



# 8051

# 单片机

## 彻底研究 经验篇

林伸茂 编著



中国电力出版社  
www.infopower.com.cn



TP368.1  
441

8051单片机技术应用系列

# 8051

# 单片机

## 彻底研究 经验篇

林伸茂 编著



中国电力出版社  
[www.infopower.com.cn](http://www.infopower.com.cn)

## 内 容 提 要

本书共分成三大部分: 第一部分是应用研究篇, 介绍 8051 单片机的诸多设计理念与硬件保护手段。第二部分是介绍以 FLAG51 为主体的烧录器, 包括 Atmel Flash Microcontroller 的烧录、EPROM 的烧录以及 E<sup>2</sup>PROM 的烧录等。第三部分是历年来作者接触到设计项目的心得以及对一些问题的看法。

本书属于 8051 进阶级书籍, 适合对单片机已经有一些经验的读者, 或了解软硬件开发的工程师阅读。

### 图书在版编目 (CIP) 数据

8051 单片机彻底研究——经验篇 / 林仲茂编著. —北京: 中国电力出版社, 2007

(8051 单片机技术应用系列)

ISBN 978-7-5083-5153-7

I. 8... II. 林... III. 单片微型计算机—基本知识 IV. TP368.1

中国版本图书馆 CIP 数据核字 (2007) 第 004573 号

北京市版权局著作权合同登记号 图字: 01-2006-5849 号

### 版 权 声 明

本书简体中文版由旗标出版股份有限公司授权中国电力出版社出版, 其专有出版发行权由中国电力出版社所有, 未经出版者书面许可, 任何单位和个人均不得以任何理由或任何方式复制或抄袭本书的部分或全部内容。

责任编辑: 刘 焱

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——经验篇

编 著: 林仲茂

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: 北京丰源印刷厂

开本尺寸: 185 × 260 印 张: 17 字 数: 408 千字

书 号: ISBN 978-7-5083-5153-7

版 次: 2007 年 5 月北京第 1 版

印 次: 2007 年 5 月第 1 次印刷

印 数: 0001—4000

定 价: 32.00 元 (含 1CD)

### 敬 告 读 者

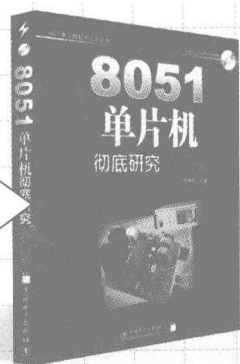
本书封面贴有防伪标签, 加热后中心图案消失

本书如有印装质量问题, 我社发行部负责退换

版 权 专 有 翻 印 必 究

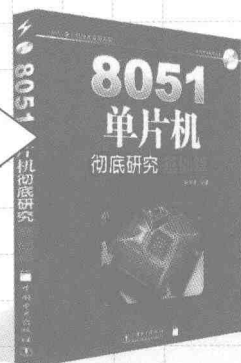
# 8051 单片机学习地图

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！



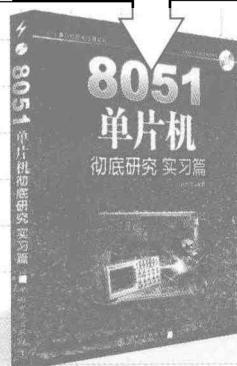
8051 单片机  
彻底研究——基础篇

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——经验篇

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——实习篇

# 序 言

## 知识分享的喜悦

20年来我一直坐在研究与开发的位置上，曾经在风和日丽的天气下进入台电核三厂的控制室内，为的是协助找出发电机组跳闸的原因；曾经在寒风冷雨的冬季到新竹工研院，查看仪器连线为何经常出问题的原因；曾经带着工具箱与手册，到鲜血尚未清洗干净的外科手术室里，检查自己设计的电动手术台是否有足够的安全性和稳定性；也曾经参与电动环保汽车的部分研发工作。

不管这些项目的结果如何，都有一个共同点：它们都是 8051 单片机的应用范例。每一个项目都是 8051 的深度应用。这也是 10 多年前我开始在《RUN!PC》杂志上写 8051 应用实例、陆陆续续大约写了三四十篇技术性文章的原因。与此同时，我个人的系统设计工作始终没有间断过。

