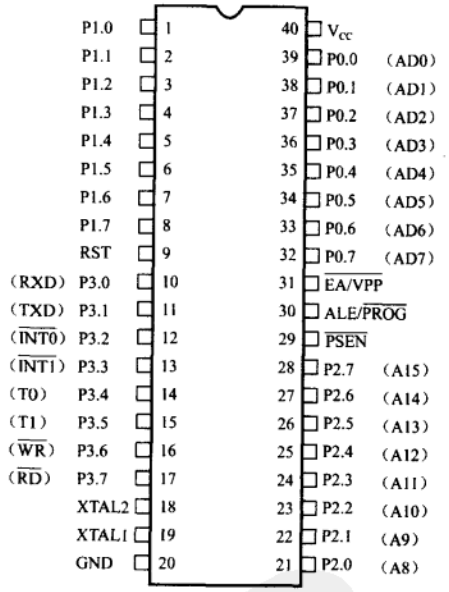
2K/4K Flash
AT89C2051/AT89C4051

AIN0 正极输入
AIN1 负极输入
P3.6 比较输出器

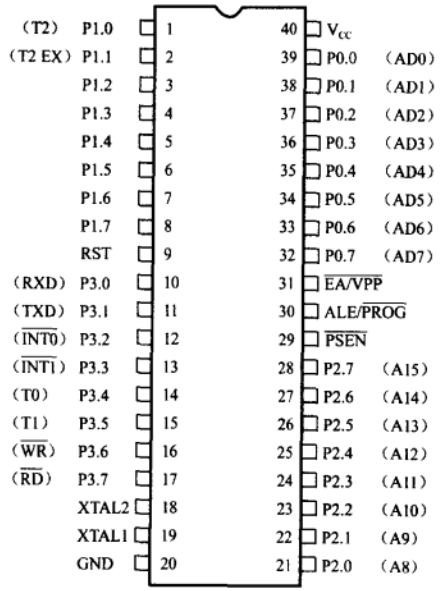


1K FLASH
AT89C1051U

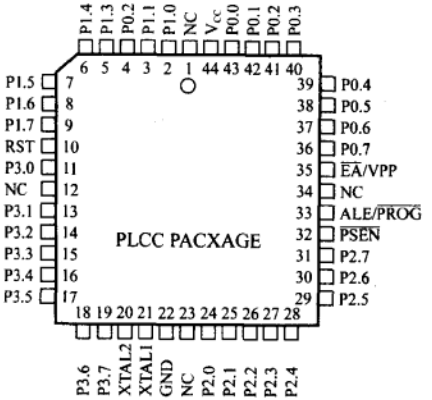
AIN0 正极输入
AIN1 负极输入
P3.6 比较输出器

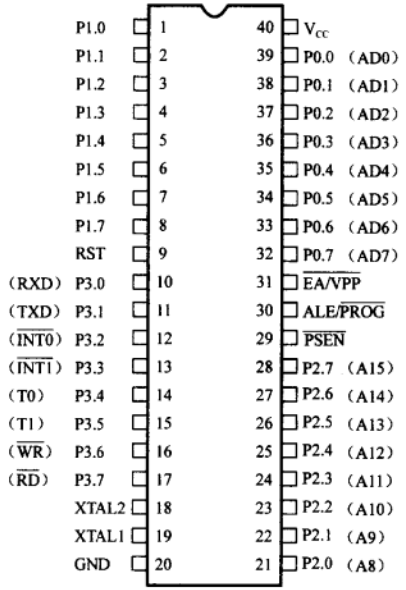


4K FLASH
AT89C51
微控制器

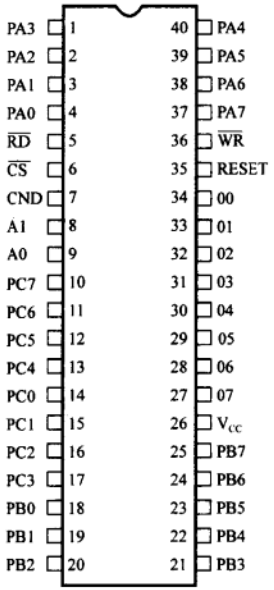


8K FLASH
AT89C52
微控制器



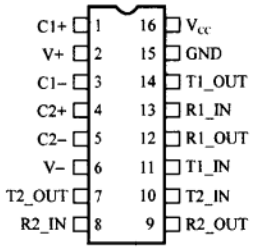


8051/8031
微控制器

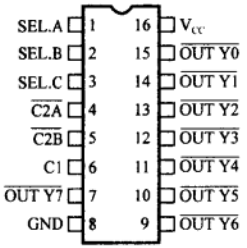


8255PP1

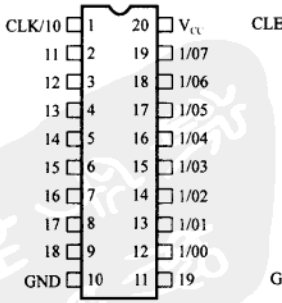
K817/TLP521
OPTO ISOLATOR



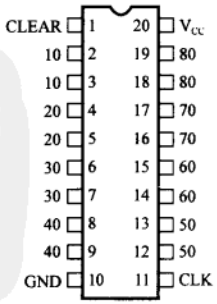
MAX232
RS-232 TRANSCEVER



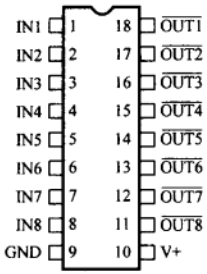
74LS138
3 TO 8 DECODER



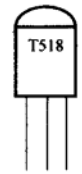
PEEL18CV8



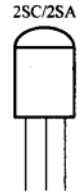
74LS273
OCTAL D-FF WITH
CLEAR INPUT



ULM2003
DARLINGTON TRANSISTER ARRAY
(OUTPUT INVERTED)



RESET IC



SMALL SIGNAL TRANSISTER

附录 C　8051 指令集总整理

影响标志位的指令整理

Instruction	Flag		
	Carry	Overflow	Aux Carry
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
NUL	0	X	
DIV	0	X	
DAA	X		
RRC	X		
RLC	X		
SETB C	1		
CLR C	0		
CPL C	X		
ANL C, bit	X		
ANL C, /bit	X		
ORL C, bit	X		
ORL C, /bit	X		
MOV C, bit	X		
CJNE	X		

注　SFR 字节地址 208 或位地址 209～215（即 PSW 或 PSW 中的位）的操作也会影响标志位设置。

指令集与寻址模式

Rn	Register R7~R0 of the currently selected Register Bank.
direct	8-bit internal data location's address. This could be an Internal Data RAM location (0~127) or a SFR[i.e., I/O port, control register, status register, etc. (128~255)].
@Ri	8-bit internal data RAM location (0~255) addressed indirectly through register R1 or R0.
#data	8-bit constant included in instruction.
#data 16	16-bit constant included in instruction.
addr 16	16-bit destination address. Used by LCALL and LJMP. A branch can be anywherre within the 64K byte Program Memory address space.
addr 11	11-bit destination address. Used by ACALL and AJMP. THe branch will be within the same 2K byte page of program memory as the first byte of the following instruction.
rel	Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.
bit	Direct Addressed bit in Internal Data RAM or Special Function Register.

8051 指令集汇总

	0	1	2	3	4	5	6	7
0	NOP	JBC bit, rel [3B, 2C]	JB bit, rel [3B, 2C]	JNB bit, rel [3B, 2C]	JC rel [2B, 2C]	JNC rel [2B, 2C]	JZ rel [2B, 2C]	JNZ rel [2B, 2C]
1	AJMP (P0) [2B, 2C]	ACALL (P0) [2B, 2C]	AJMP (P1) [2B, 2C]	ACALL (P1) [2B, 2C]	AJMP (P2) [2B, 2C]	ACALL (P2) [2B, 2C]	AJMP (P3) [2B, 2C]	ACALL (P3) [2B, 2C]
2	LJMP addr16 [3B, 2C]	LCALL addr16 [3B, 2C]	RET [2C]	RETI [2C]	ORL dir, A [2B]	ANL dir, A [2B]	XRL dir, A [2B]	ORL C, bit [2B, 2C]
3	RR A	RRC A	RL A	RLC A	ORL dir, #data [3B, 2C]	ANL dir, #data [3B, 2C]	XRL dir, #data [3B, 2C]	JMP @A + DPTR [2C]
4	INC A	DEC A	ADD A, #data [2B]	ADDC A, #data [2B]	ORL A, #data [2B]	ANL A, #data [2B]	XRL A, #data [2B]	MOV A, #data [2B]
5	INC dir [2B]	DEC dir [2B]	ADD A, dir [2B]	ADDC A, dir [2B]	ORL A, dir [2B]	ANL A, dir [2B]	XRL A, dir [2B]	MOV dir, #data [3B, 2C]
6	INC @R0	DEC @R0	ADD A, @R0	ADDC A, @R0	ORL A, @R0	ANL A, @R0	XRL A, @R0	MOV @R0, @data [2B]
7	INC @R1	DEC @R1	ADD A, @R1	ADDC A, @R1	ORL A, @R1	ANL A, @R1	XRL A, @R1	MOV @R1, #data [2B]
8	INC R0	DEC R0	ADD A, R0	ADDC A, R0	ORL A, R0	ANL A, R0	XRL A, R0	MOV R0, #data [2B]
9	INC R1	DEC R1	ADD A, R1	ADDC A, R1	ORL A, R1	ANL A, R1	XRL A, R1	MOV R1, #data [2B]
A	INC R2	DEC R2	ADD A, R2	ADDC A, R2	ORL A, R2	ANL A, R2	XRL A, R2	MOV R2, #data [2B]
B	INC R3	DEC R3	ADD A, R3	ADDC A, R3	ORL A, R3	ANL A, R3	XRL A, R3	MOV R3, #data [2B]
C	INC R4	DEC R4	ADD A, R4	ADDC A, R4	ORL A, R4	ANL A, R4	XRL A, R4	MOV R4, #data [2B]
D	INC R5	DEC R5	ADD A, R5	ADDC A, R5	ORL A, R5	ANL A, R5	XRL A, R5	MOV R5, #data [2B]
E	INC R6	DEC R6	ADD A, R6	ADDC A, R6	ORL A, R6	ANL A, R6	XRL A, R6	MOV R6, #data [2B]
F	INC R7	DEC R7	ADD A, R7	ADDC A, R7	ORL A, R7	ANL A, R7	XRL A, R7	MOV R7, #data [2B]

注 [2B]=2 Byte, [3B]=3Byte, [2C]=2Cycle, [4C]=4 Cycle, Blank=1 byte/1 cycle。

续表

	8	9	A	B	C	D	E	F
0	SJMP REL [2B, 2C]	MOV DPTR, #data 16 [3B, 2C]	ORL C, /bit [2B, 2C]	ANL C, /bit [2B, 2C]	PUSH dir [2B, 2C]	POP dir [2B, 2C]	MOVX A, @DPTR [2C]	MOVX @DPTR, A [2C]
1	AJMP (P4) [2B, 2C]	ACALL (P4) [2B, 2C]	AJMP (P5) [2B, 2C]	ACALL (P5) [2B, 2C]	AJMP (P6) [2B, 2C]	ACALL (P6) [2B, 2C]	AJMP (P7) [2B, 2C]	ACALL (P7) [2B, 2C]
2	ANL C, bit [2B, 2C]	MOV bit, C [2B, 2C]	MOV C, bit [2B]	CPL bit [2B]	CLR bit [2B]	SETB bit [2B]	MOVX A, @R0 [2C]	MOVX @R0, A [2C]
3	MOVC A, @A + PC [2C]	MOVC A, @A + DPTR [2C]	INC DPTR [2C]	CPL C	CLR C	SETB C	MOVX A, @RI [2C]	MOVX @RI, A [2C]
4	DIV AB [2B, 4C]	SUBB A, #data [2B]	MUL AB [4C]	CJNE A, #data, rel [3B, 2C]	SWAP A	DA A	CLR A	CPL A
5	MOV dir, dir [3B, 2C]	SUBB A, dir [2B]		CJNE A, dir, rel [3B, 2C]	XCH A, dir [2B]	DJNZ dir, rel [3B, 2C]	MOV A, dir [2B]	MOV dir, A [2B]
6	MOV dir, @R0 [2B, 2C]	SUBB A, @R0	MOV @R0, dir [2B, 2C]	CJNE @R0, #data, rel [3B, 2C]	XCH A, @R0	XCHD A, @R0	MOV A, @R0	MOV @R0, A
7	MOV dir, @R1 [2B, 2C]	SUBB A, @R1	MOV @R1, dir [2B, 2C]	CJNE @R1, #data, rel [3B, 2C]	XCH A, @R1	XCHD A, @R1	MOV A, @R1	MOV @R1, A
8	MOV dir, R0 [2B, 2C]	SUBB A, R0	MOV R0, dir [2B, 2C]	CJNE R0, #data, rel [3B, 2C]	XCH A, R0	DJNZ R0, rel [2B, 2C]	MOV A, R0	MOV R0, A
9	MOV dir, R1 [2B, 2C]	SUBB A, R1	MOV R1, dir [2B, 2C]	CJNE R1, #data, rel [3B, 2C]	XCH A, R1	DJNZ R1, rel [2B, 2C]	MOV A, R1	MOV R1, A
A	MOV dir, R2 [2B, 2C]	SUBB A, R2	MOV R2, dir [2B, 2C]	CJNE R2, #data, rel [3B, 2C]	XCH A, R2	DJNZ R2, rel [2B, 2C]	MOV A, R2	MOV R2, A
B	MOV dir, R3 [2B, 2C]	SUBB A, R3	MOV R3, dir [2B, 2C]	CJNE R3, #data, rel [3B, 2C]	XCH A, R3	DJNZ R3, rel [2B, 2C]	MOV A, R3	MOV R3, A
C	MOV dir, R4 [2B, 2C]	SUBB A, R4	MOV R4, dir [2B, 2C]	CJNE R4, #data, rel [3B, 2C]	XCH A, R4	DJNZ R4, rel [2B, 2C]	MOV A, R4	MOV R4, A
D	MOV dir, R5 [2B, 2C]	SUBB A, R5	MOV R5, dir [2B, 2C]	CJNE R5, #data, rel [3B, 2C]	XCH A, R5	DJNZ R5, rel [2B, 2C]	MOV A, R5	MOV R5, A
E	MOV dir, R6 [2B, 2C]	SUBB A, R6	MOV R6, dir [2B, 2C]	CJNE R6, #data, rel [3B, 2C]	XCH A, R6	DJNZ R6, rel [2B, 2C]	MOV A, R6	MOV R6, A
F	MOV dir, R7 [2B, 2C]	SUBB A, R7	MOV R7, dir [2B, 2C]	CJNE R7, #data, rel [3B, 2C]	XCH A, R7	DJNZ R7, rel [2B, 2C]	MOV A, R7	MOV R7, A

注 [2B]=2 Byte, [3B]=3Byte, [2C]=2Cycle, [4C]=4 Cycle, Blank=1 byte/1 cycle。

8051 指令集——单字节指令整理 1

	0	1	2	3	4	5	6	7
0	NOP							
1								
2			RET [2C]	RET [2C]				
3	RR A	RRC A	RL A	RLC A				JMP @A + DPTR [2C]
4	INC A	DEC A						
5								
6	INC @R0	DEC @R0	ADD A, @R0	ADDC A, @R0	ORL A, @R0	ANL A, @R0	XRL A, @R0	
7	INC @R1	DEC @R1	ADD A, @R1	ADDC A, @R1	ORL A, @R1	ANL A, @R1	XRL A, @R1	
8	INC R0	DEC R0	ADD A, R0	ADDC A, R0	ORL A, R0	ANL A, R0	XRL A, R0	
9	INC R1	DEC R1	ADD A, R1	ADDC A, R1	ORL A, R1	ANL A, R1	XRL A, R1	
A	INC R2	DEC R2	ADD A, R2	ADDC A, R2	ORL A, R2	ANL A, R2	XRL A, R2	
B	INC R3	DEC R3	ADD A, R3	ADDC A, R3	ORL A, R3	ANL A, R3	XRL A, R3	
C	INC R4	DEC R4	ADD A, R4	ADDC A, R4	ORL A, R4	ANL A, R4	XRL A, R4	
D	INC R5	DEC R5	ADD A, R5	ADDC A, R5	ORL A, R5	ANL A, R5	XRL A, R5	
E	INC R6	DEC R6	ADD A, R6	ADDC A, R6	ORL A, R6	ANL A, R6	XRL A, R6	
F	INC R7	DEC R7	ADD A, R7	ADDC A, R7	ORL A, R7	ANL A, R7	XRL A, R7	

注 [2B]=2 Byte, [3B]=3Byte, [2C]=2Cycle, [4C]=4 Cycle, Blank=1 byte/1 cycle。

8051 指令集——单字节指令整理 2

	8	9	A	B	C	D	E	F
0							MOVX A, @DPTR [2C]	MOVX @DPTR, A [2C]
1								
2							MOVX A, @R0 [2C]	MOVX @R0, A [2C]
3	MOVC A, @A + PC [2C]	MOVC A, @A + DPTR [2C]	INC DPTR [2C]	CPL C	CLR C	SETB C	MOVX A, @RI [2C]	MOVX @RI, A [2C]
4			MUL AB [4C]		SWAP A	DA A	CLR A	CPL A
5								
6		SUBB A, @R0			XCH A, @R0	XCHD A, @R0	MOV A, @R0	MOV @R0, A
7		SUBB A, @R1			XCH A, @R1	XCHD A, @R1	MOV A, @R1	MOV @R1, A
8		SUBB A, R0			XCH A, R0		MOV A, R0	MOV R0, A
9		SUBB A, R1			XCH A, R1		MOV A, R1	MOV R1, A
A		SUBB A, R2			XCH A, R2		MOV A, R2	MOV R2, A
B		SUBB A, R3			XCH A, R3		MOV A, R3	MOV R3, A
C		SUBB A, R4			XCH A, R4		MOV A, R4	MOV R4, A
D		SUBB A, R5			XCH A, R5		MOV A, R5	MOV R5, A
E		SUBB A, R6			XCH A, R6		MOV A, R6	MOV R6, A
F		SUBB A, R7			XCH A, R7		MOV A, R7	MOV R7, A