20年来，我们碰到了许多技术上的瓶颈，很幸运我们解决了大部分，剩下的问题很可能不属于纯粹的技术问题了。那么这些宝贵的资料与经验永远放在旗威科技公司的文件柜中吗？我选择了出书这一方式，来把这些经验提供给广大读者朋友，虽然这样做很辛苦，但是套一句老话“喜欢做，甘愿受”。纵使外面的环境非常复杂，我个人还是认为值得这样做的。

千万不能只顾着吃鱼，也要学会打鱼。我个人觉得做任何研究或写程序应进一步掌握其实现过程。在本书中您会看到一件产品是被如何开发出来的，以及为何要这样处理，我们在书中都会有详细的说明。

我经常告诉别人：健康不能分享，金钱也不能分享，但是知识除了独享外，可以与别人分享。旗威科技公司很乐意邀请您一起来分享 8051 的专业知识，一起分享我们写的程序。

### 系列书籍介绍

《8051 单片机彻底研究——经验篇》作为 8051 单片机系列丛书的一部分，与《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》构成了一个系统的、有机的专业技术教程丛书。

如果您想彻底了解 8051 单片机与汇编语言的写法，请务必参考《8051 单片机彻底研究——基础篇》，书中会对 8051 各个指令的用法与寄存器的操作，做了深入详尽的交待。而《8051 单片机彻底研究——实习篇》主要强调在 8051 单片机系统扩展与集成应用两方面专业知识。

### 内容收录

本书的内容取材主要有以下三个方面：

历年来我们运用 8051 单片机的实例应用。旗威科技有限公司用 8051 单片机设计过 100 种以上的工业产品，产品范围涵盖医疗、精密测量、通信及自动化等专业领域。

取自《RUN!PC》杂志历年来所发表的技术与评论性文章，并删除部分不合时宜的内容，

最后又加上我们最近几年的设计心得。

网络上有关 8051 单片机应用与 C 语言的技术文献与报道，这方面我们深信外国人开放，许多资料在网站上唾手可得，不过只有您仔细消化后，才可以算是您自己的东西。

## 如何阅读本书

本书属于 8051 进阶书籍，适合对单片机已经有一些经验的读者，或横跨软硬件的工程师阅读。我们希望您在阅读本书时，已经对 8051 的结构与程序有基本的认识，例如 SFR、bit addressable、SBUF 等等。

本书共分成三大部分：

第一部分是应用研究篇，介绍 8051 单片机的诸多设计理念与硬件保护手段。想要以子之矛攻子之盾，在这里有第一手的解答。我们也提到与硬件保护始终对立的逆向工程与 8051 之间的关系。最后还有一章专门提到 8051 的优势与缺憾。8051 当初被人称道的功能至今被贬为重大瑕疵，为何如此？请看我们的深入分析。

第二部分是介绍以 FLAG51 为主体的烧录器，包括 Atmel Flash Microcontroller 的烧录、EPROM 的烧录以及 E<sup>2</sup>PROM 的烧录等，我们很乐意地公布这些烧录程序，而且是 100% 的源代码公布。以前我们彻底研究别人的程序为的是理解当初设计者的理念。今天我们公布程序内容，欢迎您来研究我们写的程序，如果您是用心的读者绝对可以从中获得更多的启示。

第三部分是历年来我们接触到设计项目的心得以及我们对一些问题的看法。

## 致谢

编写 8051 单片机系列书绝对不是单一个人所能完成的，它绝对是一个团队的工作总整合，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。我要特别感谢以下帮助我的人们：

王圣心小姐与姜莹贞小姐：初步整理已发表过的文章，光是校稿就校了多次，并拍摄许多额外的照片，让本系列的书籍得以完成初步的架构。

李浩霖先生与曾琼惠小姐：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。

罗仕林、庄显宏与黄芳川先生：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。

凌全伯先生：提供了对 8051 另类的观点，在本书第 6 章“8051 的是是非非”中，他也加入了独到的见解，面对 8 位 CPU 就要有 8 位的思维。

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，就不会有本系列丛书的问世。

林伸茂

chipware@chipware.com.tw

# 目 录

序 言	
第 1 章 8051 的汇编语言与 C 语言	2
1-1 汇编语言短小精干	2
1-2 C 语言可以缩短开发时间	3
1-3 8051 编译器使用经验谈	7
1-4 C 语言是全能的吗	9
第 2 章 单片机保护程序的方法	12
2-1 案例一：喷气发动机的源代码	12
2-2 案例二：苹果二号与桔子二号	12
2-3 案例三：仿冒的“快打旋风”	12
2-4 我们都是看别人的程序成长起来的	13
2-5 保护方法一：数据 CHECKSUM 法	13
2-6 保护方法二：SRAM 数据保护法	13
2-7 保护方法三：PAL/GAL/PEEL 保护法	14
2-8 保护方法四：FPGA 保护法	15
2-9 保护方法五：订做一个 CPU	15
2-10 保护方法六：掩人耳目 REMARK 芯片	16
2-11 保护方法七：双 CPU 联机保护法	16
2-12 保护方法八：程序加入大量垃圾数据	16
2-13 保护方法九：隐藏式地雷保护法	17
2-14 程序高手与解题高手	17
第 3 章 8051 程序的逆向工程	20
3-1 善用 EPROM 烧录器的上传与下载功能	20
3-2 所有的设计都是从模仿开始	27
第 4 章 实验桌上的反思	30
4-1 电解电容的爆炸	30
4-2 EPROM 烧录器之后	31
4-3 如何成为单片机专业人士	34
4-4 有认真的读者才有用心的作者	36
第 5 章 C 语言的导入：SDCC	40
5-1 使用 C 语言学习 8051	40
5-2 如何获取 C	41
5-3 SDCC 操作程序	43
5-4 C 编译后所产生文件	47
5-5 学了 C 以后	62

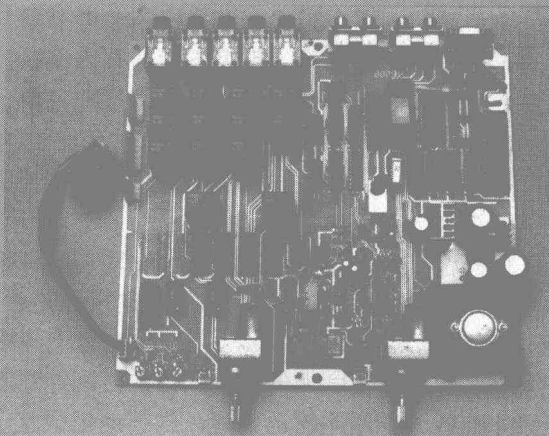
<b>第 6 章</b>	<b>8051 的是是非非</b> .....	66
6-1	8051 的众多优势.....	66
6-2	8051 的缺憾.....	69
6-3	市面上常见的 8051 CPU 变种.....	71
6-4	8051 另类观点剖析 (本节内容由凌全伯先生撰写).....	73
<b>第 7 章</b>	<b>单片机新成员 AT89C51 介绍</b> .....	82
7-1	AT89C51 内含 4KB 闪存 FLASH MEMORY.....	82
7-2	IDLE 与 POWER DOWN 模式.....	84
7-3	如何设计 AT89C51 内部的闪存.....	84
7-4	AT89C51 烧录器的使用.....	86
<b>第 8 章</b>	<b>8051 单片机新成员 AT89S8252 介绍</b> .....	90
8-1	新 CPU——AT89S8252 的尝试.....	90
<b>第 9 章</b>	<b>自制的 89C51 烧录器</b> .....	96
9-1	烧录线路分析.....	97
9-2	DIY 自己装步骤.....	98
9-3	烧录器的基本功能测试.....	100
9-4	AT89C51 烧录器的使用.....	103
9-5	烧录程序分析.....	106
9-6	FLAG51 存储器 RAM 的扩展.....	106
<b>第 10 章</b>	<b>AT89C2051 烧录器的程序修改</b> .....	112
10-1	星期一: 烧录资料研究.....	112
10-2	星期二: 线路修改.....	113
10-3	星期三: 程序加入及验证.....	115
10-4	星期四: 波形观察及烧录.....	116
10-5	星期五: 开始正式烧录.....	117
10-6	星期六: 寿命测试.....	118
10-7	星期日: 好戏上场.....	118
<b>第 11 章</b>	<b>Flash 编程器的后续开发</b> .....	122
11-1	AT89C51 会取代 8751 吗.....	122
11-2	AT89C52 已经上市了.....	122
11-3	AT89C51 烧录机的再修改.....	124
11-4	烧录程序也可用 C 语言来处理.....	126
11-5	烧录程序的最新版本.....	126
<b>第 12 章</b>	<b>揭开 EPROM 烧录的秘密</b> .....	130
12-1	EPROM 烧录的方法.....	130
12-2	EPROM 烧录线路的安排.....	132
12-3	知其然, 再知其所以然.....	136
12-4	EPROM 烧录软件的功能.....	137
12-5	跟烧录时间赛跑.....	138
12-6	自己装的您有福了.....	138