注 [2B]=2 Byte, [3B]=3Byte, [2C]=2Cycle, [4C]=4 Cycle, Blank=1 byte/1 cycle。

8051 指令集——双字节指令整理 1

	0	1	2	3	4	5	6	7
0					**JC** **rel** [2B, 2C]	**JNC** **rel** [2B, 2C]	**JZ** **rel** [2B, 2C]	**JNZ** **rel** [2B, 2C]
1	**AJMP** (P0) [2B, 2C]	**ACALL** (P0) [2B, 2C]	**AJMP** (P1) [2B, 2C]	**ACALL** (P1) [2B, 2C]	**AJMP** (P2) [2B, 2C]	**ACALL** (P2) [2B, 2C]	**AJMP** (P3) [2B, 2C]	**ACALL** (P3) [2B, 2C]
2					**ORL** **dir, A** [2B]	**ANL** **dir, A** [2B]	**XRL** **dir, A** [2B]	**ORL** **C, bit** [2B, 2C]
3								
4			**ADD** **A, #data** [2B]	**ADDC** **A, #data** [2B]	**ORL** **A, #data** [2B]	**ANL** **A, #data** [2B]	**XRL** **A, #data** [2B]	**MOV** **A, #data** [2B]
5	**INC** **dir** [2B]	**DEC** **dir** [2B]	**ADD** **A, dir** [2B]	**ADDC** **A, dir** [2B]	**ORL** **A, dir** [2B]	**ANL** **A, dir** [2B]	**XRL** **A, dir** [2B]	
6								**MOV** **@R0, @data** [2B]
7								**MOV** **@R1, #data** [2B]
8								**MOV** **R0, #data** [2B]
9								**MOV** **R1, #data** [2B]
A								**MOV** **R2, #data** [2B]
B								**MOV** **R3, #data** [2B]
C								**MOV** **R4, #data** [2B]
D								**MOV** **R5, #data** [2B]
E								**MOV** **R6, #data** [2B]
F								**MOV** **R7, #data** [2B]

注　[2B]=2 Byte, [3B]=3Byte, [2C]=2Cycle, [4C]=4 Cycle, Blank=1 byte/1 cycle。

8051 指令集——双字节指令整理 2

	8	9	A	B	C	D	E	F
0	**SJMP** **REL** [2B, 2C]		**ORL** **C, /bit** [2B, 2C]	**ANL** **C, /bit** [2B, 2C]	**PUSH** **dir** [2B, 2C]	**POP** **dir** [2B, 2C]		
1	**AJMP** (P4) [2B, 2C]	**ACALL** (P4) [2B, 2C]	**AJMP** (P5) [2B, 2C]	**ACALL** (P5) [2B, 2C]	**AJMP** (P6) [2B, 2C]	**ACALL** (P6) [2B, 2C]	**AJMP** (P7) [2B, 2C]	**ACALL** (P7) [2B, 2C]
2	**ANL** **C, bit** [2B, 2C]	**MOV** **bit, C** [2B, 2C]	**MOV** **C, bit** [2B]	**CPL** **bit** [2B]	**CLR** **bit** [2B]	**SETB** **bit** [2B]		
3								
4	**DIV** **AB** [2B, 4C]	**SUBB** **A, #data** [2B]						
5		**SUBB** **A, dir** [2B]			**XCH** **A, dir** [2B]		**MOV** **A, dir** [2B]	**MOV** **dir, A** [2B]
6	**MOV** **dir, @R0** [2B, 2C]		**MOV** **@R0, dir** [2B, 2C]					
7	**MOV** **dir, @R1** [2B, 2C]		**MOV** **@R1, dir** [2B, 2C]					
8	**MOV** **dir, R0** [2B, 2C]		**MOV** **R0, dir** [2B, 2C]			**DJNZ** **R0, rel** [2B, 2C]		
9	**MOV** **dir, R1** [2B, 2C]		**MOV** **R1, dir** [2B, 2C]			**DJNZ** **R1, rel** [2B, 2C]		
A	**MOV** **dir, R2** [2B, 2C]		**MOV** **R2, dir** [2B, 2C]			**DJNZ** **R2, rel** [2B, 2C]		
B	**MOV** **dir, R3** [2B, 2C]		**MOV** **R3, dir** [2B, 2C]			**DJNZ** **R3, rel** [2B, 2C]		
C	**MOV** **dir, R4** [2B, 2C]		**MOV** **R4, dir** [2B, 2C]			**DJNZ** **R4, rel** [2B, 2C]		
D	**MOV** **dir, R5** [2B, 2C]		**MOV** **R5, dir** [2B, 2C]			**DJNZ** **R5, rel** [2B, 2C]		
E	**MOV** **dir, R6** [2B, 2C]		**MOV** **R6, dir** [2B, 2C]			**DJNZ** **R6, rel** [2B, 2C]		
F	**MOV** **dir, R7** [2B, 2C]		**MOV** **R7, dir** [2B, 2C]			**DJNZ** **R7, rel** [2B, 2C]		

注　[2B]=2 Byte, [3B]=3Byte, [2C]=2Cycle, [4C]=4 Cycle, Blank=1 byte/1 cycle。

8051 指令集——3 字节指令整理 1

	0	1	2	3	4	5	6	7
0		**JBC** **bit, rel** [3B, 2C]	**JB** **bit, rel** [3B, 2C]	**JNB** **bit, rel** [3B, 2C]				
1								
2	**LJMP** **addr16** [3B, 2C]	**LCALL** **addr16** [3B, 2C]						
3					**ORL** **dir, #data** [3B, 2C]	**ANL** **dir, #data** [3B, 2C]	**XRL** **dir, #data** [3B, 2C]	
4								
5								**MOV** **dir, #data** [3B, 2C]
6								
7								
8								
9								
A								
B								
C								
D								
E								
F								

注　[2B]=2 Byte, [3B]=3Byte, [2C]=2Cycle, [4C]=4 Cycle, Blank=1 byte/1 cycle。

8051 指令集——3 字节指令整理 2

	8	9	A	B	C	D	E	F
0		**MOV DPTR, #data 16** [3B, 2C]						
1								
2								
3								
4				**CJNE A, #data, rel** [3B, 2C]				
5	**MOV dir, dir** [3B, 2C]			**CJNE A, dir, rel** [3B, 2C]		**DJNZ dir, rel** [3B, 2C]		
6				**CJNE @R0, #data, rel** [3B, 2C]				
7				**CJNE @R1, #data, rel** [3B, 2C]				
8				**CJNE R0, #data, rel** [3B, 2C]				
9				**CJNE R1, #data, rel** [3B, 2C]				
A				**CJNE R2, #data, rel** [3B, 2C]				
B				**CJNE R3, #data, rel** [3B, 2C]				
C				**CJNE R4, #data, rel** [3B, 2C]				
D				**CJNE R5, #data, rel** [3B, 2C]				
E				**CJNE R6, #data, rel** [3B, 2C]				
F				**CJNE R7, #data, rel** [3B, 2C]				

注　[2B]=2 Byte, [3B]=3Byte, [2C]=2Cycle, [4C]=4 Cycle, Blank=1 byte/1 cycle。

附录 D　8051 指令整理（按功能划分）

助记符		说明	字节	指令周期
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Rj	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract Register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment direct RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement register	1	12
DEC	direct	Derement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A&B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12

注　All mnemonics copyrighted @Intel Corp., 1980.