<b>第 13 章</b>	<b>EPROM 烧录器的组装步骤</b> .....	144
13-1	EPROM 烧录前您还需要哪些设备.....	145
13-2	EPROM 烧录器的 DIY 步骤.....	145
13-3	EPROM 烧录器调整步骤.....	146
13-4	EPROM 烧录器的使用.....	152
13-5	EPROM 烧录板上各个功能键的说明.....	152
13-6	个人计算机联机时可使用的命令.....	152
<b>第 14 章</b>	<b>华邦 E<sup>2</sup>PROM W27E512 烧录器</b> .....	156
14-1	EPROM 与 E <sup>2</sup> PROM 的差异.....	156
14-2	E <sup>2</sup> PROM 烧录线路的探讨与修正.....	157
14-3	E <sup>2</sup> PROM 烧录软件的修正.....	158
14-4	E <sup>2</sup> PROM 烧录板的修改步骤.....	162
14-5	W27E512 的使用时机.....	164
<b>第 15 章</b>	<b>IC 封装机的修改与规划</b> .....	168
15-1	IC 封装机, 事实上就是精密的塑胶射出成型机.....	168
15-2	初步规划.....	169
15-3	细部设计.....	171
15-4	系统测试.....	173
15-5	试用结果.....	173
15-6	结论.....	174
<b>第 16 章</b>	<b>自动化电饭锅测试线设计</b> .....	176
16-1	自动化电饭锅测试线的由来.....	176
16-2	测试流程的安排.....	176
16-3	测试站的局部设计.....	177
16-4	进行模拟测试.....	178
16-5	现场实地测试.....	179
16-6	结论.....	179
<b>第 17 章</b>	<b>橡胶加硫机的设备改善</b> .....	182
<b>第 18 章</b>	<b>经验谈: 鱼跃龙门的思索</b> .....	188
18-1	“证书”不等于“保证就业”.....	188
18-2	企业需要会思考的信息人员, 而非证照合格人员.....	188
18-3	信息教育应该做适度的调整.....	189
18-4	期盼 IT 业生力军的加入.....	189
<b>第 19 章</b>	<b>套装软件的诚信问题</b> .....	192
19-1	好软件应该内外都吸引人.....	192
19-2	中文化的程度就要看原厂的態度了.....	193
19-3	代理商最不愿意看到的英文字: TERMINATE.....	194
<b>第 20 章</b>	<b>WHO CARE</b> .....	196
20-1	捷运系统的百分之百安全才通车.....	196
20-2	全民医保计算机化无限商机变成了无限修改.....	196

20-3	高速公路的低速自动投币收费系统 .....	197
20-4	春节前的铁路语音订票系统竟然死机 .....	197
20-5	Internet 上技术询问信函的泛滥 .....	197
<b>第 21 章</b>	<b>几个深刻经验与教训 .....</b>	<b>200</b>
21-1	经验 1: 弹尽援绝——智能刺绣机的开发 .....	200
21-2	经验 2: 尚未上演就下台——电容电感的质检自动化 .....	202
21-3	经验 3: 开机状态的困扰——电表智能化测量 .....	204
<b>第 22 章</b>	<b>旗威上网了 .....</b>	<b>208</b>
22-1	旗威技术交流网诞生了 .....	208
22-2	PDF 文件是跨平台的 .....	209
22-3	E-mail 处理及回复的原则 .....	211
<b>第 23 章</b>	<b>当然你也可以做到 .....</b>	<b>214</b>
23-1	北部与南部理应无技术上的差距 .....	214
23-2	马上学习单片机都来得及 .....	215
<b>第 24 章</b>	<b>自主性研究的重要 .....</b>	<b>218</b>
24-1	企业引进技术所浮现的问题 .....	218
24-2	技术是长年的积累, 别人是学不来的 .....	218
24-3	研究与创意的始源在于教育 .....	219
24-4	提升技术最后还是在于自己 .....	220
<b>附录</b>	<b>.....</b>	<b>221</b>
附录 A	ASCII 表 .....	221
附录 B	8051 指令集总整理 .....	223
附录 C	8051 指令整理 (按功能划分) .....	233
附录 D	8051 指令整理 (按十六进制排列) .....	238
附录 E	8051 SFR 表与 RESET 后的初始值 .....	247
附录 F	SFR 特殊功能寄存器整理表 .....	248
附录 G	日文专业杂志的订购 .....	250
附录 H	PRO 族: 善用因特网上的各种 BBS .....	251
附录 I	DIS51 的深度应用 .....	253
附录 J	一张照片一个故事 .....	256