助记符		说明	字节	指令周期
LOGICAL OPERATIONS				
ANL	A,Rn	AND register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12

助记符		说明	字节	指令周期
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to DPTR to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move Exernal RAM(16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit ddr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

助记符		说明	字节	指令周期
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear diret bit	2	12
SETB	c	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	REL	Jump if Carry is set	2	24
JNC	REL	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24

助记符		说明	字节	指令周期
PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	dfrect,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

附录 E　8051 指令整理（按十六进制排列）

十六进制代码	字　节　数	助　记　符	操　作　数
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@ R0
07	1	INC	@ R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@ R0
17	1	DEC	@ R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	

续表

十六进制代码	字 节 数	助 记 符	操 作 数
23	1	RL	A
24	2	ADD	A, #data
25	2	ADD	A, data addr
26	1	ADD	A, @R0
27	1	ADD	A, @ R1
28	1	ADD	A, R0
29	1	ADD	A, R1
2A	1	ADD	A, R2
2B	1	ADD	A, R3
2C	1	ADD	A, R4
2D	1	ADD	A, R5
2E	1	ADD	A, R6
2F	1	ADD	A, R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A, #data
35	2	ADDC	A, data addr
36	1	ADDC	A, @R0
37	1	ADDC	A, @ R1
38	1	ADDC	A, R0
39	1	ADDC	A, R1
3A	1	ADDC	A, R2
3B	1	ADDC	A, R3
3C	1	ADDC	A, R4
3D	1	ADDC	A, R5
3E	1	ADDC	A, R6
3F	1	ADDC	A, R7
40	1	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr, A
43	3	ORL	data addr, #data
44	2	ORL	A, #data
45	2	ORL	A, data addr
46	1	ORL	A, @R0
47	1	ORL	A, @ R1
48	1	ORL	A, R0
49	1	ORL	A, R1

续表

十六进制代码	字 节 数	助 记 符	操 作 数
4A	1	ORL	A, R2
4B	1	ORL	A, R3
4C	1	ORL	A, R4
4D	1	ORL	A, R5
4E	1	ORL	A, R6
4F	1	ORL	A, R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr, A
53	3	ANL	data addr, #data
54	2	ANL	A, #data
55	2	ANL	A, data addr
56	1	ANL	A, @R0
57	1	ANL	A, @ R1
58	1	ANL	A, R0
59	1	ANL	A, R1
5A	1	ANL	A, R2
5B	1	ANL	A, R3
5C	1	ANL	A, R4
5D	1	ANL	A, R5
5E	1	ANL	A, R6
5F	1	ANL	A, R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	date addr, A
63	3	XRL	date addr, #data
64	2	XRL	A, #data
65	2	XRL	A, data addr
66	1	XRL	A, @R0
67	1	XRL	A, @ R1
68	1	XRL	A, R0
69	1	XRL	A, R1
6A	1	XRL	A, R2
6B	1	XRL	A, R3
6C	1	XRL	A, R4
6D	1	XRL	A, R5
6E	1	XRL	A, R6
6F	1	XRL	A, R7
70	2	JNZ	code addr

续表

十六进制代码	字 节 数	助 记 符	操 作 数
71	2	ACALL	code addr
72	2	ORL	C, bit addr
73	1	JMP	@ A+DPTR
74	2	MOV	A, #data
75	3	MOV	data addr, #data
76	2	MOV	@ R0, #data
77	2	MOV	@ R1, #data
78	2	MOV	R0, #data
79	2	MOV	R1, #data
7A	2	MOV	R2, #data
7B	2	MOV	R3, #data
7C	2	MOV	R4, #data
7D	2	MOV	R5, #data
7E	2	MOV	R6, #data
7F	2	MOV	R7, #data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C, bit addr
83	1	MOVC	A, @A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr, @ R0
87	2	MOV	data addr, @ R1
88	2	MOV	data addr, R0
89	2	MOV	data addr, R1
8A	2	MOV	data addr, R2
8B	2	MOV	data addr, R3
8C	2	MOV	data addr, R4
8D	2	MOV	data addr, R5
8E	2	MOV	data addr, R6
8F	2	MOV	data addr, R7
90	3	MOV	DPTR, #data
91	2	ACALL	code addr
92	2	MOV	bit addr, C
93	1	MOVC	A, @ A+DPTR
94	2	SUBB	A, #data
95	2	SUBB	A, data addr
96	1	SUBB	A, @R0

续表

十六进制代码	字 节 数	助 记 符	操 作 数
97	1	SUBB	A, @ R1
98	1	SUBB	A, R0
99	1	SUBB	A, R1
9A	1	SUBB	A, R2
9B	1	SUBB	A, R3
9C	1	SUBB	A, R4
9D	1	SUBB	A, R5
9E	1	SUBB	A, R6
9F	1	SUBB	A, R7
A0	2	ORL	C, /bit addr
A1	2	AJMP	code addr
A2	2	MOV	C, bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		rese rved	
A6	2	MOV	@ R0, data addr
A7	2	MOV	@ R1, data addr
A8	2	MOV	R0, data addr
A9	2	MOV	R1, data addr
AA	2	MOV	R2, data addr
AB	2	MOV	R3, data addr
AC	2	MOV	R4, data addr
AD	2	MOV	R5, data addr
AE	2	MOV	R6, data addr
AF	2	MOV	R7, data addr
B0	2	ANL	C, /bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A, #data, code addr
B5	3	CJNE	A, data addr, code addr
B6	3	CJNE	@ R0, #data, code addr
B7	3	CJNE	@ R1, #data, code addr
B8	3	CJNE	R0, #data, code addr
B9	3	CJNE	R1, #data, code addr
BA	3	CJNE	R2, #data, code addr
BB	3	CJNE	R3, #data, code addr
BC	3	CJNE	R4, #data, code addr

续表

十六进制代码	字 节 数	助 记 符	操 作 数
BD	3	CJNE	R5, #data, code addr
BE	3	CJNE	R6, #data, code addr
BF	3	CJNE	R7, #data, code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A, data addr
C6	1	XCH	A, @R0
C7	1	XCH	A, @ R1
C8	1	XCH	A, R0
C9	1	XCH	A, R1
CA	1	XCH	A, R2
CB	1	XCH	A, R3
CC	1	XCH	A, R4
CD	1	XCH	A, R5
CE	1	XCH	A, R6
CF	1	XCH	A, R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	A, data addr, code addr
D6	1	XCHD	A, @R0
D7	1	XCHD	A, @ R1
D8	2	DJNZ	R0, code addr
D9	2	DJNZ	R1, code addr
DA	2	DJNZ	R2, code addr
DB	2	DJNZ	R3, code addr
DC	2	DJNZ	R4, code addr
DD	2	DJNZ	R5, code addr
DE	2	DJNZ	R6, code addr

续表

十六进制代码	字 节 数	助 记 符	操 作 数
DF	2	DJNZ	R7, code addr
E0	1	MOVX	A, @DPTR
El	2	AJMP	code addr
E2	1	MOVX	A, @R0
E3	1	MOVX	A, @ R1
E4	1	CLR	A
E5	2	MOV	A, data addr
E6	1	MOV	A, @R0
E7	1	MOV	A, @ R1
E8	1	MOV	A, R0
E9	1	MOV	A, R1
EA	1	MOV	A, R2
EB	1	MOV	A, R3
EC	1	MOV	A, R4
ED	1	MOV	A, R5
EE	1	MOV	A, R6
EF	1	MOV	A, R7
F0	1	MOVX	@ DPTR, A
F1	2	ACALL	code addr
F2	1	MOVX	@R0, A
F3	1	MOVX	@R1, A
F4	1	CPL	A
F5	2	MOV	data addr A
F6	1	MOV	@R0, A
F7	1	MOV	@R1, A
F8	1	MOV	R0, A
F9	1	MOV	R1, A
FA	1	MOV	R2, A
FB	1	MOV	R3, A
FC	1	MOV	R4, A
FD	1	MOV	R5, A
FE	1	MOV	R6, A
FF	1	MOV	R7, A

附录 F　8051 SFR 表与 RESET 后的初始值

F8H									FFH
F0H	B 00000000								F7H
E8H									EFH
E0H	ACC 00000000								E7H
D8H									DFH
D0H	PSW 00000000								D7H
C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			CFH
C0H									C7H
B8H	IP XX000000								BFH
B0H	P3 11111111								B7H
A8H	IE 0X000000								AFH
A0H	P2 11111111								A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	THl 00000000	AUXR XXX00XX0		8FH
80H	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000		PCON 0XXX0000	87H

附录 G　SFR 特殊功能寄存器整理表

寄存器名称	地　址	可位寻址	8051	8052
ACC 累加器	E0H	*	*	*
B 通用寄存器	F0H	*	*	*
PSW 程序状态字	D0H	*	*	*
SP 堆栈指针	81H		*	*
DPH 数据指针（高位）	83H		*	*
DPL 数据指针（低位）	82H		*	*
P0 端口 0	80H	*	*	*
P1 端口 1	90H	*	*	*
P2 端口 2	A0H	*	*	*
P3 端口 3	B0H	*	*	*
IP 中断优先控制	B8H	*	*	*
IE 中断使能控制	A8H	*	*	*
TMOD 定时/计数方式控制	89H		*	*
TCON 定时/计数控制	88H	*	*	*
T2CON 第 2 定时/计数控制	C8H	*		*
TH0 TIMER0 高位设置	8CH		*	*
TL0 TIMER0 低位设置	8AH		*	*
TH1 TIMER1 高位设置	8DH		*	*
TL1 TIMER1 低位设置	8BH		*	*
TH2 TIMER2 高位设置	CDH			*
TL2 TIMER2 低位设置	CCH			*
RCAP2H 捕获寄存器（高位）	CBH			*
RCAP2L 捕获寄存器（低位）	CAH			*
SCON 串行通信控制器	98H	*	*	*
SBUF 串行数据缓冲器	99H		*	*
PCON 电源控制寄存器	87H		*	*