# 1



KTV 使用的点歌与视频控制器，含红外线遥控及音量控制，右侧上方为 8052 单片机。

# 第 1 章 8051 的汇编语言与 C 语言

有些程序设计人员对汇编语言说“恨之入骨”也不为过，习惯用 C 语言来设计系统。其实 C 语言和汇编语言各有其应用的空间，当你需要程序要精简且速度超快时，汇编语言绝对是首要之选。选择用 C 来开发就一帆风顺了吗？那也未必，欲知详情，请看本章彻底的分析。

最近，在 BBS 上的 Electronic（即关于电子设备的）板上经常看到这样的讨论：8051 的程序开发，是选用汇编语言较好，还是使用 C 语言较好？两种语言的支持者都引经据典并结合自己的实践经验为自己辩解。其实双方的说法都有道理。笔者在汇编语言与 C 语言的应用开发上都有超过 10 年以上的实践经验，愿意把自己从书本上看不到的体验与您分享。本章最终的目的并不是断定哪种语言是最好的，但是会引导读者根据自身情况以及开发的价格选用最适合自己的 8051 开发语言。

## 1-1 汇编语言短小精干

从学生时代开始我就使用汇编语言来写硬件的控制程序，最初是 LED 的灯号控制及七段显示器的数值显示，到多轴步进电动机的实时控制，都是借用强而有力的汇编语言直接控制 CPU 的运行。早期设有 PC（个人计算机）及 Apple II 微电脑的时代，我是使用所谓的微电脑学习机来学习汇编语言的。首先我把汇编程序写在记事本的右方，然后由微电脑的 DATABOOK 上查找该指令的机器码（MACHINE CODE），将此机器码写在记事本的原始程序左侧，再将此机器码以十六进制码的方式一个接一个地输入到微电脑学习机上，最后一个操作是按下学习机上的“GO”键开始执行我们先前辛苦输入的程序。以上所有的操作都是手动的，除错的步骤当然也是纯手动的。那个时代有许多好的参考书籍与资料，反倒是现在资料变少了，想自己动手做的人也成了稀有品种。程序不幸若有错误时，所有的步骤都要重来一次，这也促使我们养成每个步骤都做双重核对的习惯，而且把每个子程序都模块化，以便其他场合也可派上用场，因为写汇编语言是相当艰苦的，有些程序上的臭虫（bug）要几星期的奋斗才可以确认出来，有些不应该出错的地方竟然出错了，这些苦头只有曾经亲自尝试过的人才有可能体会。

个人计算机流行之后，写程序就没这么累了，程序的编写与编译都可以在计算机上做，剩下的操作就是将程序下传给待测的控制板，然后确认所有的操作是否都如同原先所设置的。每次打印汇编程序的报表时，数据长度超过 20 页是很正常的，还有许多程序都是数百页以上的规模。我们的经验是汇编语言的程序长度加倍时，除错的时间绝对超过原来的两倍。开发这些程序的同时，我都会另外准备一本厚厚的记事簿，详细地记载整个开发的过程以及修改的地方，当然也包括了所有的测试步骤及方式。在原始程序 SOURCE PROGRAM 上我们也做了详尽的操作说明。

有人做过一个统计：开发汇编语言程序时，每 20 行大约需要 1 man-day 的成本；这也就是说：一个有 1000 行的汇编程序，若交由一个有经验的工程师来处理，至少要花上 50 天的时间才能完成。一个人一天只能完成 20 行的程序，好像有点不可思议！但是依我们多年的经验来看，这个说法在写汇编语言的场合中确实是对的，程序越大时这个说法越有其正确性。

由于汇编语言是直接控制系统的所有输入/输出点及存储器，所以刚开始写程序时，经常会碰到系统突然死在某个循环中，无法跳出该循环的状态，最后只好采用最直接且最有效的方法——关机重来。写汇编语言对程序设计师来讲，是有相当的杀伤力及折磨性的，每次新的程序开始时，所有的程序小环节都要逐一测试与验证，所以压力是一定有的，完成一个程序后就好像历经一次母亲怀胎到生产的过程一样。写了多个大的汇编程序后，若无法适度调整心态及善用辅助的开发工具，很可能从此就永远不碰汇编程序了。真的，何苦来哉！

## 1-2 C 语言可以缩短开发时间

在 C 语言成功地由 UNIX 移转到 PC 后，写汇编语言的梦魇终于解除了，接着我们也可以看到适合各种 CPU 的 C 语言开发系统，其价格从几百元到上万元都有，这些开发软件工具的广告可在国外知名的专业杂志上看到，当然国内的 CPU 在线仿真器 (In Circuit Emulator) 开发厂商也会依国内用户的要求进口部分产品，但是数量不多，而且这些厂商对这方面的专业知识也是严重缺乏，所以凡事只有靠自己了。

我们先学习的是 PC 上的 C 语言程序 (Microsoft C 及 Turbo C)，这段学习时间至少持续一年以上，再进而购入 2500AD 的 8051C 语言编译程序，将我们原先的汇编程序改用 C 程序来处理，这时我们才发觉一个简单的控制程序若用 C 语言来撰写，要限时在 7 天以内完成是非常有可能的！举一个简单的例子来说：若要让 8051 的 P1 端口产生类似指示灯的 LED 亮法，用汇编语言的写法大概是这样的：

程序 1 用汇编语言写成的指示灯程序

```

                                ORG     0000H
START  NOV                      R1, #00H           ;加入这段小 DELAY
$1                                DJNZ   R1, $1           ;等待系统稳定后再开始
                                MOV    SP, #60H        ;STACK 值的设置
LOOP   MOV    A, #01H
LOOP_NMOV  P1, A
                                CALL   DELAY
                                RL     A
                                CJNE  A, #80H, LOOP_N
                                SJMP  LOOP
;
;延迟一小段时间，以便可以观看出其变化
DELAY  MOV    R0, #00H
$1     MOV    R1, #00H
$2     DJNZ  R1, $2
                                DJNZ  R0, $1
                                RET

```

若改用 2500AD 的 C 语言来写时，可就轻松多了。看起来简单明了，何时看程序都是一清二楚的。唯一的缺憾是用 C 转换出来的程序代码过大，下面这个程序链接了链接库后，最

后的程序空间可能会超过 600 字节。

### 程序 2 改用 C 来处理的指示灯程序

```
#include "c8051io.h"
#include "c8051sr.h"
main()
{
    car out;
    out=0x01;
    while(1)
    {
        P1=out;
        out=out<<1; /*shift left*/
        delay();
        if(out==0x80)out=0x01;
    }
}

int delay()
{
    int m;
    for(m=0;m<1000;m++){}; /*just delay,do nothing*/
}
```