PSW 各位定义

（D0H）

CY	AC	F0	RS1	RS0	0V	—	P
D7	D6	D5	D4	D3	D2	D1	D0

CY（PSW.7）　：运算进制标志位
AC（PSW.6）　：辅助运算标志位

F0（PSW.5）　　：用户自行定义用标志位
RS1（PSW.4）　　：R0～R7 寄存器组选用位
RS0（PSW.3）　　：R0～R7 寄存器组选用位
0V（PSW.2）　　：运算溢位标志位
-（PSW.1）　　：Intel 保留待未来开发用
P（PSW.0）　　：ACC 内容同位标志位

注：

（1）RS1 和 RS0 共可组成 4 个寄存器组（BANK），以切换数据存储器内的 R0～R7 值。

RS1,RS0=00，选用 BANK 0（00H～07H）
RS1,RS0=01，选用 BANK 1（08H～0FH）
RS1,RS0=10，选用 BANK 2（10H～17H）
RS1,RS0=11，选用 BANK 3（18H～1FH）

（2）PSW.0 即 Parity 校验位，通常在串行通信中，做数据检查用。

T2CON 各位定义

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2
D7	D6	D5	D4	D3	D2	D1	D0

TF2(T2CON.7)　　:TIMER2 的溢位标志位，可由软件清除
EXF2(T2CON.6)　　:TIMER2 的外部标志位
RCLK(T2CON.5)　　:串行接收时基标志位
TCLK(T2CON.4)　　:串行传送时基标志位
EXEN2(T2CON.3)　　:TIMER 2 外部使能标志位
TR2(T2CON.2)　　:TIMER 2 开始/停止控制位
C/T2(T2CON.1)　　:TIMER 2 选定计数或定时的功能
CP/RL2(T2CON.0)　:TIMER 2 的 Capture/Reload 标志位

注：8052 的 TIMER 2 操作较为复杂，详细操作请参考《8051 单片机彻底研究——实习篇》第 6 章 8051 与 8052 的差异中的介绍及说明。

SCON 串行控制端口位说明

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
D7	D6	D5	D4	D3	D2	D1	D0

SM0(SCON.7)　:串行通信模式设置位
SM1(SCON.6)　:串行通信模式设置位
SM2(SCON.5)　:多重处理器连线通信的功能控制
REN(SCON.4)　:串行通信接收使能，该位可用软件控制
TB8(SCON.3)　:串行通信第 2 和第 3 模式下欲送出的第 9 个位，可用软件设置或清除
RB8(SCON.2)　:串行通信第 2 和第 3 模式下欲送出的第 9 个位，可用软件读入再做判断
TI(SCON.1)　:串行传送中断标志位
RI(SCON.0)　:串行接收中断标志位

PCON 电源控制寄存器各位说明

SM0D	—	—	—	GF1	GF0	PD	IDL
D7	D6	D5	D4	D3	D2	D1	D0

SMOD(PCON.7)　　:双倍 Baud rate 设置
-(PCON.6)　　:原厂保留
-(PCON.5)　　:原厂保留

-(PCON.4)　　:原厂保留
GF1(PCON.3)　　:一般用途标志位(由用户自行定义)
GF0(PCON.2)　　:一般用途标志位(由用户自行定义)
PD(PCON.1)　　:降低功率控制位,等于1时会使CPU的功率下降
IDL (PCON.0)　　:系统闲置(Idle)控制位

注:①若PD和IDL同时为1,则仅有PD(Power Down)操作;②系统RESET之后,PCON的值为0XXX 000B。

IP中断优先级寄存器各位定义

—	—	PT2	PS	PT1	PX1	PT0	PX0
D7	D6	D5	D4	D3	D2	D1	D0

-(IP.7)　　:原厂保留
-(IP.6)　　:原厂保留
PT2(IP.5)　　:定义TIMER 2中断的优先次序
PS(IP.4)　　:定义串行通信中断的优先次序
PT1(IP.3)　　:定义TIMER 1中断的优先次序
PX1(IP.2)　　:定义INT 1外部中断的优先次序
PT0(IP.1)　　:定义TIMER 0中断的优先次序
PX0(IP.0)　　:定义INT 0外部中断的优先次序

注:①优先级仅分成两种,设成1时代表有较高优先级,设成0时则属低优先级;②若数个中断信号同时发生时,8051将先处理高优先级的中断,然后才是处理低优先级的中断。

IE中断使能控制寄存器各位定义

EA	—	ET2	ES	ET1	EX1	ET0	EX0
D7	D6	D5	D4	D3	D2	D1	D0

EA(IE.7)　　:CPU是否接受中断完全由此位控制,若EA=0,则CPU不接受任何中断,若EA=1,则按以下各位来决定是否接受该中断要求
-(IE.6)　　:原厂保留
ET2(IE.5)　　:TIMER 2的中断控制
ES(IE.4)　　:串行通信的中断控制
ET1(IE.3)　　:TIMER 1的中断控制
EX1(IE.2)　　:外部硬件中断(INT 1)的控制
ET0(IE.1)　　:TIMER 0的中断控制
EX0(IE.0)　　:外部硬件中断(INT 0)的控制

注:若bit=1代表允许该中断,bit=0则为禁止该中断,即纵使真有中断信号,但CPU仍不予处理。

附录 H　如何购买电子元件

买电子元件就好像购买磁盘一样，你必须知道详细规格后，才能买到正确的元件，而电子元件又比磁盘复杂许多，往往造成购买者不知如何订货，销售商也不清楚你要的是什么东西，这篇文章的主要用意是引导读者做正确的订购，进而完成软硬件实习。

一般正规厂家的电子元件（除非体积太小）都会在元件的空白处标明三项信息：厂牌、元件编号和制造日期，标明厂牌是一项负责任的态度，有些直接就将公司的英文名称印上，另外有些公司则印上公司特有的符号，以代表所生产的厂商，更进一步的也会同时印出是哪一个地方厂商所生产的。元件编号则代表这个元件的特有规格和应用范围，在大多场合下，相同编号的元件应该是能够互换的，但是在有些特殊场合中硬件上的若干差异，往往会造成整个电子电路的不正常操作，这方面往往有经验的钻研者才看出其中的原因，在这里我们还是先假定不同厂牌相同元件编号的电子元件应该是能够互换的。在购买时，我们应当以相同厂牌的元件为第一个优先条件，当条件不合时才考虑买替代品。元件上第三个标明的字样应该是制造日期了，通常以 4 个数字代表，前两个数字为制造年份（公元）、后两个数字是制造的周数，例如 0038 代表这个元件是 2000 年第 38 周所制造的，知道了元件上所提供的三项信息后，我们才不会买到不适用的电子元件。

1. 元件价格原则

（1）电子零售一定较贵，所以最好每次能一次购买多项元件。

（2）数量少一定比数量多的单价贵，集合多人的元件清单一次购买也会较划算。

（3）个人名义购买较公司名义购买贵，这是订购数量超过某个值后，才会凸显出的效果。

2. 到哪一家采购元件

商店的专业性（Professional）应该是第一项考虑的因素，当你走过电子专卖店时，若发现玻璃柜中摆的完全是各种式样的线路板或是各种形状的电子接头时，你大可安心地走进去购买线路板或是接头，反之如果摆满了各种完全不同的元件时，你就得费心去跟老板或店员沟通了。

在专卖店里，各种元件的相关数据信息比较齐全，到这种地方买元件往往会多学到一些知识，而且容易问对人。常见的元件专卖店有：

（1）音响专卖店：随身听、CD 数字音响等。

（2）电视游戏机专卖店：国内外各种厂牌的 TV 游戏机。

（3）电路 IC 专卖店：各国数字和仿真 IC。

（4）PC 板专卖店：专业制作线路板及各种万用线路板。

（5）高频元件专卖店：高频 RF 发射接收元件以及各种款式的天线。

（6）测试仪器专卖店：示波器及信号发生器等。

此外，还有专卖电子加工设备的商店（烙铁、锡炉、电动工具等），提供了练习时必备的工具，假如你想自行做线路的连接时，两三种颜色的镀银线是少不了的，这些东西在专卖店中都可以买到而且价格也较实在。

3. 如何购买元件

知道了在哪个地点买元件后，接下来就是进到电子元件商店买东西了，一般初学者都是在看到文章或线路说明后，才想买些元件回家试试看，如果你的情况也是这样，请记得顺便

将参考文章也复印一份带在身边，以便随时参考。我们分别以两种情况来说明正确的元件购买方法。

状况一（错误示范）：

问：我要买 LT485，请问你们有没有卖？

答：LT485？（停了一会儿）好像没有。

问：那其他家是不是买得到呢？

答：不知道（回答得很干脆）。

这种情况的发生率相当高，问的人无法掌握重点，回答的人若对元件编号很熟时，往往以没有元件为理由而结束这段谈话。以相同的问法多问几家八成都是碰钉子的份。

状况二（正确的元件询问）：

问：先生，我要买一个 RS485 线路上用的 LT485 IC 元件，不知道这儿是否有卖？

答：RS485 上用的元件？我帮你查一下。通常 RS-485 都是使用 TISN75176 或 NS 的 75176 这两个元件，你要的这个元件应该是新开发出来的吧！