以下就是上述 C 语言程序经过转译成 8051 汇编语言的程序，由于篇幅的限制，我们仅列出程序重要的部分，其余的声明部分都暂时删除，这是由 2500AD 公司的 8051 C 编译程序转译的结果，改用其他编译器时结果可能不会相同。

```
32 0000          _main:          .equal $
33              ;
34              ;
35 0000          ?ASC0:          .equal 0
36 0000          ?TSC0:          .equal 0
37 0001          ?LSC0:          .equal 1
38 0000          C2 AF          clr     ea
39 0002          EE            mov     a,r6
40 0003          24 FF          add     a,#.low.-?TSC0-?LSC0
41 0005          FE            mov     r6,a
42 0006          EF            mov     a,r7
43 0007          34 FF          addc   a,#.high.-?TSC0-?LSC0
44 0009          FF            mov     r7,a
45 000A          D2 AF          setb   ea
46              ;out=0x01
47 000C          90 00 00      mov     dptr,#-1+?LSC0
48 000F          74 01          mov     a,#1
49 0011          12 00 00      lcall  st_oper1_stack_char
50              ;while
51 0014          ?TOL0:          .equal $
52 ;(1)
53 ;
54 0014          79 00          mov     r1,#0
```

```

55 0016      74 01          mov     a,#1
56 0018      C3           clr     c
57 0019      99           subb   a,r1
58 001A      70 03      jnz    +3
59 001C      02 00 55    ljmp   ?BOL0
60 001F      ?PLC0:      .equal $
61          ;P1=out
62 001F      90 00 00    mov    dptr,#-1+?LSC0
63 0022      12 00 00    lcall  __ld_oper1_stack_char
64 0025      F5 00      mov    .low._P1,a
65          ;out=out<<1
66 0027      79 01      mov    r1,#1
67 0029      90 00 00    mov    dptr,#-1+?LSC0
68 002C      12 00 00    lcall  __ld_oper1_stack_char
69 002F      12 00 00    lcall  __sl_char
70 0032      90 00 00    mov    dptr,#-1+?LSC0
71 0035      12 00 00    lcall  __st_oper1_stack_char
72          ;delay()
73 0038      12 00 62    lcall_delay
74          ;(out==0x80)
75 003B      79 80      mov    r1,#128
76 003D      90 00 00    mov    dptr,#-1+?LSC0
77 0040      12 00 00    lcall  __ld_oper1_stack_char
78 0043      C3           clr    c
79 0044      99           subba,r1
80 0045      60 03      jz     +3
81 0047      02 00 52    ljmp   ?BOL1
82 004A      ?PLC1:      .equal $
83          ;out=0x01
84 004A      90 00 00    mov    dptr,#-1+?LSC0
85 004D      74 01      mov    a,#1
86 004F      12 00 00    lcall  __st_oper1_stack_char
87  ;;
88          ;
89          ;
90 0052      ?BOL1:      .equal $
91 0052      02 00 14    ljmp   ?TOL0
92 0055      ?BOL0:      .equal $
93          ;
94 0055      ?BOF0:      .equal $
95 0055      C2 AF      clr    ea
96 0057      EE           mov    a,r6
97 0058      24 01      add    a,#.low.?LSC0+?TSC0+0
98 005A      FE           mov    r6,a
99 005B      EF           mov    a,r7
100 005C      34 00      add    ca,#.high.?LSC0+?TSC0+0
101 005E      FF           mov    r7,a
102 005F      D2 AF      setb  ea
103 0061      22           ret
104          ;

```

## 第 1 章

```

105          0062      _delay:.equal $
106          ;
107
108          0000      ?ASC1:      .equal 0
109          0000      ?TSC1:      .equal 0
110          0002      ?LSC1:      .equal 2
111 0062      C2 AF          clr ea
112 0064      EE          mov     a,r6
113 0065      24 FE          add a,#.low.-?TSC1-?LSC1
114 0067      FE          mov     r6,a
115 0068      EF          mov     a,r7
116 0069      34 FF          addc  a,#.high.-?TSC1-?LSC1
117 006B      FF          mov     r7,a
118 006C      D2 AF          setb  ea
119          ;m=0;
120 006E      90 00 00      mov     dptr,#-2+?LSC1
121 0071      78 00          mov r0,#.low.0
122 0073      7A 00          mov r2,#.high.0
123 0075      12 00 00      lcall _st_oper1_stack_int
124 0078      ?FLC2:      .equal $
125          ;m<1000;
126 0078      79 E8          mov r1,#.low.1000
127 007A      7B 03          mov r3,#.high.1000
128 007C      90 00 00      mov     dptr,#-2+?LSC1
129 007F      12 00 00      lcall  _ld_oper1_stack_int
130 0082      12 00 00      lcall  __cmp_int
131 0085      30 E7 06      jnb   acc.7,+6
132 0088      30 D2 09      jnb   ov,+9
133 008B      02 00 B1      ljmp          ?BOL2
134 008E      20 D2 03      jb    ov,+3
135 0091      02 00 B1      ljmp          ?BOL2
136 0094      02 00 AE      ljmp          ?FLB2
137 0097      ?FLA2:      .equal $
138 ;m++)
139 0097      90 00 00      mov     dptr,#-2+?LSC1
140 009A      12 00 00      lcall  __ld_oper1_stack_int
141 009D      E8          mov     a,r0
142 009E      24 01          add    a,#1
143 00A0      F8          mov     r0,a
144 00A1      EA          mov     a,r2
145 00A2      34 00          addc   a,#0
146 00A4      FA          mov     r2,a
147 00A5      90 00 00      mov     dptr,#-2+?LSC1
148 00A8      12 00 00      lcall  __st_oper1_stack_int
149 00AB      02 00 78      ljmp          ?FLC2
150 00AE      ?FLB2:      .equal $
151          ;
152 00AE      02 00 97      ljmp          ?FLA2
153 00B1      ?BOL2:      .equal $