问：没错，拿出带在身上的线路图和说明，先生你参考一下，它里面说 LT485 可以取代传统使用的 75176，而且性能更佳，厂牌是 LINEAR TECHNOLOGY 这家公司，不知是否有代理商？

答：有的，我的印象中 LT 的代理商好像是 XX 公司，不过，很抱歉我们这里并没有这个元件，可是依照你的线路上的功用看来，使用 75176 也是可以的，只是你还要的元件比较省电，从数据上看来，LT485 的耗电量只有 75176 的十分之一而已。但是我们这里只有 75176，你还要吗？

问：如果改用你所说的元件时，IC 引脚相不相符，这样行得通吗？

答：引脚完全一样，不信的话这边有 75176 的，你可以拿去复印后，待会儿再还我就可以了。

问：那就各买两个好了，拿回家试试看。

答：好的，电源记得要接正确。

购买者事先就准备好资料，并且在购买时又提到元件的真正用途，所以销售商（假如他已有相当的电子元件基础）就能明白购买者的真正需要，进而建议其他的代用元件。当然，我们并不可能要求所有的专卖店都有如此高的水准，但是能回答你问题的专卖店还是存在的。

4. 小心一手价钱买到二手元件

除非你事先知道你是以较低的价钱，去买二手（从其他线路板上拔起来）的元件，否则应该是以新品的价格去买电子元件，为了防止旧品新卖的情形，我们可以用下列几种方法判断元件是否为新品：

（1）检查制造日期，新品的制造日期应该在最近两年内，若元件上是制造日期标示着 9635，你会认为它是新的吗？

（2）检查元件的引脚是否有生锈无光泽，引脚会因时间过久而渐渐氧化，进而失去光泽。

（3）专卖店的风格，如果它的价格特别便宜或有兼卖杂牌的旧元件时，请特别留意该店卖给你所谓的新品，这里所谓的新品有时就是保证没有人用过的备用品。

附录 I　如何识别晶体管（三极管）的引脚

晶体管（三极管）只有 3 只引脚，引脚名称分别为集电极（Collector）、基极（Base）和发射极（Emitter），在电路设计上，我们用流经基极的微小电流（I_b）来控制集电极和发射极间的大电流（I_e），其关系式为 $I_e=I_b+I_c=(1+\beta)I_e$，其中的 β 值就是我们俗称的放大倍率。三极管基本上是用两个二极管（Diode）串接而成的，请看图 I-1 和图 I-2 所示的晶体管构造说明。

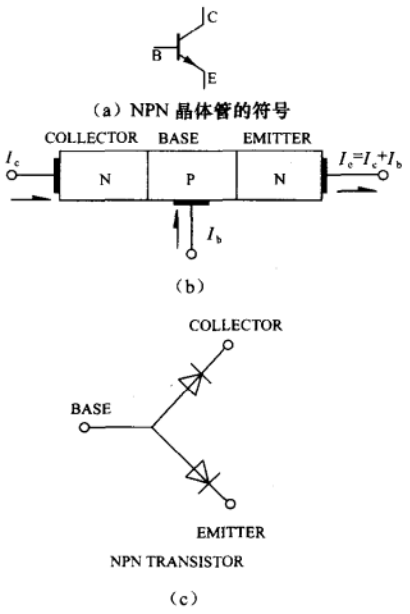


图 I-1　NPN 晶体管的符号与内部构造

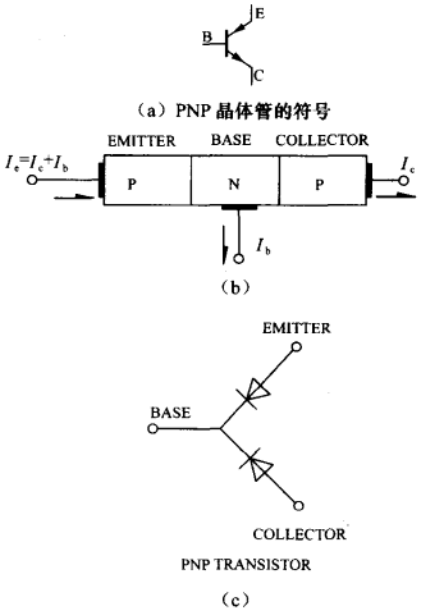


图 I-2　PNP 晶体管的符号与内部构造

在晶体管引脚测量时，图 I-1c 和图 I-2c 对我们的帮助最大，二极管的特性为单向导通，并且导通时其两端的电压降一定保持在 0.6～0.8V 间，只要把握此原则就很容易找出晶体管的基极来，实际测量时只需一台数字电表即可，首先我们将数字电表切到测量二极管的挡位，找来一个发光二极管（也是单向导通），然后用红黑测试棒去接触 LED 的两根脚，看看何种接法可以点亮 LED。请看图 I-3，标准的 LED 引脚图，以及何种情况 LED 会被点亮，同时表头也会有电压值出现。

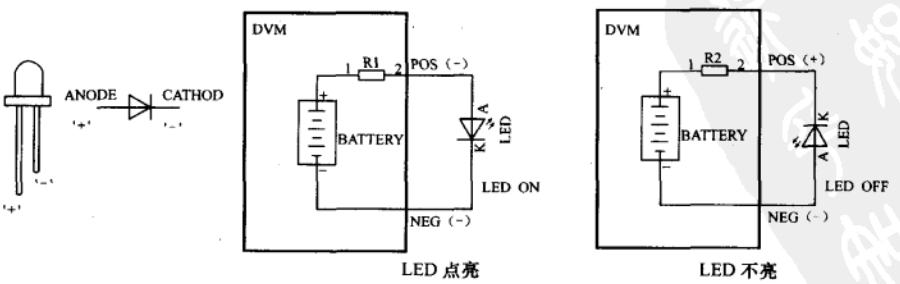


图 I-3　由 LED 的亮灭观察数字电表的正输出端

上述的步骤是很重要的，因为我们要先判断数字电表的测试棒上，哪一边是输出正电压，哪一边是输出负电压，测试时我们将红色测试棒接到电表标示（+）的地方，黑色测试棒接到标示 COM 即（-）的地方，试验后结果是数字电表的红色测试棒这边才是真正产生正电压的地方，此点请牢记在心。

如果我们找到一个已有标识 BCE 引脚的 NPN 晶体管，此时，我们将红色测试棒端摆在 B 极上，电表仍切换二极管测试挡位上，用黑色测试棒碰触 C 极和 E 极，此时应该可量到约 0.7V 的顺向电压，表示晶体管是处于顺向偏压状态，反之当黑色测试棒摆在 B 极上，用红色测试棒接触 C 极和 E 极，应该量不到任何电压值才正确，否则我们就应判定该晶体管已损坏了。

PNP 晶体管的量测情况刚好相反，请回头看 I-2c 表示图，当黑色测试棒接在 B 极上，红色测试棒碰触 C 极和 E 极时，数字电表可以量到一个约 0.7V 的电压值，而红色测试棒接到 B 极，以黑色测试棒碰触 C 或 E 极时，电表应该读不到值才算是正确。做这方面测试时，通常我们是左手拿着晶体管（引脚朝上），右手像使用筷子的方式拿着两支测试棒，以左手的转动来变换测试引脚。到目前为止，我们已经可以判断出晶体管是 NPN 型或是 PNP 型，还有 Base 基极的真正引脚。

晶体管 CE 脚的判定有两种方式，第一个方式为经验法，首先我们将电表切换到欧姆挡（最好固定在 100kΩ的欧姆挡），以晶体管 CE 极漏电流的多寡判断引脚。以 NPN 型的晶体管为例，若 B 极保持空接，E 到 C 极的漏电流会较 C 到 E 极的漏电流还大，这也就是说，E 到 C 的电阻值会较小，当把红色测试棒接 NPN 电晶体管的 E 极，黑色测试棒接 C 极时，量测到的电阻值较小，而把测试棒对调时，其电阻值会高到数兆欧以上。所以在判断 NPN 电晶体管的 CE 脚时，只要看到测试棒接触到 CE 极，且电阻值较小时，红色测试棒接触的这一端就是 E 极。

PNP 型晶体管测量时，同样方法先确定 B 极是哪一引脚，然后将测试棒跨接在 CE 极上，找出两种接法中何种接法电阻值较小，此时红色测试棒这一端就是 C 极。这个方法是由多次测试经验后所得到的，顺利的话，可在 15～20s 内判断出晶体管的 CBE 引脚来，不过如果判断过程中你有所怀疑时，请再用第二个方式做 double check，以免发生差错。

第二个判定 CE 极的方式为外加电阻辅助法，只要外加一个电阻在 CB 极（NPN 型晶体管）或 BE 极（PNP 型晶体管）上，就可以找出 C 极和 E 极来，测量时请将电表切换到欧姆挡，请看图 I-4 和图 I-5 的测量法，这是视晶体管是否有放大效果而找出 CE 极的正确位置。图中的 R 电阻范围由 100kΩ～1MΩ均可，甚至可用手指来替代都行，即运用拿晶体管的左手食指同时接触 BC 极或 BE 极，操作熟练后判断速度更快。

我们将一般的晶体管引脚判定程序简述整理如下，以方便检查未知极及引脚的晶体管。

先假定电晶体管是属 NPN 型的，找出晶体管的 B 极，若无法成功时，再假定晶体管是属于 PNP 型的，找出其基极（Base），若仍不成功时，则认定此晶体管非双极性型（Bipolar）或是该晶体管已经损坏。

使用 CE 极判断方式一，找出 C 和 E 极的位置。

再使用 CE 极判断方式二，确认方式一找出的 CE 极位置是否正确。

现在的数字电表价格已经不高了，通常其量测挡上有一个二极管测量挡，我们可以用此挡很方便地判断出晶体管的 NPN 及 PNP 极性，还有 B 极的位置。数字电表在二极管量测时，是以定电流输出（Constant Current）方式做为检测的依据，外加的电阻最好是一个固定数值，

此时用手指接触晶体管的引脚方法并不很正确。利用数字电表测量时，最好的方法是将晶体管暂时插在实习用的“面包”板上，再将外加的电阻插在其他空孔上，这样一来测试就非常顺手了，同时我们也可以从数字电表的面板上读到正确的电阻值。

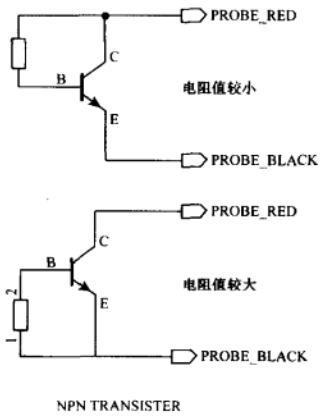


图 I-4　NPN 晶体管 CE 极的判断法，当在 CB 极间加上 100kΩ的电阻时，可以量到较小的Ω值，该电阻可用手指代替

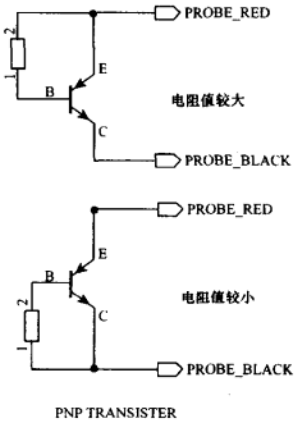


图 I-5　PNP 晶体管 CE 极的判断法，当在 BE 极间加上 100kΩ的电阻时，可以量到较小的Ω值，该电阻可用手指代替

以上所谈的都是理论性的，若想熟练的话，只要到电子元件行随意买 10～20 个晶体管，先不看晶体管编号，直接量测并判断 BCE 脚，然后再由编号去查出其确实的引脚，只要经过一次透彻的考验，就可以终生享用，何乐而不为呢？

附录 J　如何看 Data Sheet

每一个型号的 IC 都有属于自己的数据手册（Data Sheet），从 IC 的设计、规格、应用范例、封装技术到适用范围，在数据手册中都会清楚地提到，当然，每一家公司所生产同编号的 IC 在数据手册中的内容还是会不太一样，但在应用及脚位的安排上却是一样的，而其最大的差异，在于测试与分类的方式有所不同，以下便是针对如何阅读数据表所做的深入探讨。

1. 引脚安排与介绍

几乎所有生产 IC 的公司，其数据手册都会把 IC 的引脚说明安排放在最前面，其中一定会标示的引脚是电源脚（Vcc）及接地脚（GND），这是 IC 要运行最重要的引脚，一般的设计会把接地脚（GND）设在与第一脚同一边的最后一个脚位，而电源脚则是在接地脚的对角方向，也是 IC 的最后一个脚位，举例来说：如果 IC 有 16 个脚，那么接地脚通常会是第 8 脚，而电源脚则是第 16 脚，当然也是有很多例外的设计，不过以这样设计的电源引脚方式是比较有利于电路的设计。

至于其他的引脚，会因 IC 的功能而有明显的不同，以特定功能的 IC 来说，计数器的 IC 通常会有 CLK 的脉冲输入引脚，计数的输出引脚，RST 的重新设置引脚；Timer 则会有 TRIG 触发引脚，当然也少不了 RST 脚及输出的 OUT 引脚等，碍于篇幅，在此便不多做介绍，而这些引脚的功能与简称，会在数据手册的一开始便清楚地描述，以便在后面的资料中可以用较简洁的方式来表示各个引脚之间的相互关系及应用方法。

2. 系统框图或等效逻辑电路图

在数据手册中的第二个重点，便是系统框图及等效逻辑电路图。一般来说，特定功能的 IC 通常会把等效电路图标示出来，而可编程的数字 IC 则会把系统的框图标示出来，这是方便电路设计者可以更清楚地知道 IC 的工作原理，及此 IC 适不适用于某个设计电路的重要指针，但是要注意的是：在这里所标示的只是系统的示意图，实际上的功能与应用范围仍必须参考后续的数据才能下定论，这里只是提供你做个简单的参考，而在稍后的资料中，其运算上可能会用到这里所介绍的一些相关电路，所以这里也要稍做一下思考，不要太快就跳过去。

3. 直流或交流电源下的特性参数

这里可以说是 IC 重要规格的一览表，也是设计主要的参考依据，这里会清楚地标示 IC 的工作电压范围，工作电流范围，逻辑高低电压的电平等指标，这些参数是用来提醒你设计上所要注意的一些细节。比如说，如果设计时所提供的工作电压过高，可能会导致芯片烧毁；如果电压不足，芯片又可能不正常运行或是根本就不操作；输出的电压电平不足，便可能需要加入其他放大器来协助电位的上拉，在此就必须仔细地将每一个参数都稍作了解，以免在电路设计时因疏忽而导致不可挽救的错误。

4. 特性测试图表

这里会标示 IC 在某种的特性变化时，所产生的一些相对关系，比如说在固定电压下，在不同的振荡频率工作时，所需要的电流量有何不同？或是芯片在不同温度下工作时，其电压与电流的输出入会有怎样的变化？这都是在特性测试的图表中可以找得到答案，当然，越负责任的厂商会把越多相关的信息摆在这里，相对的 IC 价格也就会比较高，一分钱一分货，在 IC 的领域里这似乎是不变的法则。

5. 应用电路的范例

通常在IC的数据手册中还会有一些电路的应用范例，并附上运算的公式及应用的范围，这便有助于一个从事系统开发者早一点熟练使用这个IC的方法，而不用盲目地摸索测试，减少很多设计上的成本，也有助于系统设计周期的缩短。参考几家公司生产的同型IC电路范例，便可以集思广益，更了解IC的电路特性及应用的方式，这里绝对值得你多花点时间研究研究的。

6. 封装的尺寸一览

对于从事电路板线路设计的技术人员来说，这是相当宝贵的资料，因为整个IC详细的尺寸都在这里标示得一清二楚，在布线时要建立新元件就不用再拿着一把游标尺边量边猜了。不过并不是所有的IC都会附上封装的数据，常用的TTL或是CMOS芯片，其规格应该是全世界统一的规格，因此在多数的线路设计软件中都会把这些数据建入资料库，生产IC的厂商便不需要把这些数据编成数据手册了。如果你还是需要这些数据时，也可以跟厂商索取完整的光盘数据，那内容可能就会比一般在网络上流传的数据手册还详尽许多。

7. 法律责任的问题

没想到还有这样的信息吧？其实一个负责任的厂商，会把IC应用的范围及责任归属一并写在数据手册里，拿Atmel的AT89C2051芯片的数据表来说，在内容最后就有提到这样一段话："Atmel's products are not authorized for use as critical components in life support devices or systems."这句话大概的意思是：当你设计有关于生命维持的设备或系统时，这个IC是不适用的。哪些是属于生命维持设备或系统呢？比如心律调整器、特护病房用的维生设备，如果这些设备在不该停的时候停了，导致一个生命的结束，归究原因后发现是芯片损坏所导致的结果，那么Atmel公司是不会负任何的责任，因为在数据表中早已明确这个IC的适用场合，如果设计者没有注意到这一段法律声明的话，出事的时候是会吃上官司的。

附录K 如何焊接

焊接是一个电子技术人员所必须具备的基本能力之一，当然在这里所提到的焊接指的是电路板的锡焊，和机械工程上所应用的电焊、点焊等多种焊接的方式是有所不同的。一个合格电路设计与开发的技术人员，除了具备电子电路上的丰富知识之外，想要有一流的电路检修能力还必须具备熟练的焊接技巧，否则在实战中是无法与人相比的。