```



```

154                                     ;
155 00B1                                ?BOF1:    .equal $
156 00B1      C2 AF                      clr ea
157 00B3      EE                          mov    a,r6
158 00B4      24 02                      add a,#.low.?LSC1+?TSC1+0
159 00B6      FE                          mov    r6,a
160 00B7      EF                          mov    a,r7
161 00B8      34 00                      add ca,#.high.?LSC1+?TSC1+0
162 00BA      FF                          mov    r7,a
163 00BE      D2 AF                      setb   ea
164 00BD      22                          ret

```

### 1-3 8051 编译器使用经验谈

写 8051 的汇编程序时是完全无法取巧的。如果刚开始学习 8051 的程序语言，我们强烈建议由汇编语言开始学起，因为只有这样才能促使学习者完全理解整个 8051 CPU 动作的来龙去脉。不会设计线路也不要紧，因为市面上 8051 书籍多得很，只要参考几种现成的线路后，就可以依样照做了，可是汇编程序的撰写，除了自己以外，别人都无法帮得上！我们的经验是别人的程序看得越多，你的程序架构就会写得越好。如果有好的写法或子程序也可以在自己充分地吸收后归为己有。如果身旁收集的工具程序很多，程序开发的时间会相对地减少许多，不需要每次都从头开始除错。

市面上的 8051 参考书籍适合初学者的居多，可能无法找到相当理想的参考程序，我们建议可以到旗威科技公司的网站 (<http://www.chipware.com.tw>) 上获取以下几个相当有参考价值的参考程序：

readtemp.asm 温度的测量显示与联机串行通信程序。  
flagdisp.asm 七段显示器的控制程序。

以上程序除了包含有许多常用的汇编语言子程序外，都有使用定时中断及串行通信等等的使用范例，必须逐一推敲所有的程序，才能对控制系统有更进一步的理解与认识。这些学习的经历必须自己走过才算是把 8051 给带进门来。也可以把许多 8051 应用产品的程序 EPROM 拔下来，用 EPROM 烧录器将程序数据转存在 PC 机中，然后用 8051 的反汇编程序 DIS51.EXE（请自行到旗威科技公司的网站中下载此程序）将原先的机器码翻译成汇编语言的格式，再来分析其程序的写法与控制模式。我们所认识的汇编语言高手中，有不少人是用这种方式去磨练的，当然也包括我个人在内。这种学习方法最辛苦，但是收获却是最大的。我曾经用最原始的方法读过多个 Z80、6502 及 8051 满载 64KB 的系统程序，所以现在碰到类似的系统时，都可以在最短的时间内了解问题所在。至于旗威科技公司开发的其他 8051 应用程序，请不要全面予以反汇编，因为这些程序有大多数是用 8051 的 C 语言来处理的，程序规模相当大，全部通过堆栈传递参数，没有相当的程序基础是完全无法了解的。

我们另一方面也建议：如果有多余预算的话，不一定非得买 ICE 不可，但是买一台较像样的示波器，绝对可以在 8051 程序的开发过中帮上大忙。以前我们只用一台 20MHz 的 Hitachi（日立）示波器，再通过汇编程序中的无穷循环，就可以对硬件线路及系统程序进行全面性的除错。近几年来我们发现若用数字存储示波器 DSO（Digital Storage Oscilloscope）来除错更是方便，尤其是串行通信及 AD 转换的程序除错，至少可以减少一半以上的时间。