1. 焊接工具的介绍

在正式介绍焊接方法之前，我们先简单地介绍焊接所要准备的基本工具：

（1）烙铁。对于一个初学者而言，一把实用的烙铁是不可或缺的，市场上的烙铁通常分为电烙铁及瓦斯烙铁两大类，如果你所要从事焊接工作的环境大多在有 110V 或 220V 交流电源插座的地方，那么购买电烙铁会是你最佳的选择；相反地，如果你所要从事焊接工作的环境无法取得 AC 电源的话，那么一把好的瓦斯烙铁就是你最佳的选择了！在此以大多数人使用上的习惯为参考依据，特地针对电烙铁作介绍。

一般来说，由于低阶的电烙铁构造较为简单，所以成本上会明显比瓦斯烙铁便宜很多，一把 40W 的电烙铁大约可以提供 200～300℃的工作温度，足够应付大多数电路板的焊接工作。以我们的经验，就一个初学者来说，一把 40W 的电烙铁应该是相当够用了，当你以这把烙铁用到它烧坏的时候，你的焊接技术也差不多是高手了！这时可以选择高级有温控的电烙铁，你可以视自己的需求来选购适合的烙铁。

（2）焊锡。有了烙铁却没有焊锡，就像有了车子却没有轮子一般，选购品质良好的焊锡，可以使电子元件牢靠地固定在接点上，且引脚不容易氧化而导致接触不良或阻抗过大。焊锡本身是一种特殊的合金，主要的成分为锡及铅，市面上的焊锡也有加入银以增加其导电性，其价格会比一般的锡铅合金的焊锡贵上数倍，除非你的电路有特别的需求，或是对杂波的处理有较高的标准，否则选用锡铅比例为 68:32，一千克 0.8mm 焊锡丝就能满足各种电路在焊接上的需求了。

2. 焊接的介绍

以下的部分，要探讨焊接的技巧及常犯的错误。

♦ 焊接的技巧：为了方便介绍，在此所提的焊接方法是以电烙铁的焊接技巧为主。

（1）在焊接前先要把烙铁插电预热，将烙铁架上的泡绵先泡水弄湿，并准备好前面所提到的工具及你所要焊接的电子元件及电路板。

（2）预热约 3min 后即可开始焊接，在此之前可以先用焊锡丝接触烙铁前端，确认焊接温度已经足够。如果焊锡可以顺利融化，那么就可以开始进行焊接了。

（3）烙铁的正确拿法应该是以握笔的方式为最佳，另外一手则拿着焊锡。焊接时烙铁尖端和电路板的夹角以 45°左右为最佳。

（4）焊接时先将电子元件引脚折好置于电路板上，再把电路板翻面压好，以烙铁尖端接触在电子元件的引脚上，再以焊锡去接触，见焊锡一熔解便可以马上移开烙铁。

（5）焊接电子元件的顺序最好先从较矮的电子元件先焊接，在翻面固定时会比较容易些，养成习惯后可以很快就将电路板焊好，操作上也比较顺手。

（6）焊好的焊点应该是呈内凹弧线的圆锥状焊点，如果你的焊点是球状的，那么你的焊接功力还必须多多磨练！球状的焊点很可能内部是中空的，会形成所谓的“假焊或虚焊”状

态，导致电路不通或导通不良信号断断续续的现象，如果有这样的焊点，建议用吸锡器将锡吸走重焊，多一分训练就多一分熟练，焊接没有捷径，多练习就可以变成高手。

（7）焊接二极管、IC、晶体管、电容、SMD 等电子元件时，请不要将烙铁碰触引脚超过5s，否则可能会造成元件内的线路烧毁或短路。

◆ 特殊电子元件的焊接

SMD：这是焊接难度比较高的电子元件，一开始最重要的还是先确认每一支脚位是否都摆放到正确的位置，接着将对角的两支引脚先焊上，要焊接时先在烙铁尖端上吃一点点的焊锡，再直接点到 SMD 的两个对角引脚上，等两对角都固定后，将一大团的锡直接熔化在 SMD 的一端，以拖曳的方式将焊锡“拖”到每一支脚位上，再依同样的方式在其他各边都焊好。

SMD 焊好后，最好可以用放大镜再确认每一支引脚是否都焊上，而且没有跨接的情况，经过 SMD 的焊接训练，相信所有的焊接对您来说都是轻而易举的。

3. 注意事项

焊接本身是具有危险性的工作，烙铁高温下可能造成操作者烫伤，另外，焊锡熔解时的化学烟雾，电子元件引脚上的化学溶剂残留，这些都可能对你的身体造成伤害，所以请以谨慎的态度来面对焊接这项工作，要是掉以轻心的话，危险就在不知不觉中悄悄来临。

附录 L　如何上网找元件

这是一个网络科技日新月异的时代，网络信息为我们带来了很多的便利性，如果不能加以好好利用，对生活在这时代的我们实在有些可惜。下面介绍如何上网搜索电子元件数据的方法。

对于技术性的文章，我们通常会建议通过搜索引擎 Google 查找，它的网址是 http://www.google.com，在这里可以找到很多你所需要的专业知识网站，及一些你可能觉得很冷门、很难找的技术文件。如果你有比较充裕的上网时间，建议你不妨多多利用这个搜索引擎，但是这并不保证你所要找的数据一定可以找得到！但应该能提供一些相关的线索，不会让你像无头苍蝇般找不到头绪。举个例子来说，假如我想要找 AT89C2051 芯片的相关资料，我应该怎么找呢？

最简单的做法就是直接把 AT89C2051 输入搜索引擎中，然后按搜索键搜索，这样所找到的网站链接一定是完整提到这个字符串的数据，当然这样的找法也是最笨的方法，运气好的话，你可以马上就得到这个芯片的数据；运气不好的话，可能连相关网站都找不到！比较好的方法应该是下面所要介绍的方法：

搜索引擎中有两个很好用的符号，一个是“+”（或），另一个是“&”（与），这两个符号在使用上大同小异，但搜索的结果却不太相同，虽然这并不是我们所要探讨的重点，但稍后我们用到这两个符号的机会是非常频繁的，所以要先把他们牢牢记住。

回到主题，现在我们所要找的是芯片，编号是“AT89C2051”，所以比较好的搜索字符串应该是输入“chip+AT89C2051”，这样子可以确保你所找的数据应该是跟芯片有关，而不会找到某种叫“AT89C2051”的赛车编号或是准考证号码。

若是搜索后所得到的结果，数据有数百个甚至上千个，那么我又该从何看起呢？千万不要一个一个看，也不要走马观花挑着看，这时你所要看的是这些网站出现关键字的共性，通常这个共性只要看前面三页就可以发现其端倪它以现在的主题为例，你可以发现大多数的网站都会出现“ATMEL”的字样，它并不是“AT89C2051”，但是它却是生产“AT89C2051”的制造厂商，因此下一个操作就是搜索“ATMEL”，直接从“ATMEL”的网站上搜索相关数据。

到此为止，你应该可以正确地找到有关 AT89C2051 的官方资料，其中应该包含芯片规格、外观、线路图、应用范例、指令集等，当然只有这些资料是不够的，就像买东西一样，我们知道产品的功能、使用方法和使用场合，但是最想知道的是他贵不贵、好不好用、用过的人对这项产品的评价信息，假如你搜索工作到此就打住了，那么你的损失就很大！至于其他信息的搜索，可以按照前面所提到的方法，用类别字（如 chip、IC、Atmel 等）加上关键字（AT89C2051）的方法来寻找，相信你会有很多的收获的。

依照前面所提的方法，如果还是无法找到你所要找的元件，那么你还可以这么做，那么我们会建议你用较另类的方法来搜索。

假如你对这个芯片的编号有一点概念的话，那么下面的方法会帮你找到更多你所想象不到的资料，这个方法叫做“拆字法”。

“拆字法”，顾名思义就是要把字符串拆开，可是拆字的原则是啥？再以"AT89C2051"为例，一般的习惯会拆成 AT89C+2051 或是 AT+89C+2051 两种方式，再加上 chip、IC 之

类的类别字，这样所找到的资料应该也不会离你所想象的太远，更高级一些，还可以把拆开的字分开搜索，如 2051 或是 AT89C，相信会让你对这个 IC 的相关资料有更深刻的了解。

以上所提到的方法，也可以把“+”全都换成“&”再试试看。拆字法的适用时机，是在你对所要找的电子元件完全没有头绪的情况下，用“土法炼钢”的方式又完全找不到资料时，可以孤注一掷的方法，当你用拆字的方法找过 10 来种电子元件的资料后，大概就没有难题可以难倒你了。

[General Information]
书名=8051单片机彻底研究：基础篇
作者=林伸茂编著
页数=350
出版社=中国电力出版社
出版日期=2007
SS号=11828731
DX号=161000068912
URL=http://book.szdnet.org.cn/bookDetail.jsp?dxNumber=16100006891 2&d=5AB1CFBC1F0839DB67F448F4AF4942CE